IEC 61499-1

Edition 2.0 2012-11

# INTERNATIONAL STANDARD

# NORME INTERNATIONALE

colour inside

## Function blocks – Part 1: Architecture

## Blocs fonctionnels – Partie 1: Architecture





IEC 61499-1

Edition 2.0 2012-11

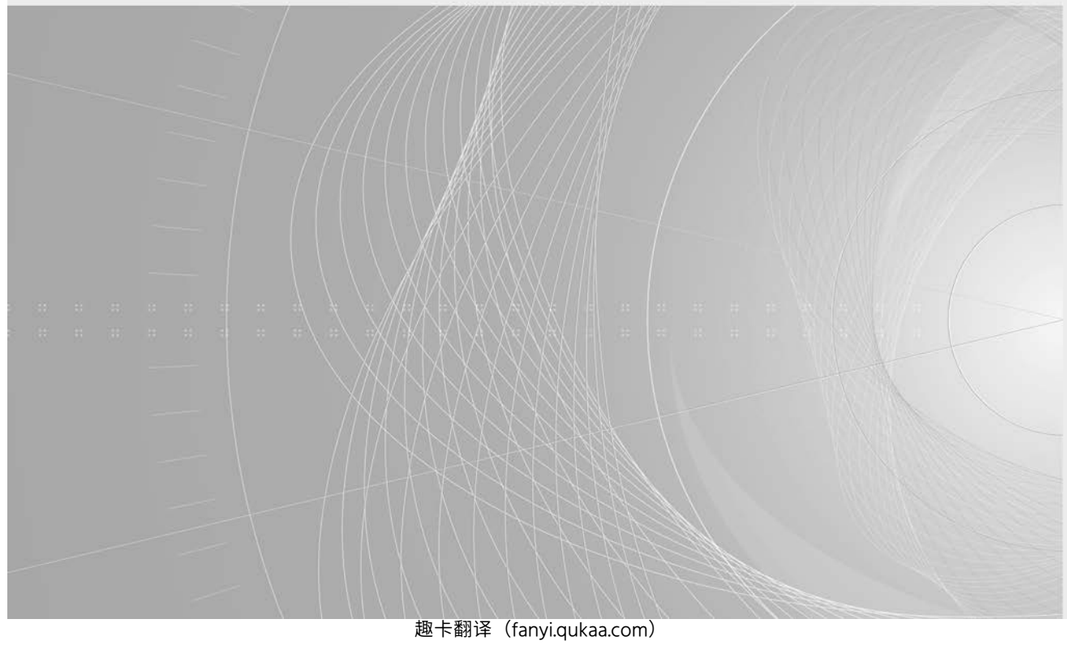
INTERNATIONAL STANDARD

NORME INTERNATIONALE

里面的颜色

## 功能块 第1部分：架构

## Blocsfonctionnels 第1部分：架构



IEC 61499-1:2012

IEC Central Office
3, rue de Varembé
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

## About the IEC

The International Electrotechnical Commission (IEC) is the leading global organization that prepares and publishes International Standards for all electrical, electronic and related technologies.

## About IEC publications

The technical content of IEC publications is kept under constant review by the IEC. Please make sure that you have the latest edition, a corrigenda or an amendment might have been published.

**Useful links:**

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,…).
It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available on-line and also once a month by email.

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

## A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

## A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

**Liens utiles:**

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,…).
Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.





IEC中心办公室

3 ruedeVarembé
20

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

## 关于IEC

国际电工委员会(IEC)是全球领先的组织，负责编写和发布
所有电气、电子和相关技术的国际标准。

## 关于IEC出版物

IEC出版物的技术内容由IEC不断审查。请确保您拥有最新版本，可能已经发布了勘误或修正。

IEC出版物搜索www.iec.chsearchpub

高级搜索使您能够按各种标准（参考号、文本、技术委员会……）查找IEC出版物。它还提供有关项目、替换和撤回的出版物的信息。

世界领先的电子和电气术语在线词典，包含30 000多个英语和法语术语和定义，以及其他语言的等效术语。也称为国际电工词汇（IEV）在线。

IEC刚刚发布webstore.iec.ch刚刚发布

随时了解所有新的IEC出版物。刚刚发布详细介绍了所有已发布的新出版物。可在线获取，也可通过电子邮件每月一次。

客户服务中心webstore.iec.chcsc

如果您想就本出版物向我们提供反馈或需要进一步帮助，请联系客户服务中心：csc@iec.ch。

## 关于IEC

国际电工委员会(IEC)是世界领先的开发和出版
与电力、电子和相关技术相关的所有事物的国际标准。

## 关于IEC出版物

IEC出版物的技术内容不断被审查。请确保您拥有最新版本，可能已发布更正或修正。

搜索IEC出版物www.iec.chsearchpub

高级搜索允许您使用不同的标准（参考编号、文本、技术委员会等）查找IEC出版物。它还提供有关项目和被取代或撤回的出版物的信息。

世界上第一部电子电气术语在线词典。它包含超过30 000个英语和法语术语和定义，以及其他语言的等效术语。一样

刚刚发布IECwebstore.iec.ch刚刚发布

随时了解新的IEC出版物。刚刚出版详细介绍了新出版的出

可在线获取，也可通过电子邮件每月一次。

国际(IEV)在线。
客户服务webstore.iec.chcsc

如果您希望对本出版物发表意见或有任何疑问，请联系我们：csc@iec.ch。

IEC 61499-1

Edition 2.0 2012-11

# INTERNATIONAL STANDARD

# NORME INTERNATIONALE

colour inside

## Function blocks – Part 1: Architecture

## Blocs fonctionnels – Partie 1: Architecture

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

PRICE CODE XF
CODE PRIX

ICS 25.040; 35.240.50

ISBN 978-2-83220-481-8

**Warning! Make sure that you obtained this publication from an authorized distributor.**
**Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**





IEC 61499-1

Edition 2.0 2012-11

# INTERNATIONAL STANDARD

# NORME INTERNATIONALE

里面的颜色

## 功能块 第1部分：架构

## 构建块–第1部分：架构

INTERNATIONAL ELECTROTECHNICAL COMMISSION

COMMISSION ELECTROTECHNIQUE INTERNATIONALE

价格代码 XF
代码价格

ICS 25.040; 35.240.50

ISBN 978-2-83220-481-8

**警告！确保您从授权经销商处获得此出版物。**
**注意力！请确保您已通过授权经销商获得本出版物。**

CONTENTS





– 2 – 61499-1 © IEC:2012

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

# FUNCTION BLOCKS –

# Part 1: Architecture

## FOREWORD

1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as "IEC Publication(s)"). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.

2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.

3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.

5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.

6) All users should ensure that they have the latest edition of this publication.

7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.

8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.

9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

International Standard IEC 61499-1 has been prepared by subcommittee 65B: Measurement and control devices, of IEC technical committee 65: Industrial-process measurement, control and automation.

This second edition cancels and replaces the first edition published in 2005. This edition constitutes a technical revision.

This edition includes the following significant technical changes with respect to the previous edition:

- *Execution control* in basic function blocks (5.2) has been clarified and extended:
  - Dynamic and static parts of the EC transition condition are clearly delineated by using the `ec_transition_event[guard_condition]` syntax of the Unified Modeling Language (UML) (5.2.1.3, B.2.1).
  - The terminology "crossing of an EC transition" (3.10) is used preferentially to "clearing" to avoid the misinterpretation that the entire transition condition corresponds to a Boolean variable that can be "cleared."





国际电工委员会_____

# 功能块——

# 第1部分：架构

1)国际电工委员会(IEC)是一个由所有国家电工委员会(IECNationalCommittees)组成的全球标准化组织。IEC的目标是促进有关电气和电子领域标准化的所有问题的国际合作。为此，除其他活动外，IEC还发布了国际标准、技术规范、技术报告、公开可用规范(PAS)和指南（以下简称"IEC出版物"）。他们的准备工作委托给技术委员会；任何对所处理主题感兴趣的IEC国家委员会都可以参与这项准备工作。与IEC联络的国际、政府和非政府组织也参与了这项准备工作。IEC与国际标准化组织(ISO)根据两个组织之间的协议确定的条件密切合作。

2)由于每个技术委员会都有来自所有感兴趣的IEC国家委员会的代表，因此IEC关于技术问题的正式决定或协议尽可能地表达了对相关主题的国际意见共识。

3)IEC出版物具有国际使用的推荐形式，并在这个意义上被IEC国家委员会接受。尽管已尽一切合理努力确保IEC出版物的技术内容准确无误，但IEC不对它们的使用方式或任何最终用户的任何误解负责。

4)为了促进国际统一，IEC国家委员会承诺在其国家和地区出版物中尽可能透明地应用IEC出版物。任何IEC出版物与相应的国家或地区出版物之间的任何分歧都应在后者中明确指出。

5)IEC本身不提供任何符合性证明。独立认证机构提供合格评定服务，并在某些领域提供IEC合格标志。IEC不对独立认证机构提供的任何服务负责。

6)所有用户应确保他们拥有本出版物的最新版本。

7)对于任何人身伤害、财产损失或其他任何性质的直接或间接损失，IEC或其董事、雇员、雇员或代理人（包括个别专家及其技术委员会和IEC国家委员会的成员）不承担任何责任，或因出版、使用或依赖本IEC出版物或任何其他IEC出版物而产生的成本（包括法律费用）和开支。

8)请注意本出版物中引用的规范性参考文献。使用引用的出版物对于正确应用本出版物是必不可少的。

9)提请注意本IEC出版物的某些元素可能是专利权的主题。IEC不负责识别任何或所有此类专利权。

国际标准IEC61499-1由IEC技术委员会65：工业过程测量、控制和自动化的分委员会65B：测量和控制设备制定。

第二版取消并代替2005年出版的第一版。本版构成技术修订。

此版本相对于上一版本包括以下重大技术更改：

- 基本功能块(5.2)中的执行控制已得到澄清和扩展：
  - EC转换条件的动态和静态部分使用统一建模语言(UML)(5.2.1.3 B.2.1)的ec_transition_event[guard_condition]语法清楚地描述。
  - 术语"EC转换的交叉"（3.10）优先用于"清除"，以避免误解整个转换条件对应于可以"清除"的布尔变量。

– Operation of the ECC state machine in 5.2.2.2 has been clarified and made more rigorous.

– Event and data outputs of adapter instances (plugs and sockets) can be used in EC transition conditions, and event inputs of adapter instances can be used as EC action outputs.

- *Temporary variables* (3.97) can be declared (B.2.1) and used in algorithms of basic function blocks.
- *Service sequences* (6.1.3) can now be defined for basic and composite function block types and adapter types, as well as service interface types.
- The syntax for *mapping* of FB instances from applications to resources has been simplified (Clause B.3).
- Syntax for definition of *segment types* (7.2.3) for network segments of system configurations has been added (Clause B.3).
- Function block types for interoperation with programmable controllers are defined (Clause D.6).
- The READ/WRITE management commands (Table 8) now apply only to *parameters*.

The text of this part of IEC 61499 is based on the following documents:

| FDIS | Report on voting |
|---|---|
| 65B/845/FDIS | 65B/855/RVD |

Full information on the voting for the approval of this standard can be found in the report on voting indicated in the above table (when voting is completed).

This publication has been drafted in accordance with the ISO/IEC Directives, Part 2.

A list of all parts of the IEC 61499 series can be found, under the general title *Function blocks*, on the IEC website.

Terms used throughout this International Standard that have been defined in Clause 3 appear in *italics*.

The committee has decided that the contents of this publication will remain unchanged until the stability date indicated on the IEC web site under "http://webstore.iec.ch" in the data related to the specific publication. At this date, the publication will be

- reconfirmed,
- withdrawn,
- replaced by a revised edition, or
- amended.

**IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.**



– 5.2.2.2中ECC状态机的操作已得到澄清并变得更加严格。

– 适配器实例（插头和插座）的事件和数据输出可用于EC转换条件，适配器实例的事件输入可用作EC操作输出。

- 临时变量(3.97)可以被声明(B.2.1)并用于基本功能块的算法。
- 现在可以为基本和复合功能块类型和适配器类型以及服务接口类型定义服务序列(6.1.3)。
- 将FB实例从应用程序映射到资源的语法已被简化（条款B.3）。
- 为系统配置的网段添加了段类型定义的语法（7.2.3）（条款B.3）。
- 定义了与可编程控制器互操作的功能块类型（条款D.6）。

READWRITE管理命令（表8）现在仅适用于参数。

IEC61499本部分的文本基于以下文件：

| FDIS | 投票报告 |
|---|---|
| | |

批准本标准的全部表决情况见上表所示的表决报告（表决完成时）。

本出版物是根据ISOIEC指令第2部分起草的。

IEC61499系列所有部分的列表可以在IEC网站上的通用标题功能块下找到。

本国际标准中第3条中定义的术语以斜体显示。

委员会已决定，该出版物的内容将保持不变，直到IEC网站“http:webstore.iec.ch”下与特定出版物相关的数据中指明的稳定日期为止。届时，该刊物将

-
-
- 被修订版取代，或
- amended.

**重要信息——本出版物封面上的“内部颜色”标志表明它包含被认为有助于正确理解其内容的颜色。因此，用户应使用彩色打印机打印此文档。**

## INTRODUCTION

IEC 61499 consists of the following parts, under the general title *Function blocks*:

- Part 1 (this document) contains:
  - general requirements, including scope, normative references, definitions, and reference models;
  - rules for the declaration of *function block types*, and rules for the behavior of *instances* of the types so declared;
  - rules for the use of function blocks in the *configuration* of distributed industrial-process measurement and control *systems* (IPMCSs);
  - rules for the use of function blocks in meeting the communication requirements of distributed IPMCSs;
  - rules for the use of function blocks in the management of *applications, resources* and *devices* in distributed IPMCSs.
- Part 2 defines requirements for *software tools* to support the following systems engineering tasks:
  - the specification of *function block types*;
  - the functional specification of *resource types* and *device types*;
  - the specification, analysis, and validation of distributed IPMCSs;
  - the *configuration*, *implementation*, operation, and maintenance of distributed IPMCSs;
  - the exchange of *information* among *software tools*.
- Part 3 (Tutorial information) has been withdrawn due to the widespread current availability of tutorial and educational materials regarding IEC 61499. However, an updated 2nd Edition of Part 3 may be developed in the future.
- Part 4 defines rules for the development of *compliance profiles* which specify the features of IEC 61499-1 and IEC 61499-2 to be implemented in order to promote the following attributes of IEC 61499-based systems, devices and software tools:
  - interoperability of devices from multiple suppliers;
  - portability of software between software tools of multiple suppliers; and
  - configurability of devices from multiple vendors by software tools of multiple suppliers.





IEC61499由以下部分组成，总标题为功能块：

- Par
  - 一般要求，包括范围、规范性参考、定义和参考模型；
  - 功能块类型的声明规则，以及如此声明的类型实例的行为规则；
  - 在分布式工业过程测量和控制系统(IPMCS)的配置中使用功能块的规则；
  - 满足分布式IPMCS通信要求的功能块使用规则；

    在分布式IPMCS中管理应用程序、资源和设备时使用功能块的规则。
- 第2部分定义了支持以下系统工程任务的软件工具的要求：
  - 功能块类型的规范；
  - 资源类型和设备类型的功能规范；
  - 分布式IPMCS的规范、分析和验证；
  - 分布式IPMCS的配置、实施、操作和维护；

    软件工具之间的信息交换。
- 由于有关IEC61499的教程和教育材料的广泛可用性，第3部分（教程信息）已被撤回。但是，未来可能会开发第3部分的更新的第2版。
- 第4部分定义了开发合规性配置文件的规则，其中指定了要实施的IEC61499-1和IEC61499-2的特性，以促进基于IEC61499的系统、设备和软件工具的以下属性：
  - 来自多个供应商的设备的互操作性；
  - 多个供应商的软件工具之间的软件可移植性；和
  - 通过多个供应商的软件工具可配置来自多个供应商的设备。

# FUNCTION BLOCKS –

# Part 1: Architecture

## 1 Scope

This part of IEC 61499 defines a generic architecture and presents guidelines for the use of *function blocks* in distributed industrial-process measurement and control systems (IPMCSs). This architecture is presented in terms of implementable reference *models*, textual syntax and graphical representations. These models, representations and syntax **can be used for**:

- the specification and standardization of *function block types*;
- the functional specification and standardization of system elements;
- the implementation independent specification, analysis, and validation of distributed IPMCSs;
- the *configuration*, *implementation*, operation, and maintenance of distributed IPMCSs;
- the exchange of *information* among *software tools* for the performance of the above *functions*.

This part of IEC 61499 does not restrict or specify the functional capabilities of IPMCSs or their system elements, except as such capabilities are represented using the elements defined herein. IEC 61499-4 addresses the extent to which the elements defined in this standard may be restricted by the functional capabilities of compliant systems, subsystems, and devices.

Part of the purpose of this standard is to provide reference models for the use of function blocks in other standards dealing with the support of the system life cycle, including system planning, design, implementation, validation, operation and maintenance. The models given in this standard are intended to be generic, domain independent and extensible to the definition and use of function blocks in other standards or for particular applications or application domains. It is intended that specifications written according to the rules given in this standard be concise, implementable, complete, unambiguous, and consistent.

NOTE 1 The provisions of this standard alone are not sufficient to ensure interoperability among devices of different vendors. Standards complying with this part of IEC 61499 can specify additional provisions to ensure such interoperability.

NOTE 2 Standards complying with this part of IEC 61499 can specify additional provisions to enable the performance of *system*, *device*, *resource* and *application* management *functions*.

## 2 Normative references

The following documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

IEC 61131-1, *Programmable controllers – Part 1: General*

IEC 61131-3:2003, *Programmable controllers – Part 3: Programming languages*

IEC/ISO 7498-1:1994, *Information technology – Open systems interconnection – Basic reference model: The basic model*





# 功能块——
# 第1部分：架构

IEC61499的这一部分定义了一个通用架构，并提出了在分布式工业过程测量和控制系统(IPMCS)中使用功能块的指南。该架构以可实现的参考模型、文本语法和图形表示形式呈现。这些模型、表示和语法可用于：

- 功能块类型的规范和标准化；
- 系统要素的功能规范和标准化；
- 分布式IPMCS的实施独立规范、分析和验证；
- 分布式IPMCS的配置、实施、操作和维护；
- 为执行上述功能的软件工具之间的信息交换.

IEC61499的这一部分不限制或指定IPMCS或其系统元素的功能能力，除非这些能力使用此处定义的元素来表示。IEC61499-4解决了本标准中定义的元素可能受到兼容系统、子系统和设备的功能能力限制的程度。

本标准的部分目的是为其他标准中处理系统生命周期支持的功能块的使用提供参考模型，包括系统规划、设计、实施、验证、操作和维护。本标准中给出的模型是通用的、独立于领域的，并且可扩展至其他标准或特定应用或应用领域中功能块的定义和使用。根据本标准给出的规则编写的规范旨在简洁、可实施、完整、明确和一致。

注1：仅本标准的规定不足以确保不同供应商的设备之间的互操作性。符合IEC61499的这一部分的标准可以指定额外的规定来确保这种互操作性。

注2符合IEC61499的这一部分的标准可以指定附加条款，以实现系统、设备、资源和应用程序管理功能的性能。

## 规范性参考

以下文件的全部或部分在本文件中被规范引用，对其应用是必不可少的。对于注明日期的参考文献，仅引用的版本适用。对于未注明日期的引用文件，引用文件的最新版本（包括任何修改）适用。

IEC61131-1，可编程控制器-第1部分：一般

IEC61131-3:2003，可编程控制器 第3部分：编程语言

IECISO7498-1:1994，信息技术 开放系统互连 基本参考模型：基本模型

ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

## 3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

NOTE Terms defined in Clause 3 are *italicized* where they appear in definitions and Notes to entry of other terms as well as throughout the body of the document.

**3.1**
**acceptor**
*function block instance* which provides a *socket adapter* of a defined *adapter interface type*

**3.2**
**adapter connection**
*connection* from a *plug adapter* to a *socket adapter* of the same *adapter interface type*, which carries the flows of *data* and *events* defined by the *adapter interface type*

**3.3**
**adapter interface type**
*type* which consists of the definition of a set of *event inputs, event outputs, data inputs*, and *data outputs*, and whose *instances* are *plug adapters* and *socket adapters*

**3.4**
**algorithm**
finite set of well-defined rules for the solution of a problem in a finite number of *operations*

**3.5**
**application**
*software functional unit* that is specific to the solution of a problem in industrial-process measurement and control

Note 1 to entry: An application can be distributed among *resources*, and might communicate with other applications.

**3.6**
**attribute**
property or characteristic of an *entity*, for instance, the version identifier of a *function block type* specification

**3.7**
**basic function block type**
*function block type* that cannot be decomposed into other function blocks and that utilizes an *execution control chart (ECC)* to control the *execution* of its *algorithms*

**3.8**
**bidirectional transaction**
*transaction* in which a request and possibly *data* are conveyed from an *requester* to a *responder*, and in which a response and possibly data are conveyed from the responder back to the requester



ISOIEC8824-1:2008，信息技术 抽象语法符号一(ASN.1)：
基本符号规范

ISOIEC10646:2003，信息技术 通用多字节编码字符
*Set*

## 术语和定义

就本文件而言，以下术语和定义适用。

注：第3条中定义的术语在出现在定义和其他术语输入注释以及整个文件正文中的位置用斜体表示。

3.1接受器功能块实例，它提供定义的适配器接口类型的套接字适配器

3.2适配器连接从插头适配器到具有相同适配器接口类型的插座适配器的连接，承载由适配器接口类型定义的数据流和事件流。

3.3适配器接口类型由定义一组事件输入、事件输出、数据输入和数据输出组成的类型，其实例为插头适配器和插座适配器

3.4algorithm有限集定义明确的规则，用于在有限次数的操作中解决问题

3.5应用软件功能单元，专门用于解决工业过程测量和控制中的问题

注1：一个应用程序可以分布在资源之间，并且可能与其他应用程序通信。

3.6attribute实体的属性或特性，例如，功能块类型规范的版本标识符

3.7基本功能块类型不能分解成其他功能块并利用执行控制图（ECC）来控制其算法执行的功能块类型

3.8双向事务请求和可能的数据从请求者传送到响应者的事务，其中响应和可能的数据从响应者传送回请求者

**3.9**
**character**
member of a set of elements that is used for the representation, organization, or control of *data*

3.9用于表示、组织或控制数据的一组元素的字符成员

**3.10**
**crossing**
clearing
<of an EC transition> *operation* by means of which control is passed from the predecessor *EC state* of an *EC transition* to its successor EC state

3.10crossclearing<ofanECtransition>操作，通过该操作将控制从EC转换的前导EC状态传递到其后继EC状态

Note 1 to entry: This operation consists of de-activation of the predecessor EC state, followed by activation of the successor EC state.

注1：该操作包括先行EC状态的去激活，然后是后继EC状态的激活。

**3.11**
**communication connection**
*connection* that utilizes the communication mapping function of one or more *resources* for the conveyance of *information*

3.11通信连接利用一种或多种资源的通信映射功能来传递信息的连接。

**3.12**
**communication function block**
*service interface function block* that represents the *interface* between an *application* and the communication mapping function of a *resource*

3.12通信功能块serviceinterfacefunctionblock表示应用程序和资源的通信映射功能之间的接口

**3.13**
**communication function block type**
*function block type* whose *instances* are *communication function blocks*

3.13通信功能块类型以通信功能块为实例的功能块类型

**3.14**
**component function block**
*function block instance* which is used in the specification of an *algorithm* of a *composite function block type*

3.14组件功能块functionblockinstance用于复合功能块类型算法规范的功能块实例

Note 1 to entry: A component function block can be of *basic*, *composite* or *service interface type*.

注1：组件功能块可以是基本、复合或服务接口类型。

**3.15**
**component subapplication**
*subapplication instance* that is used in the specification of a *subapplication type*

3.15组件subapplicationsubapplication用于规范子应用类型的实例

**3.16**
**composite function block type**
*function block type* whose *algorithms* and the control of their *execution* are expressed entirely in terms of interconnected *component function blocks*, *events*, and *variables*

3.16复合功能块类型功能块类型，其算法及其执行的控制完全用互连的组件功能块、事件和变量来表示

**3.17**
**concurrent**
pertaining to *algorithms* that are *executed* during a common period of time during which they may have to alternately share common *resources*

3.17并发指在一个公共时间段内执行的算法，在此期间它们可能不得不交替共享公共资源

**3.18**
**configuration (of a *system* or *device*)**
selecting *functional units*, assigning their locations and defining their interconnections

3.18配置（系统或设备的）选择功能单元，分配它们的位置并定义它们的互连

**3.19**
**configuration parameter**
*parameter* related to the *configuration* of a *system*, *device* or *resource*

3.19configurationparameter与系统、设备或资源的配置相关的参数

**3.20**
**confirm primitive**
*service primitive* which represents an interaction in which a *resource* indicates completion of some *algorithm* previously *invoked* by an interaction represented by a *request primitive*

3.20确认原语服务原语，表示一种交互，其中资源指示完成先前由请求原语表示的交互调用的某些算法

**3.21**
**connection**
association established between *functional units* for conveying *information*

3.21功能单元之间建立的连接关联，用于传递信息

**3.22**
**critical region**
*operation* or sequence of operations which is *executed* under the exclusive control of a locking object which is associated with the *data* on which the operations are performed

3.22临界区操作或操作序列，在与执行操作的数据相关联的锁定对象的排他控制下执行

**3.23**
**data**
reinterpretable representation of *information* in a formalized manner suitable for communication, interpretation or processing

**3.23**

*tion* in a formalized manner suitable for 沟通、解释或处理

**3.24**
**data connection**
association between two *function blocks* for the conveyance of *data*

3.24数据连接两个功能块之间用于数据传输的关联

**3.25**
**data input**
*interface* of a *function block* which receives *data* from a *data connection*

3.25数据输入接口从数据连接接收数据的功能块的数据输入接口

**3.26**
**data output**
*interface* of a *function block* which supplies *data* to a *data connection*

3.26数据输出接口向数据连接提供数据的功能块

**3.27**
**data type**
set of values together with a set of permitted *operations*

3.27数据类型一组值连同一组允许的操作

**3.28**
**declaration**
mechanism for establishing the definition of an *entity*

3.28建立实体定义的声明机制

Note 1 to entry: A declaration can involve attaching an *identifier* to the entity, and allocating *attributes* such as *data types* and *algorithms* to it.

注1：声明可以涉及将标识符附加到实体，并为其分配数据类型和算法等属性。

**3.29**
**device**
independent physical *entity* capable of performing one or more specified *functions* in a particular context and delimited by its *interfaces*

3.29独立于设备的物理实体，能够在特定上下文中执行一项或多项指定功能并由其接口界定

Note 1 to entry: A *programmable controller system* as defined in IEC 61131-1 is a *device*.

注1：IEC61131-1中定义的可编程控制器系统是一个设备。

**3.30**
**device management application**
*application* whose primary function is the management of multiple *resources* within a *device*

3.30设备管理应用程序主要功能是管理一个设备内的多个资源的应用程序

**3.31**
**entity**
particular thing, such as a person, place, *process*, object, concept, association, or *event*

3.31entity特定事物，例如人、地点、过程、对象、概念、关联或事件

**3.32**
**event**
instantaneous occurrence that is significant to scheduling the *execution* of an *algorithm*

Note 1 to entry: The execution of an algorithm may make use of *variables* associated with an event.

**3.33**
**event connection**
association among *function blocks* for the conveyance of *events*

**3.34**
**event input**
*interface* of a *function block* which can receive *events* from an *event connection*

**3.35**
**event output**
*interface* of a *function block* which can issue *events* to an *event connection*

**3.36**
**exception**
*event* that causes suspension of normal *execution*

**3.37**
**execution**
process of carrying out a sequence of *operations* specified by an *algorithm*

Note 1 to entry: The sequence of operations to be executed may vary from one *invocation* of a *function block instance* to another, depending on the rules specified by the function block's *algorithm* and the current values of *variables* in the function block's data structure.

**3.38**
**execution control action**
**EC action**
element associated with an *execution control state,* which identifies an *algorithm* to be *executed,* an *event* to be issued, or both

Note 1 to entry: Timing of algorithm execution and event issuance are addressed in 5.2.2.

**3.39**
**execution control chart**
**ECC**
graphical or textual representation of the causal relationships among *events* at the *event inputs* and *event outputs* of a *function block* and the *execution* of the function block's *algorithms*, using *execution control states, execution control transitions,* and *execution control actions*

**3.40**
**execution control initial state**
**EC initial state**
*execution control state* that is active upon initialization of an *execution control chart*

**3.41**
**execution control state**
**EC state**
situation in which the behavior of a *basic function block* with respect to its *variables* is determined by the *algorithms* associated with a specified set of *execution control actions*





3.32事件瞬时发生，对调度算法的执行很重要

注1：算法的执行可能会使用与事件相关的变量。

3.33事件连接功能块之间用于传送事件的关联

3.34事件输入接口可以从事件连接接收事件的功能块

3.35事件输出接口可以向事件连接发出事件的功能块

3.36导致正常执行暂停的异常事件

3.37executionprocess执行算法指定的一系列操作的过程

注1：根据功能块算法指定的规则和功能块数据结构中变量的当前值，要执行的操作顺序可能因功能块实例的一次调用而异。

3.38执行控制动作与执行控制状态相关联的EC动作元素，它标识要执行的算法、要发出的事件，或两者兼有。

注1：算法执行和事件发布的时间在5.2.2中讨论。

3.39执行控制图ECC使用执行控制状态、执行控制转换和执行控制动作来表示功能块的事件输入和事件输出处的　　事件与功能块算法的执行之间的因果关系的图形或文本。

3.40执行控制初始状态EC初始状态执行控制状态，在执行控制图初始化时处于活动状态

3.41执行控制状态ECstate基本功能块相对于其变量的行为由与一组指定的执行控制动作相关的算法确定的状态。

**3.42**
**execution control transition**
**EC transition**
means by which control passes from a predecessor *execution control state* to a successor *execution control state*

3.42执行控制转换EC转换是指控制从前一个执行控制状态转移到一个后继执行控制状态。

**3.43**
**fault**
abnormal condition that may cause a reduction in, or loss of, the capability of a *functional unit* to perform a required *function*

3.43fault可能导致功能单元执行所需功能的能力降低或丧失的异常情况

**3.44**
**function**
specific purpose of an *entity* or its characteristic action

3.44function实体的特定目的或其特征动作

**3.45**
**function block**
**function block instance**
*software functional unit* comprising an individual, named copy of a data structure upon which associated *operations* may be performed as specified by a corresponding *function block type*

3.45功能块functionblockinstance软件功能单元，包括一个单独的、命名的数据结构副本，在该副本上可以执行相应的功能块类型指定的相关操作

Note 1 to entry: Typical operations of a function block include modification of the values of the data in its associated data structure.

注1：功能块的典型操作包括修改其相关数据结构中的数据值。

Note 2 to entry: The *function block instance* and its corresponding *function block type* defined in IEC 61131-3 are programming language elements with a different set of features.

注2：IEC61131-3中定义的功能块实例及其相应的功能块类型是具有不同特征集的编程语言元素。

**3.46**
**function block network**
*network* whose nodes are *function blocks* or *subapplications* and their *parameters* and whose branches are *data connections* and *event connections*

3.46functionblocknetwork以功能块或子应用及其参数为节点，以数据连接和事件连接为分支的网络

Note 1 to entry: This is a generalization of the *function block diagram* defined in IEC 61131-3.

注1：这是IEC61131-3中定义的功能框图的概括。

**3.47**
**function block type**
*type* whose *instances* are *function blocks*

3.47功能块类型实例为功能块的类型

Note 1 to entry: Function block types include basic function block types, composite function block types, and service interface function block types

注1：功能块类型包括基本功能块类型、复合功能块类型和服务接口功能块类型

**3.48**
**functional unit**
*entity* of *hardware* or *software,* or both, capable of accomplishing a specified purpose

3.48功能单元能够完成特定目的的硬件或软件实体，或两者兼有。

**3.49**
**hardware**
physical equipment, as opposed to programs, procedures, rules and associated documentation

**3.49硬件物理**
equipment, as opposed to programs, procedures, rules and associated

**3.50**
**identifier**
one or more *characters* used to name an *entity*

3.50identifier用来命名一个实体的一个或多个字符

**3.51**
**implementation**
development phase in which the *hardware* and *software* of a *system* become operational

3.51implementationdevelopment系统的硬件和软件开始运行的阶段

**3.52**
**indication primitive**
*service primitive* which represents an interaction in which a *resource* either
a) indicates that it has, on its own initiative, *invoked* some *algorithm*; or
b) indicates that an *algorithm* has been invoked by a peer *application*

**3.53**
**information**
meaning that is currently assigned to *data* by means of the conventions applied to that data

**3.54**
**input variable**
*variable* whose value is supplied by a *data input*, and which may be used in one or more *operations* of a *function block*

Note 1 to entry: An *input parameter* of a *function block*, as defined in IEC 61131-3, is an *input variable*.

**3.55**
**instance**
*functional unit* comprising an individual, named *entity* with the *attributes* of a defined *type*

**3.56**
**instance name**
*identifier* associated with and designating an *instance*

**3.57**
**instantiation**
creation of an *instance* of a specified *type*

**3.58**
**interface**
shared boundary between two *functional units,* defined by functional characteristics, signal characteristics, or other characteristics, as appropriate

**3.59**
**internal operation**
<of a *function block*> *operation* associated with an *algorithm* of a function block, with its *execution* control, or with the functional capabilities of the associated *resource*

**3.60**
**internal variable**
*variable* whose value is used or modified by one or more *operations* of a *function block*, but is not supplied by a *data input* or to a *data output*

**3.61**
**invocation**
process of initiating the *execution* of the sequence of *operations* specified in an *algorithm*

**3.62**
**link**
design element describing the *connection* between a *device* and a *network segment*

**3.63**
**literal**
lexical unit that directly represents a value





3.52指示原语表示一种交互的服务原语，其中资源a)表明它主动调用了某种算法；或b)表示一个算法已被对等应用程序调用

3.53信息指当前通过应用于数据的约定赋予数据的含义

3.54输入变量其值由数据输入提供的变量，可用于功能块的一项或多项操作

注1：功能块的输入参数，如IEC61131-3中定义的，是输入变量。

3.55实例功能单元，包括具有定义类型属性的单个命名实体

3.56实例名称与实例相关联并指定实例的标识符

3.57实例化创建指定类型的实例

3.58interface两个功能单元之间的共享边界，由功能特性、信号特性或其他特性定义，视情况而定。

3.59internaloperation<功能块的>与功能块的算法、执行控制或相关资源的功能能力相关的操作

3.60internalvariable其值被功能块的一个或多个操作使用或修改，但不由数据输入或数据输出提供的变量。

3.61invocationprocess启动执行算法中指定的操作序列的过程

3.62链路设计元素，描述设备与网段之间的连接

3.63直接表示值的文字词法单元

**3.64**
**management function block**
*function block* whose primary *function* is the management of *applications* within a *resource*

3.64管理功能块功能块，其主要功能是管理资源内的应用程序

**3.65**
**management resource**
*resource* whose primary *function* is the management of other *resources*

3.65managementresource主要功能是管理其他资源的资源

**3.66**
**mapping**
set of features or *attributes* having defined correspondence with the members of another set

3.66映射一组特征或属性与另一组的成员具有定义的对应关系

**3.67**
**message**
ordered series of *characters* intended to convey *information*

3.67message用于传达信息的有序字符系列

**3.68**
**message sink**
part of a communication *system* in which *messages* are considered to be received

3.68messagesink消息接收器通信系统的一部分，消息被认为在其中被接收。

**3.69**
**message source**
part of a communication *system* from which *messages* are considered to originate

3.69messagesource消息源通信系统的一部分，消息被认为是源自该部分。

**3.70**
**model**
mathematical or physical representation of a system or a process

3.70模型系统或过程的数学或物理表示

**3.71**
**multitasking**
mode of operation that provides for the *concurrent execution* of two or more *algorithms*

3.71多任务操作模式，提供两个或多个算法的并发执行

**3.72**
**network**
arrangement of nodes and interconnecting branches

3.72节点和互连分支的网络排列

**3.73**
**operation**
well-defined action that, when applied to any permissible combination of known *entities*, produces a new *entity*

3.73operation定义明确的操作，当应用于已知实体的任何允许组合时，产生一个新实体

**3.74**
**output variable**
*variable* whose value is established by one or more *operations* of a *function block,* and is supplied to a *data output*

3.74输出变量其值由功能块的一个或多个操作建立，并提供给数据输出的变量

Note 1 to entry: An *output parameter* of a *function block*, as defined in IEC 61131-3, is an *output variable.*

注1：功能块的输出参数，如IEC61131-3中所定义，是一个输出变量。

**3.75**
**parameter**
*variable* that is given a constant value for a specified *application* and that may denote the application

3.75参数变量，为指定的应用程序被赋予一个常数值，并且可以表示应用程序

**3.76**
**plug**
**plug adapter**
*instance* of an *adapter interface type* which provides a starting point for an *adapter connection* from a *provider* function block

3.76plugplugadapter适配器接口类型的实例，它为来自提供程序功能块的适配器连接提供起点

**3.77**
**provider**
*function block instance* which provides a *plug adapter* of a defined *adapter interface type*

**3.78**
**request primitive**
*service primitive* which represents an interaction in which an *application* invokes some *algorithm* provided by a *service*

**3.79**
**requester**
*functional unit* which initiates a *transaction* via a *request primitive*

**3.80**
**resource**
*functional unit* which has independent control of its operation, and which provides various *services* to *applications,* including the scheduling and *execution* of *algorithms*

Note 1 to entry: The RESOURCE defined in IEC 61131-3:2003, 1.3.66 is a programming language element corresponding to the *resource* defined above.

Note 2 to entry: A *device* contains one or more resources.

**3.81**
**resource management application**
*application* whose primary function is the management of a single *resource*

**3.82**
**responder**
*functional unit* which concludes a *transaction* via a *response primitive*

**3.83**
**response primitive**
*service primitive* which represents an interaction in which an *application* indicates that it has completed some *algorithm* previously *invoked* by an interaction represented by an *indication primitive*

**3.84**
**sample**, verb
to sense and retain the instantaneous value of a *variable* for later use

**3.85**
**scheduling function**
*function* which selects *algorithms* or *operations* for *execution,* and initiates and terminates such execution

**3.86**
**segment**
physical partition of a *communication network*

**3.87**
**service**
functional capability of a *resource* which can be modeled by a sequence of *service primitives*

**3.88**
**service interface function block**
*function block* which provides one or more *services* to an *application*, based on a *mapping* of *service primitives* to the function block's *event inputs, event outputs, data inputs* and *data outputs*





3.77提供者功能块实例，提供定义的适配器接口类型的插头适配器

3.78请求原语服务原语，表示应用程序调用服务提供的某种算法的交互。

3.79请求者功能单元，通过请求原语发起事务

3.80资源功能单元，独立控制其运行，为应用程序提供各种服务，包括算法的调度和执行

注1：IEC61131-3:2003 1.3.66中定义的RESOURCE是与上面定义的资源相对应的编程语言元素。

注2：一个设备包含一个或多个资源。

3.81资源管理应用程序主要功能是管理单一资源的应用程序

3.82响应者功能单元，通过响应原语结束交易

3.83响应原语表示交互的服务原语，在该交互中应用程序表明它已经完成了先前由指示原语表示的交互调用的某个算法

3.84sample verb感知和保留变量的瞬时值以备后用

3.85调度功能选择执行的算法或操作，并启动和终止这种执行的功能

3.86段通信网络的物理分区

3.87服务功能能力可以通过一系列服务原语建模的资源

3.88服务接口functionblockfunctionblock，根据服务原语到功能块的事件输入、事件输出、数据输入和数据输出的映射，向应用程序提供一项或多项服务。

**3.89**
**service primitive**
abstract, implementation-independent representation of an interaction between an *application* and a *resource*

3.89服务原语应用程序和资源之间交互的抽象、独立于实现的表示

**3.90**
**service sequence diagram**
diagram representing a sequence of *service primitives*

3.90服务序列图表示服务原语序列的图

**3.91**
**socket**
**socket adapter**
*instance* of an *adapter interface type* which provides an end point for an *adapter connection* to an *acceptor* function block

3.91socketsocket适配器适配器接口类型的实例，它为适配器连接到接受器功能块提供端点

**3.92**
**software**
intellectual creation comprising the programs, procedures, rules, *configurations* and any associated documentation pertaining to the operation of a *system*

3.92软件智能创造，包括与系统操作有关的程序、程序、规则、配置和任何相关文档

**3.93**
**software tool**
*software* that is used for the production, inspection or analysis of other software

3.93软件工具用于制作、检查或分析其他软件的软件

**3.94**
**subapplication instance**
*instance* of a *subapplication type* inside an *application* or inside a subapplication type

Note 1 to entry: A subapplication instance may be distributed among *resources*, i.e. its component function blocks or the content of its component subapplications may be assigned to different resources.

3.94subapplicationinstance应用程序内部或子应用程序类型内部的子应用程序类型的实例

注1：一个子应用实例可以分布在资源之间，即它的组件功能块或其组件子应用的内容可以分配给不同的资源。

**3.95**
**subapplication type**
*functional unit* whose body consists of interconnected *component function blocks* or *component subapplications*

Note 1 to entry: A subapplication type enables the creation of substructures of *applications* in the form of a self-similar hierarchy.

3.95subapplicationtype功能单元，其主体由互连的组件功能块或组件子应用程序组成。

注1：子应用程序类型能够以自相似层次结构的形式创建应用程序的子结构。

**3.96**
**system**
set of interrelated elements considered in a defined context as a whole and separated from its environment

Note 1 to entry: Such elements may be both material objects and concepts as well as the results thereof (e.g. forms of organisation, mathematical methods, and programming languages).

Note 2 to entry: The system is considered to be separated from the environment and other external systems by an imaginary surface, which can cut the links between them and the considered system.

3.96系统一组相互关联的元素，在定义的上下文中被视为一个整体，并与其环境分离

注1：这些元素既可以是物质对象，也可以是概念及其结果（例如，组织形式、数学方法和编程语言）。

注2：系统被认为是通过一个假想的表面与环境和其他外部系统分开，这可以切断它们与所考虑系统之间的联系。

**3.97**
**temporary variable**
*variable* whose value is initialized, used and possibly modified during *execution* of an *algorithm*; that is not visible outside the body of the algorithm, and whose value does not persist from one execution of the algorithm to the next

3.97临时变量其值在算法执行期间被初始化、使用和可能修改的变量；在算法主体之外不可见，并且其值不会从算法的一次执行到下一次执行

**3.98**
**transaction**
unit of service in which a request and possibly *data* is conveyed from a *requester* to a *responder*, and in which a response and possibly data may also be conveyed from the responder back to the requester

3.98transactionunitofservice其中一个请求和可能的数据从请求者传送到响应者，其中响应和可能的数据也可以从响应者传送回请求者

**3.99**
**type**
*software* element which specifies the common *attributes* shared by all *instances* of the type

**3.100**
**type name**
*identifier* associated with and designating a *type*

**3.101**
**unidirectional transaction**
*transaction* in which a request and possibly *data* is/are conveyed from an *requester* to a *responder*, and in which a response is not conveyed from the responder back to the requester

**3.102**
**variable**
*software entity* that may take different values, one at a time

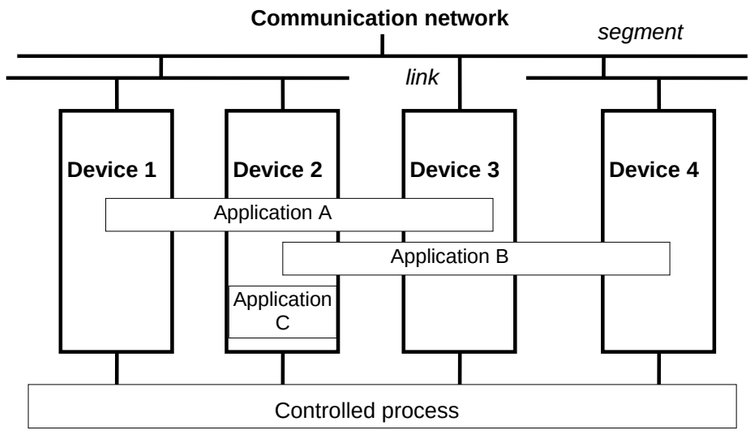
Note 1 to entry: The values of a variable are usually restricted to a certain *data type*.

Note 2 to entry: Variables may be classified as *input variables, output variables*, *internal variables* and *temporary variables*.

## 4 Reference models

### 4.1 System model

For the purposes of IEC 61499, an industrial process measurement and control *system* (IPMCS) is modeled, as shown in Figure 1, as a collection of *devices* interconnected and communicating with each other by means of a communication *network* consisting of *segments* and *links*. Devices are connected to network segments via *links*.




NOTE The controlled process is not part of the measurement and control system.

**Figure 1 – System model**

A *function* performed by the IPMCS is modeled as an *application* which may reside in a single device, such as application C in Figure 1, or may be distributed among several devices, such as applications A and B in Figure 1. For instance, an application may consist of one or more control loops in which the input sampling is performed in one device, control processing is performed in another, and output conversion in a third.





3.99type软件元素，它指定由该类型的所有实例共享的公共属性

3.100类型名称与类型关联并指定类型的标识符

3.101单向事务一个请求和可能的数据从请求者传送到响应者的事务，其中响应不从响应者传回请求者。

3.102可变的软件实体，可以采用不同的值，一次一个

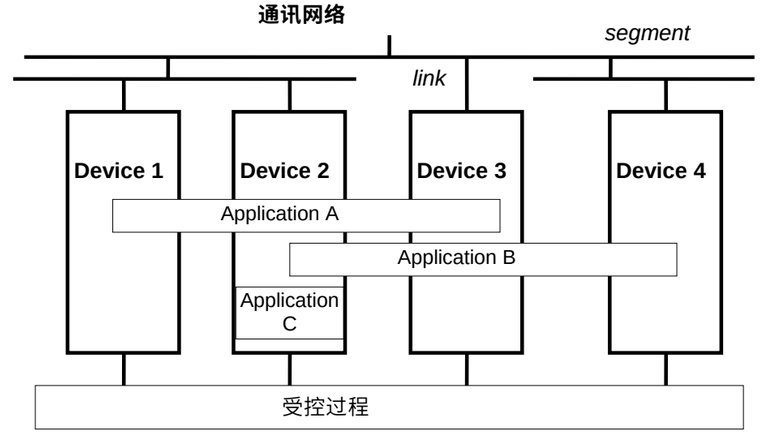
注1：变量的值通常被限制为某种数据类型。

注2：变量可分为输入变量、输出变量、内部变量和临时变量。

## 4 参考模型

### 系统型号

出于IEC61499的目的，工业过程测量和控制系统(IPMCS)被建模，如图1所示，作为通过由段和链路组成的通信网络相互连接和通信的设备集合。设备通过链路连接到网段。




注：受控过程不是测量和控制系统的一部分。

**图1 系统模型**

IPMCS执行的功能被建模为一个应用程序，该应用程序可以驻留在单个设备中，例如图1中的应用程序C，也可以分布在多个设备中，例如图1中的应用程序A和B。例如，一个应用程序可以由一个或多个控制回路组成，其中输入采样在一个设备中执行，控制处理在另一个设备中执行，输出转换在第三个设备中执行。

## 4.2 Device model

As illustrated in Figure 2, a *device* shall contain at least one *interface,* that is, process interface or communication interface, and can contain zero or more *resources*.

NOTE 1 A device is considered to be an *instance* of a corresponding device *type*, defined as specified in 7.2.2.

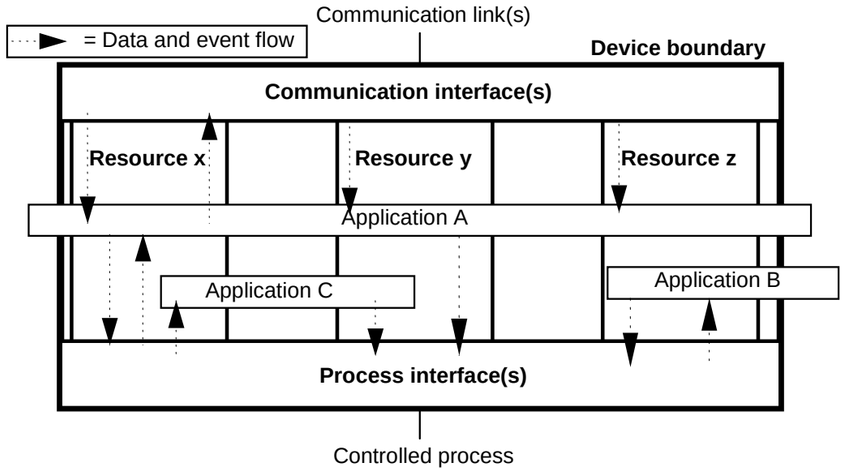
NOTE 2 A device that contains no resources is considered to be functionally equivalent to a *resource* as defined in 4.3.

A "process interface" provides a *mapping* between the physical process (analog measurements, discrete I/O, etc.) and the resources. Information exchanged with the physical process is presented to the resource as *data* or *events*, or both.

Communication *interfaces* provide a mapping between resources and the information exchanged via a communication *network*. Services provided by communication interfaces may include:

- presentation of communicated information to the resource as *data* or *events*, or both;
- additional services to support programming, *configuration*, diagnostics, etc.

Communication *links* may either be associated directly with a *device*, or with an instance of a specific *resource* type (communication resource), onto which part of the distributed application may or may not be mapped, depending on the resource type.




NOTE This figure shows a possible internal structure of Device 2 from Figure 1.

**Figure 2 – Device model**

## 4.3 Resource model

For the purposes of IEC 61499, a *resource* is considered to be a *functional unit*, which has independent control of its operation, contained in a *device*. It may be created, configured, parameterized, started up, deleted, etc., without affecting other resources.

NOTE 1 A resource is considered to be an *instance* of a corresponding resource *type*, defined as specified in 7.2.1.

NOTE 2 Although a resource has independent control of its operation, its operational states might need to be coordinated with those of other resources for the purposes of installation, test, etc.





**设备型号**

如图2所示，一个设备应至少包含一个接口，即进程接口或通信接口，并且可以包含零个或多个资源。

注1设备被认为是相应设备类型的实例，定义见7.2.2。

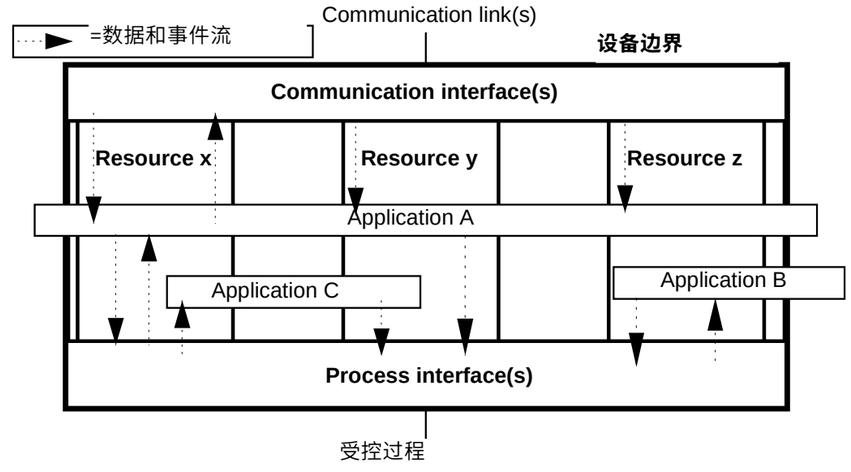
注2：不包含资源的设备被认为在功能上等同于4.3中定义的资源。

“过程接口”提供物理过程（模拟测量、离散IO等）和资源之间的映射。与物理过程交换的信息以数据或事件或两者的形式呈现给资源。

通信接口提供资源和通过通信网络交换的信息之间的映射。通信接口提供的服务可能包括：

- 将已传达的信息作为数据或事件呈现给资源，或两者兼而有之；
- 支持编程、配置、诊断等的附加服务。

通信链路可以直接与设备相关联，也可以与特定资源类型（通信资源）的实例相关联，根据资源类型，分布式应用程序的一部分可能映射或可能不映射到该实例。




注意此图显示了图1中设备2的可能内部结构。

**图2 设备型号**

**资源模型**

就IEC61499而言，资源被认为是一个功能单元，它可以独立控制其操作，包含在设备中。它可以被创建、配置、参数化、启动、删除等，而不影响其他资源。

注1资源被认为是相应资源类型的实例，定义见7.2.1。

注2：尽管资源对其操作具有独立控制权，但出于安装、测试等目的，其操作状态可能需要与其他资源的操作状态相协调。

The *functions* of a resource are to accept *data* and/or *events* from the process and/or communication *interfaces*, process the data and/or events, and to return data and/or events to the process and/or communication interfaces, as specified by the *applications* utilizing the resource.

NOTE 3 Besides supporting the functions enumerated above, specific types of resources might represent the capability to implement interface functions such as process interfaces or lower layer communication services over communication links. Depending on the type of those resources, these services might or might not be the only ones they are able to provide.

NOTE 4 The consideration of other possible aspects of resources is beyond the scope of this standard.

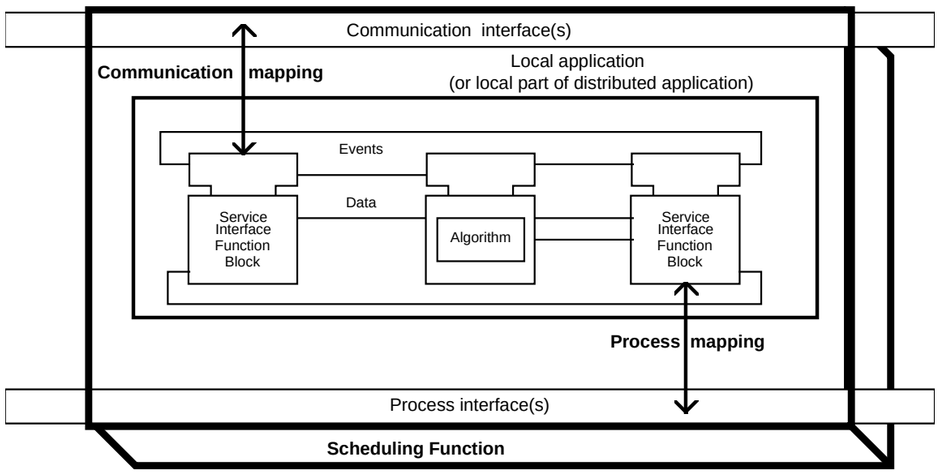
As illustrated in Figure 3, a resource is modeled by the following.

- One or more "local applications" (or local parts of distributed applications). The *variables* and *events* handled in this part are *input* and *output variables* and events at *event inputs* and *event outputs* of *function blocks* that perform the *operations* needed by the application.
- A "process mapping" part whose function is to perform a *mapping* of *data* and *events* between *applications* and process *interface(s).* As shown in Figure 3, this mapping may be modeled by *service interface function blocks* specialized for this purpose.
- A "communication mapping" part whose function is to perform a *mapping* of *data* and *events* between *applications* and *communication interfaces.* As shown in Figure 3, this mapping may be modeled by *service interface function blocks* specialized for this purpose.
- A scheduling *function* which effects the execution of, and data transfer between, the function blocks in the applications, according to the timing and sequence requirements determined by:

a) the occurrence of events;

b) function block interconnections; and

c) scheduling information such as periods and priorities.




NOTE 1 This figure is illustrative only. Neither the graphical representation nor the location of function blocks is normative.

NOTE 2 Communication and process interfaces can be shared among resources.

**Figure 3 – Resource model**



资源的功能是从进程和/或通信接口接受数据和/或事件，处理数据和/或事件，并将数据和/或事件返回到进程和/或通信接口，由使用该资源的应用程序指定.

注3除了支持上面列举的功能外，特定类型的资源可能代表实现接口功能的能力，例如通过通信链路实现进程接口或低层通信服务。根据这些资源的类型，这些服务可能是也可能不是它们能够提供的唯一服务。

注4对资源其他可能方面的考虑超出了本标准的范围。

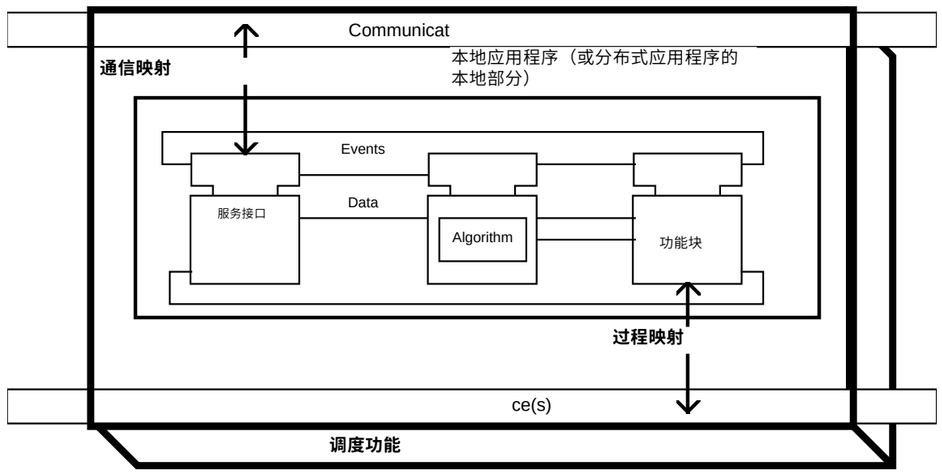
如图3所示，资源通过以下方式建模。

- 一个或多个“本地应用程序”（或分布式应用程序的本地部分）。这部分处理的变量和事件是执行应用程序所需操作的功能块的事件输入和事件输出处的 输入和输出变量和事件。
- “进程映射”部分，其功能是在应用程序和进程接口之间执行数据和事件的映射。如图3所示，这种映射可以由专门为此目的的服务接口功能块建模。
- “通信映射”部分，其功能是在应用程序和通信接口之间执行数据和事件的映射。如图3所示，这种映射可以由专门为此目的的服务接口功能块建模。
- 一种调度功能，根据由以下因素确定的时序和顺序要求，影响应用程序中功能块的执行和数据传输：

a)事件的发生；

b)功能块互连；和

c)调度信息，例如周期和优先级。




注1该图仅用于说明。图形表示和功能块的位置都不是规范的。

注2：通信和过程接口可以在资源之间共享。

**图3 资源模型**

### 4.4 Application model

For the purposes of this document, an *application* consists of a *function block network,* whose nodes are *function blocks* or *subapplications* and their *parameters* and whose branches are *data connections* and *event connections*.

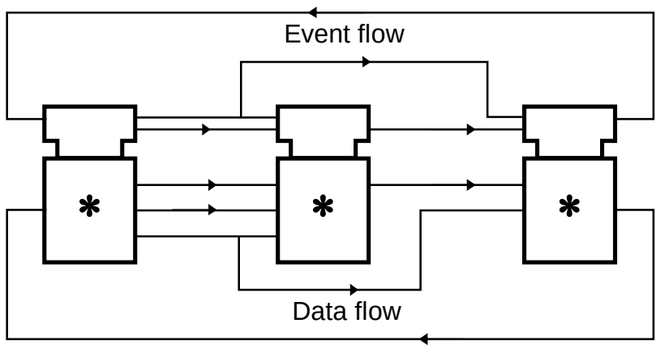
Subapplications are *instances* of *subapplication types*, which like applications consist of *function block networks*. Application names, subapplication and function block *instance names* may therefore be used to create a hierarchy of *identifiers* that can uniquely identify every *function block instance* in a *system*.

An application can be distributed among several *resources* in the same or different *devices*. A *resource* uses the causal relationships specified by the application to determine the appropriate responses to *events* which may arise from communication and process interfaces or from other functions of the resource. These responses may include:

- scheduling and *execution* of *algorithms;*
- modification of *variables;*
- generation of additional events;
- interactions with communication and process interfaces.

In the context of this document, applications are defined by *function block networks* specifying event and data flow among *function block* or *subapplication instances,* as illustrated in Figure 4. The event flow determines the scheduling and *execution* by the associated resource of the *operations* specified by each function block's *algorithm(s),* according to the rules given in 5.2.2.

Standards, components and systems complying with this standard may utilize alternative means for scheduling of execution. Such alternative means shall be exactly specified using the elements defined in this standard.




NOTE 1 "*" represents function block or subapplication instances.

NOTE 2 This figure is illustrative only. The graphical representation is not normative.

**Figure 4 – Application model**

### 4.5 Function block model

#### 4.5.1 Characteristics of function block instances

A *function block (function block instance)* is a *functional unit* of *software* comprising an individual, named copy of the data structure specified by a *function block type*, which persists from one *invocation* of the function block to the next. The characteristics of function block instances are described in 4.5.1, and function block type specifications are described in 4.5.2.





**应用模型**

就本文档而言，应用程序由功能块网络组成，其节点是功能块或子应用程序及其参数，其分支是数据连接和事件连接。

子应用程序是子应用程序类型的实例，它与应用程序一样由功能块网络组成。因此，应用程序名称、子应用程序和功能块实例名称可用于创建标识符层次结构，该标识符可以唯一地标识系统中的每个功能块实例。

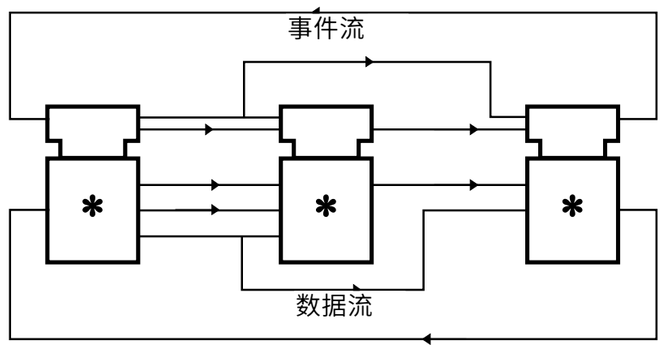
一个应用程序可以分布在相同或不同设备中的多个资源中。资源使用应用程序指定的因果关系来确定对可能来自通信和处理接口或资源的其他功能的事件的适当响应。这些回应可能包括：

- 算法的调度和执行；
- 修改变量；
- 产生额外的事件；

  与通信和过程接口的交互。

在本文档的上下文中，应用程序由功能块网络定义，指定功能块或子应用程序实例之间的事件和数据流，如图4所示。事件流通过相关资源确定每个指定的操作的调度和执行功能块的算法，根据5.2.2中给出的规则。

符合本标准的标准、组件和系统可以使用替代方法来安排执行。此类替代方法应使用本标准中定义的元素准确指定。




注1“*”代表功能块或子应用实例。

注2该图仅用于说明。图形表示不规范。

**图4 应用模型**

### 4.5 功能块型号

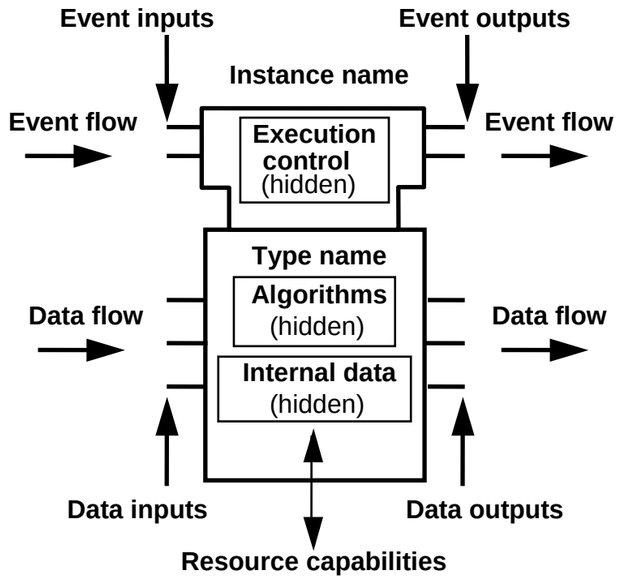
**功能块实例的特征**

功能块（功能块实例）是软件的功能单元，包括由功能块类型指定的数据结构的单个命名副本，该副本从功能块的一次调用持续到下一次调用。功能块实例的特性在4.5.1中描述，功能块类型规范在4.5.2中描述。

A *function block instance* exhibits the following characteristic features as illustrated in Figure 5:

- its *type name* and *instance name*;
- a set of *event inputs,* each of which can receive *events* from an *event connection* which may affect the execution of one or more *algorithms*;
- a set of *event outputs,* each of which can issue *events* to an *event connection* depending on the execution of *algorithms* or on some other functional capability of the *resource* in which the function block is located;
- a set of *data inputs,* which may be *mapped* to corresponding *input variables*;
- a set of *data outputs,* which may be *mapped* to corresponding *output variables*;
- internal *data,* which may be *mapped* to a set of *internal variables*;
- functional characteristics which are determined by combining internal data or state information, or both, with a set of *algorithms*, functional capabilities of the associated *resource,* or both. These functional characteristics are defined in the function block's *type* specification.

NOTE Internal state information can be represented by *internal variables* or by an internal representation of an execution control state machine.




NOTE This figure is illustrative only. The graphical representation is not normative.

**Figure 5 – Characteristics of function blocks**

The algorithms contained within a function block are in principle invisible from the outside of the function block, except as described formally or informally by the provider of the function block. Additionally, the function block may contain internal *variables* or state information, or both, which persist between invocations of the function block's algorithms, but which are not accessible by data flow connections from the outside of the function block.

Access to internal variables and state information of function block instances may be provided by additional functional capabilities of the associated resource.

Means for specifying the causal relationships among event inputs, event outputs, and execution of algorithms are defined in Clauses 5 and 6.

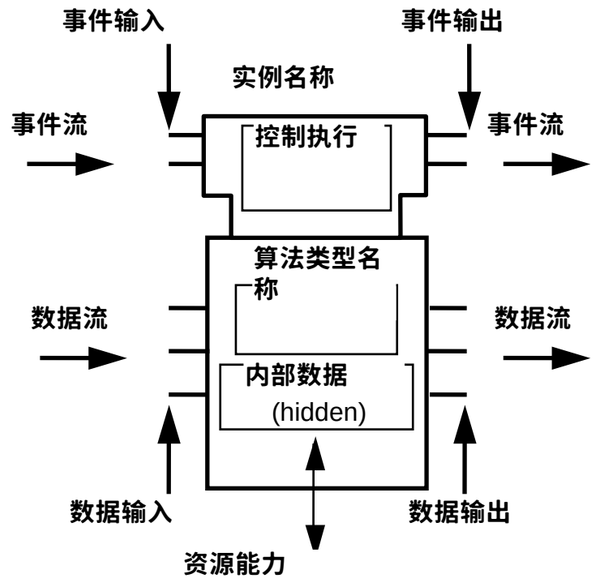


功能块实例具有以下特征，如图所示

Fig

- 它的类型名称和实例名称；
- 一组事件输入，每个输入都可以从事件连接接收事件，这些事件可能会影响一个或多个算法的执行；
- 一组事件输出，每个事件输出都可以根据算法的执行或功能块所在资源的某些其他功能能力向事件连接发出事件；
- 一组数据输入，可以映射到相应的输入变量；
- 一组数据输出，可以映射到相应的输出变量；
- 内部数据，可以映射到一组内部变量；
- 通过将内部数据或状态信息或两者与一组算法、相关资源的功能能力或两者结合来确定的功能特性。这些功能特性在功能块的类型规范中定义。

注：内部状态信息可以由内部变量或执行控制状态机的内部表示来表示。




注意此图仅用于说明。图形表示不规范。

**图5 功能块的特性**

功能块中包含的算法原则上从功能块的外部是不可见的，除非功能块的提供者正式或非正式地描述。此外，功能块可能包含内部变量或状态信息或两者，它们在功能块算法的调用之间持续存在，但不能通过来自功能块外部的数据流连接访问。

可以通过相关资源的附加功能能力提供对功能块实例的内部变量和状态信息的访问。

用于指定事件输入、事件输出和算法执行之间的因果关系的方法在第5章和第6章中定义。

### 4.5.2 Function block type specifications

A *function block type* is a *software* element which specifies the characteristics of all *instances* of the type, including:

- its *type name*;
- the number, names, type names and order of *event inputs* and *event outputs.*
- the number, names, *data type* and order of input, output and internal *variables*;

Mechanisms for the *declaration* of these characteristics are defined in 5.2.1.

In addition, the function block type specification defines the functionality of *instances* of the type. This functionality may be expressed as follows:

- For *basic function block types,* declaration mechanisms are provided in 5.2.1.3 for the specification of *algorithms*, which operate on the values of *input variables, output variables,* and *internal variables* to produce new values of *output variables* and *internal variables.* The associations among the *invocation* of algorithms and the occurrence of *events* at event inputs and outputs are expressed in terms of an *execution control chart* (ECC), using the declaration mechanisms defined in 5.2.1.4.
- The functionality of an *instance* of a *composite function block type* or a *subapplication type* is declared, using the mechanisms defined in 5.3.1 and 5.4.1 respectively, in terms of *data connections* and *event connections* among its *component function blocks* or subapplications and the event and data inputs and outputs of the composite function block or the subapplication.
- The functionality of an instance of a *service interface function block type* is described by a *mapping* of *service primitives* to *event inputs, event outputs, data inputs* and *data outputs*, using the declaration mechanisms defined in 6.1.
- Other means such as natural language text may be used for describing the functionality of a function block type; however, the specification of such means is beyond the scope of this standard.

### 4.5.3 Execution model for basic function blocks

As shown in Figure 6, the *execution* of *algorithms* for *basic function blocks* is invoked by the **execution control** portion of a *function block instance* in response to events at event inputs. This *invocation* takes the form of a request to the **scheduling function** of the associated *resource* to schedule the execution of the algorithm's *operations*. Upon completion of execution of an algorithm, the execution control generates zero or more events at *event outputs* as appropriate.

*Events at event inputs* are provided by connection to *event outputs* of other function block instances or the same function block instance. Events at these event outputs may be generated by execution control as described above, or by the "communication mapping", "process mapping", "scheduling", or other functional capability of the *resource.*

NOTE 1 Execution control in composite function blocks is achieved via event flow within the function block body.

Figure 6 depicts the order of events and algorithm execution for the case in which a single event input, a single algorithm, and a single event output are associated. The relevant times in this diagram are defined as follows:

$t_1$: relevant input variable values (i.e., those associated with the event input by the WITH qualifier defined in 5.2.1.2) are made available;

$t_2$: the event at the event input occurs;

$t_3$: the execution control function notifies the resource scheduling function to schedule an algorithm for execution;





**功能块类型规格**

功能块类型是一个软件元素，它指定该类型的所有实例的特征，包括：

- 它的类型名称；
- 事件输入和事件输出的数量、名称、类型名称和顺序。
  输入、输出和内部变量的数量、名称、数据类型和顺序；

声明这些特性的机制在5.2.1中定义。

此外，功能块类型规范定义了该类型实例的功能。这个功能可以表达如下：

- 对于基本功能块类型，5.2.1.3中提供了声明机制，用于规范算法，对输入变量、输出变量和内部变量的值进行操作，以产生输出变量和内部变量的新值。使用5.2.1.4中定义的声明机制，算法调用与事件输入和输出处事件发生之间的关联以执行控制图(ECC)的形式表示。

- 复合功能块类型或子应用类型的实例的功能性分别使用5.3.1和5.4.1中定义的机制在数据连接方面进行声明
  块或子应用程序以及复合功能块或子应用程序的事件和数据输入和输出。

- 服务接口功能块类型实例的功能通过使用6.1中定义的声明机制将服务原语映射到事件输入、事件输出、数据输入和数据输出来描述。

- 其他方式如自然语言文本可用于描述功能块类型的功能；但是，这种方式的规范超出了本标准的范围。

**基本功能块的执行模型**

如图6所示，基本功能块的算法执行由功能块实例的执行控制部分调用，以响应事件输入s处的事件。此调用采用对关联资源的调度功能的请求的形式，以调度算法操作的执行。在完成算法的执行时，执行控制在适当的事件输出处生成零个或多个事件。

事件输入处的事件通过连接到其他功能块实例或同一功能块实例的事件输出来提供。这些事件输出处的事件可以由如上所述的执行控制或由“通信映射”、“进程映射”、“调度”或资源的其他功能能力生成。

注1复合功能块中的执行控制是通过功能块体内的事件流实现的。

图6描述了单个事件输入、单个算法和单个事件输出相关联的情况下的事件顺序和算法执行。该图中的相关时间定义如下：

$t_1$: 相关的输入变量值（即与由5.2.1.2中定义的WITH限定符输入的事件相关联的值）可用；

$t_2$: 事件输入处的事件发生；

$t_3$: 执行控制功能通知资源调度功能调度算法执行；

$t_4$: algorithm execution begins;

$t_5$: the algorithm completes the establishment of values for the output variables associated with the event output by the WITH qualifier defined in 5.2.1.2;

$t_6$: the resource scheduling function is notified that algorithm execution has ended;

$t_7$: the scheduling function invokes the execution control function;

$t_8$: the execution control function signals an event at the event output.

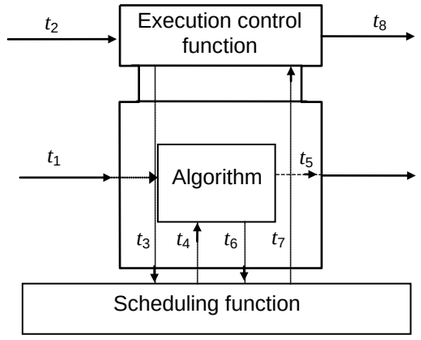
As shown in Figure 7, the significant timing delays in this case which are of interest in application design are:

$T_{setup} = t_2 - t_1$

$T_{start} = t_4 - t_2$ (time from event at event input to beginning of algorithm execution)

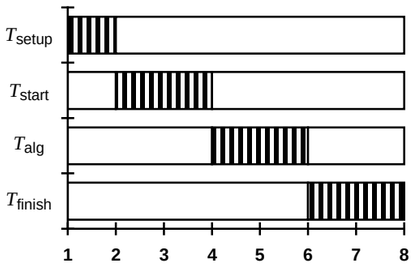
$T_{alg} = t_6 - t_4$ (algorithm execution time)

$T_{finish} = t_8 - t_6$ (time from end of algorithm execution to event at event output)




NOTE This figure is illustrative only. The graphical representation is not normative.

**Figure 6 – Execution model**




NOTE The axis labels 1,2,... in the above figure correspond to the times $t_1$, $t_2$,... in Figure 6.

**Figure 7 – Execution timing**





$t_4$: 算法执行开始；

$t_5$: 该算法通过5.2.1.2中定义的WITH限定符完成了与事件输出相关的输出变量值的建立；

$t_6$: 通知资源调度函数算法执行结束；

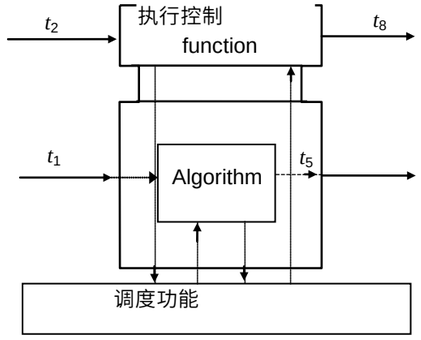
$t_7$: 调度函数调用执行控制函数；

执行控制功能在事件输出处发出事件信号。

如图7所示，在这种情况下，应用设计感兴趣的显着时序延迟是：

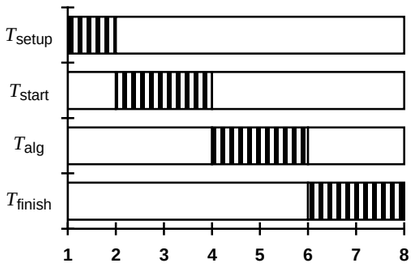
Tstart=t4 t2（从事件输入事件到算法开始执行的时间）

Tfinish=t8 t6（从算法执行结束到事件输出事件的时间）




注意此图仅用于说明。图形表示不规范。

**图6 执行模型**




注意上图中的轴标签1 2 …对应于图6中的时间t1 t2 …。

**图7 执行时序**

Specific requirements for the graphical representation of *function block types* are given in 5.2.1.1.

NOTE 2 Depending on the problem to be solved, various requirements might exist for the synchronization of the values of *input variables* with the *execution* of *algorithms in order to ensure predictability of the results of algorithm execution*. Such requirements could include, for example:

- assurance that the values of variables used by an algorithm remain stable during the execution of the algorithm;
- assurance that the values of variables used by an algorithm correspond to the data present upon the occurrence of the event at the event input which caused the scheduling of the algorithm for execution;
- assurance that the values of variables used by all algorithms scheduled for execution in a function block correspond to the data present upon the occurrence of the event at the event input which caused the scheduling of the first such algorithm for execution.

NOTE 3 *Resources* might need to schedule the *execution* of *algorithms* in a *multitasking* manner. The specification of attributes to facilitate such scheduling is described in Annex G.

### 4.6 Distribution model

As illustrated in Figure 8a, an *application* or *subapplication* can be distributed by allocating its *function block instances* to different *resources* in one or more *devices*. Since the internal details of a function block are hidden from any application or subapplication utilizing it, a function block shall form an atomic unit of distribution. That is, all the elements contained in a given function block instance shall be contained within the same resource.

The functional relationships among the function blocks of an application or subapplication shall not be affected by its distribution. However, in contrast to an application or subapplication confined to a single resource, the timing and reliability of communications functions will affect the timing and reliability of a distributed application or subapplication.

The following clauses apply when applications or subapplications are distributed among multiple resources:

- Clause 6 defines the requirements for communication services to support distribution of applications or subapplications among multiple devices;
- Clause 7 defines the requirements for the case where multiple applications or subapplications are distributed among multiple resources and devices.

### 4.7 Management model

Figures 8b and 8c provide a schematic representation of the management of *resources* and *devices*. Figure 8b illustrates a case in which a *management resource* provides shared facilities for management of other *resources* within a device, while Figure 8c illustrates the distribution of management services among resources within a device. Management *applications* may be modeled using **implementation-dependent** *service interface function blocks* and *communication function blocks*.

NOTE 1 6.3 defines *service interface function block types* for management of *applications*, and IEC 61499-2 provides examples of their usage.

NOTE 2 *Management applications* might contain *service interface function block instances* representing *device* or *resource instances* for the purpose of querying or modifying device or resource *parameters*.



5.2.1.1给出了功能块类型图形表示的具体要求。

注2：根据要解决的问题，输入变量值与算法执行的同步可能存在各种要求，以确保算法执行结果的可预测性。此类要求可能包括，例如：

- 确保算法使用的变量值在算法执行期间保持稳定；
- 确保算法使用的变量值与事件输入发生事件时出现的数据相对应，该事件导致算法调度执行；
- 确保在功能块中调度执行的所有算法使用的变量值对应于在事件输入处发生事件时出现的数据，该事件导致调度第一个此类算法执行。

注3资源可能需要以多任务方式调度算法的执行。促进此类调度的属性规范在附件G中进行了描述。

**分销模式**

如图8a 所示，可以通过将其功能块实例分配给一个或多个设备中的不同资源来分发应用程序或子应用程序。由于功能块的内部细节对任何使用它的应用程序或子应用程序都是隐藏的，因此功能块应形成一个原子分布单元。也就是说，给定功能块实例中包含的所有元素都应包含在同一资源中。

应用程序或子应用程序的功能块之间的功能关系不应受到其分布的影响。然而，与受限于单一资源的应用程序或子应用程序相比，通信功能的时序和可靠性将影响分布式应用程序或子应用程序的时序和可靠性。

当应用程序或子应用程序分布在多个资源中时，以下条款适用：

- 第6章定义了通信服务的要求，以支持在多个设备之间分配应用程序或子应用程序；
- 第7章定义了多个应用程序或子应用程序分布在多个资源和设备之间的情况的要求。

**管理模式**

图8b和8c提供了资源和设备管理的示意图。图8b说明了管理资源为管理设备内的其他资源提供共享设施的情况，而图8c说明了管理服务在设备内的资源之间的分布。管理应用程序可以使用依赖于实现的服务接口功能块和通信功能块来建模。

注16.3定义了用于管理应用程序的服务接口功能块类型，IEC61499-2提供了它们的使用示例。

注2管理应用程序可能包含代表设备或资源实例的服务接口功能块实例，用于查询或修改设备或资源参数。

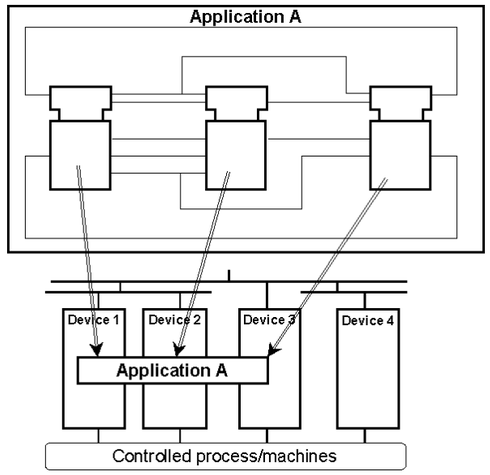



**Figure 8a – Distribution model**

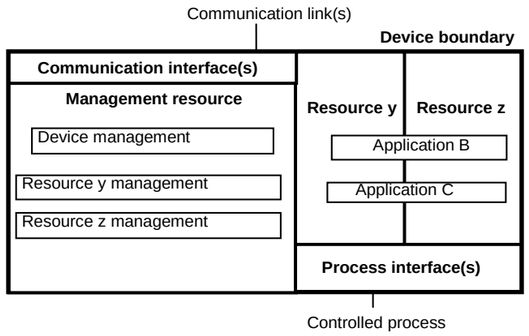



**Figure 8b – Shared management model**

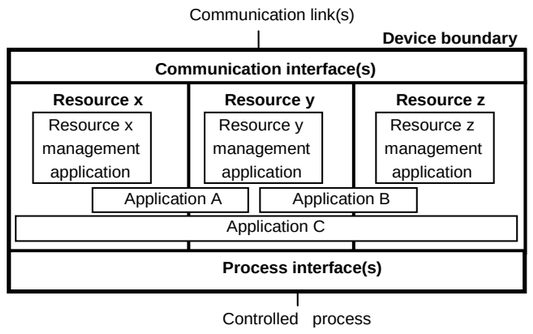



**Figure 8c – Distributed management model**

**Figure 8 – Distribution and management models**





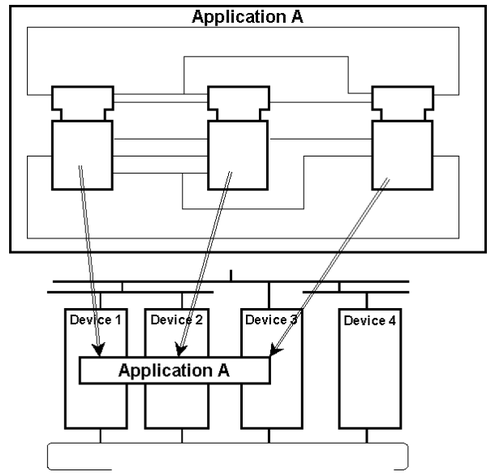



**图8a 分布模型**

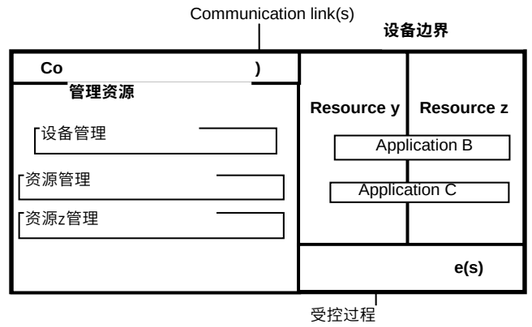



**图8b 共享管理模型**

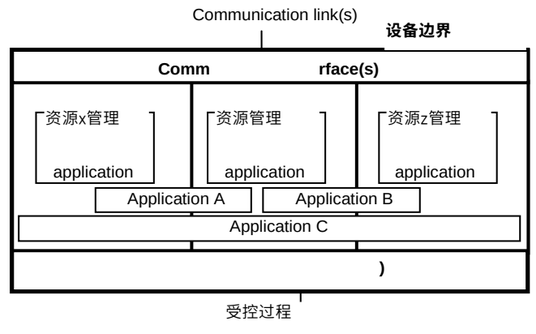



**图8c 分布式管理模型**

**图8 分配和管理模型**

### 4.8 Operational state models

Any given *system* has to be designed, commissioned, operated and maintained. This is modeled through the concept of the system "life cycle". In turn, a system is composed of several *functional units* such as *devices, resources,* and *applications,* each of which has its own life cycle.

Different actions may have to be performed to support *functional units* at each step of the life cycle. To characterize which action can be done and maintain integrity of functional units, "operational states" should be defined, e.g., OPERATIONAL, CONFIGURABLE, LOADED, STOPPED, etc.

Each operational state of a functional unit specifies which actions are authorized, together with an expected behavior.

A system may be organized in such a way that certain functional units may possess or acquire the right of modifying the operational states of other functional units.

Examples of the use of operational states are:

- a functional unit in a RUNNING state, i.e., in execution, may not be able to receive a download action;
- a distributed functional unit may need to maintain a consistent operational state across its components and develop a strategy to propagate changes of operational state through them.

Specific operational states for managed *function block instances* are defined in 6.3.2.

## 5 Specification of function block, subapplication and adapter interface types

### 5.1 Overview

As illustrated in Figure 9, Clause 5 defines the means for the type specification of three kinds of blocks:

- Subclause 5.2 defines the means for specifying and determining the behavior of instances of *basic function block types,* as illustrated in Figure 9a. In this type of function block, execution control is specified by an *execution control chart (ECC),* and the *algorithms* to be executed are declared as specified in compliant Standards as defined in IEC 61499-4.
- Subclause 5.3 defines the means for specifying *composite function block types,* as illustrated in Figure 9b. In this type of function block, algorithms and their execution control are specified through event and data connections in one or more *function block networks.*
- Subclause 5.4 defines the means for specifying *subapplication types,* as illustrated in Figure 9c. In this type of block, algorithms and their execution control are specified as for composite function block types, but with the specific property that *component function blocks* of subapplications may be distributed among several *resources.* Subapplications may be nested, such that the body of a subapplication may also contain *component subapplications.*

Other means may be used for describing the behavior of instances of a function block type. The specification of such means is beyond the scope of this standard; therefore it is required that when such means are used, an unambiguous *mapping* shall be given between their terms and concepts and the corresponding terms and concepts of this standard.





**运行状态模型**

任何给定的系统都必须进行设计、调试、操作和维护。这是通过系统“生命周期”的概念建模的。反过来，一个系统又由设备、资源、应用等若干功能单元组成，每个单元都有自己的生命周期。

在生命周期的每个步骤中，可能必须执行不同的操作来支持功能单元。为了表征可以执行哪些操作并保持功能单元的完整性，应定义“操作状态”，例如，OPERATIONAL、CONFIGURABLE、LOADED、STOPPED等。

功能单元的每个操作状态都指定了授权的操作以及预期的行为。

一个系统的组织方式可以使某些功能单元可以拥有或获得修改其他功能单元的操作状态的权利。

使用操作状态的例子有：

- 处于RUNNING状态的功能单元，即在执行中，可能无法接收下载动作；
- 分布式功能单元可能需要在其组件之间保持一致的操作状态，并制定策略以通过它们传播操作状态的变化。

托管功能块实例的特定操作状态在6.3.2中定义。

## 5 功能块、子应用程序和适配器接口类型的规范

如图9所示，第5条定义了三种块类型规范的方法：

- 子条款5.2定义了用于指定和确定基本功能块类型实例的行为的方法，如图9a所示。在这种类型的功能块中，执行控制由执行控制图(ECC)指定，并且要执行的算法被声明为符合IEC61499-4中定义的标准。
- 子条款5.3定义了指定复合功能块类型的方法，如图9b所示。在这种类型的功能块中，算法及其执行控制是通过一个或多个功能块网络中的事件和数据连接来指定的。
- 子条款5.4定义了指定子应用程序类型的方法，如图9c所示。在这种类型的块中，算法及其执行控制被指定为复合功能块类型，但具有子应用的组件功能块可以分布在多个资源中的特定属性。子应用程序可以嵌套，这样子应用程序的主体也可以包含组件子应用程序。

可以使用其他方式来描述功能块类型的实例的行为。此类装置的规范超出了本标准的范围；因此，要求在使用此类手段时，应在其术语和概念与本标准的相应术语和概念之间给出明确的映射。

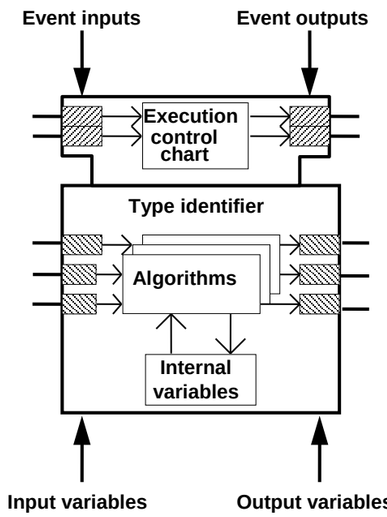



Figure 9a – Basic function block (5.2)

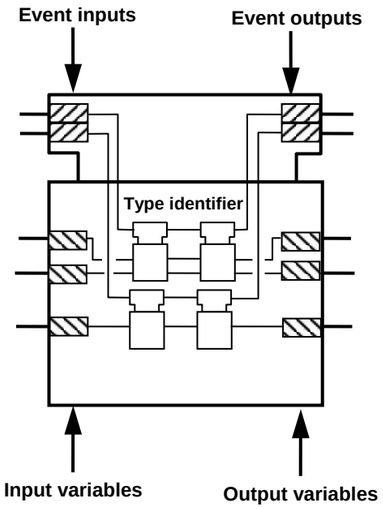
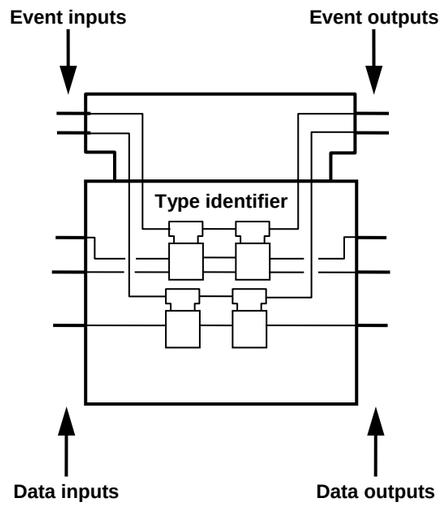



Figure 9b – Composite function block (5.3)




NOTE This figure is illustrative only. The graphical representation is not normative.

Figure 9c – **Subapplications** (5.4)

**Figure 9 – Function block and subapplication types**

## 5.2 Basic function blocks

### 5.2.1 Type declaration

#### 5.2.1.1 General

A *basic function block* utilizes an *execution control chart (ECC)* to control the *execution* of its *algorithms*.

As illustrated in Figure 10, a *basic function block type* can be declared textually according to the syntax specified in Clause B.2 or graphically according to the following rules:





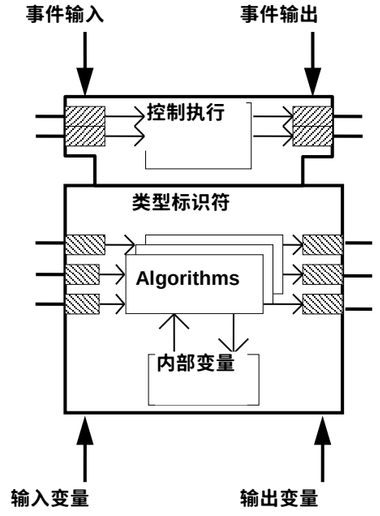



图9a 基本功能块(5.2)

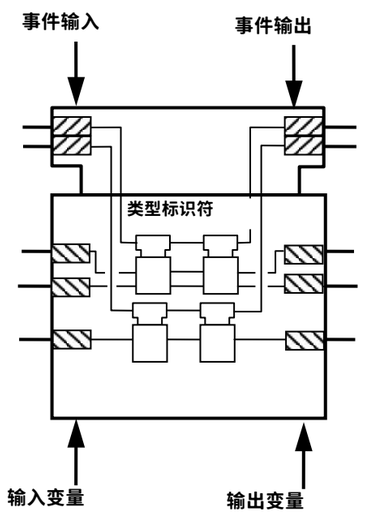
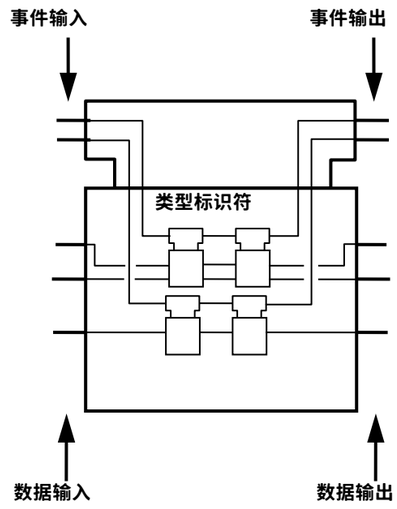



图9b 复合功能块(5.3)




注意此图仅用于说明。图形表示不规范。

**图9 功能块和子应用类型**

## 5.2 基本功能块

### 5.2.1 类型声明

基本功能块利用执行控制图(ECC)来控制其算法的执行。

如图10所示，基本功能块类型可以根据B.2中规定的语法以文本形式声明，也可以根据以下规则以图形形式声明：

a) the function block *type name* is shown at the top center of the lower portion of the block;

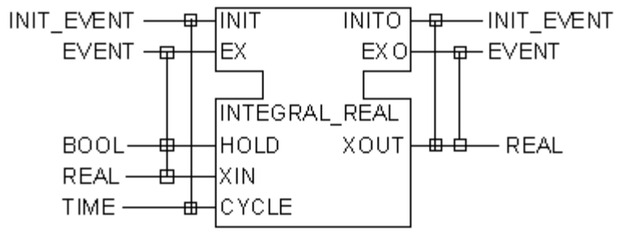
b) the names and *type declarations* of *input variables* and *socket adapters* are shown at the left edge of the lower portion of the block;

c) the names and *type declarations* of *output variables* and *plug adapters* are shown at the right edge of the lower portion of the block;

d) the *interface* of the function block type to *events* is declared in the upper portion of the block as specified in 5.2.1.2;

e) the *algorithms* associated with the function block type are declared as specified in 5.2.1.3;

f) control of the *execution* of the associated algorithms is declared as specified in 5.2.1.4.




NOTE 1 See Annex F for a textual declaration of this example.

NOTE 2 This example is illustrative only. Details of the specification are not normative.

**Figure 10 – Basic function block type declaration**

### 5.2.1.2 Event interface declaration

As shown in Figure 10, the *interface* of a *basic function block type* to *events* can be declared textually according to the syntax given in Clause B.2, or graphically according to the following rules.

a) *Event interfaces* are located in a distinct area at the top of the block.

b) *Event input* names are shown at the left-hand side of the upper portion of the block.

c) *Event output* names are shown at the right-hand side of the upper portion of the block.

d) Event *types* are shown outside the block adjacent to their associated event inputs or outputs.

NOTE 1 If no event type is given for an event input or output, it is considered to be of the default type EVENT.

NOTE 2 An event output of type EVENT can be connected to an event input of any type, and an event input of type EVENT can receive an event of any type.

NOTE 3 An event output of any type other than EVENT can only be connected to an event input of the same type or of type EVENT.

NOTE 4 An event *type* is implicitly declared by its use in an event declaration.

As illustrated in Figure 10 and Annex F, the `WITH` qualifier or a graphical equivalent shall be used to specify an association among *input variables* or *output variables* and an *event* at the associated *event input* or *event output*, respectively.

Each *input variable* and *output variable* appears in zero or more `WITH` clauses or their graphical equivalents.

NOTE 5 This information can be used to determine the required communication *services* when *configuring* a distributed *application* as described in Clause 7.

NOTE 6 An input variable that does not appear in any `WITH` clause cannot be connected with an output variable of another function block. The values of such variables either remain at their declared initial values or are established by management commands such as WRITE, as described in 6.3.2.



a)功能块类型名称显示在功能块下部的顶部中心；

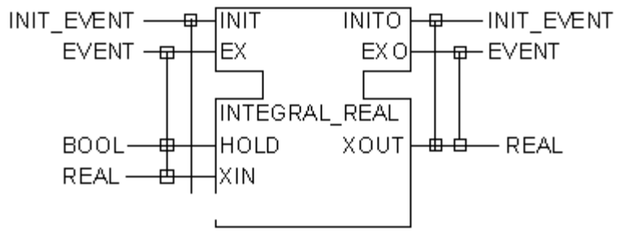
b)输入变量和套接字适配器的名称和类型声明显示在块下部的左边缘；

c)输出变量和插头适配器的名称和类型声明显示在块下部的右边缘；

d)功能块类型与事件的接口在块的上部声明，如5.2.1.2中规定的那样；

e)与功能块类型相关的算法按照5.2.1.3中的规定进行声明；

f) 相关算法执行的控制在5.2.1.4中规定。




注1参见附录F以了解该示例的文本声明。

注2这个例子只是说明性的。规范的细节不是规范性的。

**图10 基本功能块类型声明**

### 事件接口声明

如图10所示，基本功能块类型与事件的接口可以根据第B.2节中给出的语法以文本形式声明，也可以根据以下规则以图形形式声明。

a)事件接口位于块顶部的不同区域。

b)事件输入名称显示在模块上部的左侧。

c)事件输出名称显示在块上部的右侧。

d)事件类型显示在与其相关的事件输入或输出相邻的块外。

注1如果没有为事件输入或输出指定事件类型，则认为它是默认类型EVENT。

注2：EVENT类型的事件输出可以连接到任何类型的事件输入，EVENT类型的事件输入可以接收任何类型的事件。

注3：除EVENT之外的任何类型的事件输出只能连接到相同类型或EVENT类型的事件输入。

注4事件类型通过在事件声明中的使用来隐式声明。

如图10和附录F所示，应使用WITH限定符或图形等效项来分别指定输入变量或输出变量与相关事件输入或事件输出处的 事件之间的关联。

每个输入变量和输出变量都出现在零个或多个WITH子句或其图形等效项中。

注5：当配置第7章所述的分布式应用程序时，该信息可用于确定所需的通信服务。

注6未出现在任何WITH子句中的输入变量不能与另一个功能块的输出变量连接。这些变量的值要么保持在其声明的初始值，要么由管理命令（如WRITE）建立，如6.3.2所述。

NOTE 7 An output variable that does not appear in any WITH clause can be connected to an input variable of another function block or can be "read" by management commands such as READ, as described in 6.3.2.

NOTE 8 See 4.5.3 for an application of the WITH qualifier to the execution model of a basic function block.

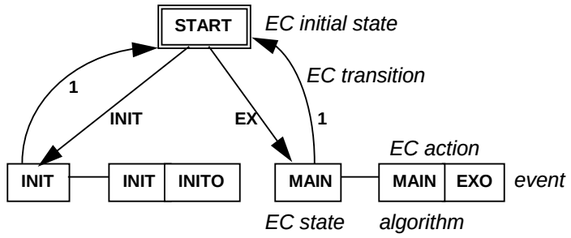



**Figure 11 – ECC example**

### 5.2.2 Behavior of instances

#### 5.2.2.1 Initialization

Initialization of a basic function block *instance* by a *resource* shall be functionally equivalent to the following procedure:

a) The value of each *input*, *output*, and *internal variable* shall be initialized to the corresponding initial value given in the function block *type* specification. If no such initial value is defined, the value of the variable shall be initialized to the default initial value defined for the data type of the variable.

b) Any additional algorithm-specific initializations shall be performed; for example, all *initial steps* of IEC 61131-3 *Sequential Function Charts (SFCs)* shall be activated and all other *steps* shall be deactivated.

c) The *EC initial state* of the function block's *Execution Control Chart (ECC)* shall be activated, all other *EC states* shall be deactivated, and the ECC operation state machine defined in 5.2.2.2 shall be placed in its initial (`S0`) state.

NOTE The conditions under which a resource performs such initialization are **implementation-dependent.**

The function block *type* may also specify an initialization *algorithm* to be performed upon the occurrence of an appropriate event, for example the `INIT` algorithm shown in Figure 11. An *application* can then specify the conditions under which this algorithm is to be executed, for example by connecting an output of an instance of the `E_RESTART` type defined in Annex A to an appropriate event input, for example the `INIT` input shown in Figure 10.

#### 5.2.2.2 Algorithm invocation

*Execution* of an *algorithm* associated with a *function block instance* is *invoked* by a request to the **scheduling function** of the *resource* to schedule the execution of the algorithm's *operations*.

NOTE 1 The operations performed by an algorithm can vary from one execution to the next due to changed internal states of the function block, even though the function block may have only a single algorithm and a single event input triggering its execution.

Algorithm invocation for an instance of a *basic function block type* shall be accomplished by the functional equivalent of the operation of its *execution control chart (ECC).* The operation of the ECC shall exhibit the behavior defined by the state machine in Figure 12 and Table 1.

NOTE 2 It is a consequence of this model that an occurrence of an event at an event input will not cause a transition containing the event to be crossed, if the transition is not associated with the currently active state, i.e., if the event is not relevant in the given state. However, *sampling* of the input variables associated to the event by a WITH construct will occur in any case.



注7：没有出现在任何WITH子句中的输出变量可以连接到另一个功能块的输入变量，或者可以通过READ等管理命令"读取"，如6.3.2所述。

注8参见4.5.3将WITH限定符应用于基本功能块的执行模型。

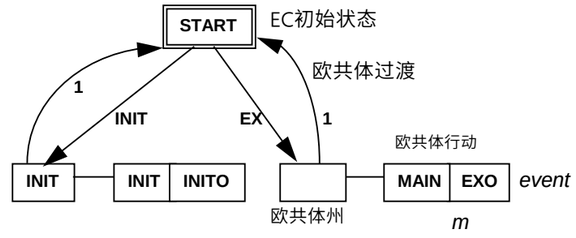



**图11 ECC示例**

### 5.2.2 实例的行为

资源对基本功能块实例的初始化在功能上等同于以下过程：

a)每个输入、输出和内部变量的值应初始化为功能块类型规范中给出的相应初始值。如果没有定义这样的初始值，则变量的值应初始化为为变量的数据类型定义的默认初始值。

b)应执行任何额外的特定于算法的初始化；例如，IEC61131-3顺序功能图(SFC)的所有初始步骤均应激活，所有其他步骤均应停用。

c)功能块的执行控制图(ECC)的EC初始状态应被激活，所有其他EC状态应被停用，并且5.2.2.2中定义的ECC操作状态机应置于其初始(s0)状态。

**注意资源执行这种初始化的条件是依赖于实现的。**

功能块类型还可以指定在发生适当事件时要执行的初始化算法，例如图11中所示的INIT算法。然后应用程序可以指定执行该算法的条件，例如通过将附件A中定义的E_RESTART类型实例的输出连接到适当的事件输入，例如图10中所示的INIT输入。

#### 算法调用

与功能块实例相关联的算法的执行由对资源调度功能的请求调用，以调度算法操作的执行。

注1：由于功能块内部状态的变化，算法执行的操作可能会因功能块的内部状态变化而从一次执行到下一次执行有所不同，即使功能块可能只有一个算法和一个触发其执行的事件输入。

基本功能块类型实例的算法调用应通过其执行控制图（ECC）操作的功能等效来完成。ECC的操作应表现出图12和表1中状态机定义的行为。

注2：如果转换与当前活动状态不相关，即如果事件不与在给定的状态下相关。但是，在任何情况下都将通过WITH构造对与事件关联的输入变量进行采样。

### 5.2.2.3 Algorithm declaration

As shown in Annex F, *algorithms* associated with a *basic function block type* may be included in the function block type declaration according to the rules for declaration of the function block type specification given in Annex B. Other means may also be used for the specification of the identifiers and bodies of algorithms; however, the specification of such means is beyond the scope of this standard.

The declaration of an algorithm may include the declaration of temporary variables that:

- are only visible in the body of the algorithm;
- are initialized upon each invocation of the algorithm;
- may be used and modified during execution of the algorithm; and
- do not have values that persist between executions of the algorithm.

### 5.2.2.4 Declaration of algorithm execution control

The sequencing of algorithm invocations for *basic function block types* may be declared in the function block type specification. If the algorithms of a basic function block type are defined as specified in 5.2.1.3 (or otherwise identified), then the sequencing of algorithm invocation for such a function block can be in the form of an *Execution Control Chart (ECC)* consisting of *EC states, EC transitions*, and *EC actions*. These elements are represented and interpreted as follows:

a) the ECC is included in an *execution control* section of the function block type declaration, considered to reside in the upper portion of the block;

b) the ECC shall contain exactly one *EC initial state,* represented graphically as a double-outlined shape with an associated *identifier*. The EC initial state shall have no associated EC actions;

c) the ECC shall contain one or more *EC states,* represented graphically as single-outlined shapes, each with an associated *identifier*;

d) the ECC can utilize but not modify variables declared in the function block type specification;

e) an *EC state* can have zero or more associated *EC actions.* The association of the EC actions with the EC state can be expressed in graphical or textual form;

f) the *algorithm* (if any) associated with an EC action, and the *event* (if any) to be issued on completion of the algorithm, shall be expressed in graphical or textual form;

g) an *EC transition* is represented graphically or textually as a directed link from one EC state to another (or to the same state);

h) each EC transition shall have an associated transition condition, containing a reference to an *event*, a *guard condition*, or both, expressed in the syntax defined for the non-terminal `ec_transition_condition` in B.2.1.

Figure 11 illustrates the elements of an ECC. Similar textual declarations using the syntax of Clause B.2 are given in Annex F.

NOTE 1 The notation `1` (one), illustrated in Figure 11, is considered to be equivalent to `[TRUE]` representing a transition condition with no associated event and a guard condition that is always `TRUE`.

NOTE 2 In this restricted domain, the same symbol (e.g., `INIT`) can be used to represent an EC state and algorithm name, since the referent of the symbol can be inferred easily from its usage.

NOTE 3 The text in *italics* is not part of the ECC.

NOTE 4 One-to-one association of events with algorithms, as illustrated in this figure, is frequently encountered but is not the only possible usage. See Table A.1 for examples of other usages: The E_SPLIT block shows an association of two event outputs with one state but no algorithms; E_MERGE shows an association of one output event but no algorithms with two event inputs; E_DEMUX shows any of several algorithms associated with a single input event; etc.





**算法声明**

如附件F所示，与基本功能块类型相关的算法可以根据附件B中给出的功能块类型规范的声明规则包含在功能块类型声明中。其他方式也可用于规范算法的标识符和主体；但是，这种方式的规范超出了本标准的范围。

算法的声明可能包括临时变量的声明：

- 仅在算法主体中可见；
- 在每次调用算法时初始化；
- 可以在算法执行期间使用和修改；和
- 没有在算法执行之间持续存在的值。

**算法执行控制声明**

基本功能块类型的算法调用顺序可以在功能块类型规范中声明。如果基本功能块类型的算法按照5.2.1.3中的规定定义（或以其他方式标识），则此类功能块的算法调用顺序可以采用由EC组成的执行控制图(ECC)的形式状态、EC转换和EC动作。这些元素的表示和解释如下：

a)ECC包含在功能块类型声明的执行控制部分中，被认为位于块的上部；

b)ECC应包含准确的一个EC初始状态，用图形表示为带有相关标识符的双轮廓形状。EC初始状态应没有相关的EC动作；

c)ECC应包含一个或多个EC状态，用图形表示为单个轮廓形状，每个都有一个相关的标识符；

d)ECC可以使用但不能修改功能块类型规范中声明的变量；

e)一个EC状态可以有零个或多个相关的EC动作。EC动作与EC状态的关联可以用图形或文本的形式表示；

与EC动作相关的算法（如果有），以及在算法完成时发出的事件（如果有），应以图形或文本形式表示；

g)EC转换以图形或文本方式表示为从一个EC状态到另一个（或到同一状态）的有向链接；

h)每个EC转换应有一个关联的转换条件，包含对事件、保护条件或两者的引用，以B.2.1中为非终端ec_transition_condition定义的语法表示。

图11说明了ECC的元素。使用以下语法的类似文本声明
条款B.2在附录F中给出。

注1：图11所示的符号1（一）被认为等同于[TRUE]，表示没有关联事件的转换条件和始终为TRUE的保护条件。

注2在此受限域中，相同的符号（例如INIT）可用于表示EC状态和算法名称，因为可以从其用法中轻松推断出符号的所指对象。

注3斜体文本不是ECC的一部分。

注4事件与算法的一对一关联，如图所示，经常遇到，但不是唯一可能的用法。有关其他用法的示例，请参见表A.1：E_SPLIT块显示两个事件输出与一个状态但没有算法的关联；E_MERGE显示一个输出事件的关联，但没有具有两个事件输入的算法；E_DEMUX显示与单个输入事件相关的几种算法中的任何一种；等等

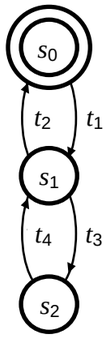



**Figure 12 – ECC operation state machine**

**Table 1 – States and transitions of ECC operation state machine**

| State | | Operations |
|---|---|---|
| $s_0$ | | -- |
| $s_1$ | | evaluate transitions[c,e] |
| $s_2$ | | perform actions[d,e] |
| **Transition** | **Condition** | **Operations** |
| $t_1$ | an input event occurs[a] | Sample inputs[b,e] |
| $t_2$ | no transition is crossed | |
| $t_3$ | a transition is crossed | |
| $t_4$ | actions completed | |

a The resource shall ensure that no more than one input event occurs at any given instant in time.

b This operation consists of *sampling* (or its functional equivalent) of the input variables associated with the current input event by a WITH declaration as described in 5.2.1.2.

c This operation consists of evaluating the transition conditions at the EC transitions following the active EC state and crossing the first EC transition (if any) for which a TRUE guard_condition as defined in B.2.1 is found, according to the following rules:

1 "Crossing the EC transition" shall consist of deactivating its predecessor EC state and activating its successor EC state.

2 The order in which the transition conditions are evaluated shall correspond to the order in which the transitions are declared as defined in B.2.1, or equivalently in the XML syntax defined in IEC 61499-2.

3 The guard_condition of a transition condition containing only an `event_input_name` shall have the default value TRUE.

4 If state $s_1$ was entered via $t_1$, only transition conditions associated with the current input event via its event_input_name as defined in B.2.1, or transition conditions with no event associations, shall be evaluated.

5 If state $s_1$ was entered via $t_4$, only transition conditions with no event associations shall be evaluated.

d This operation consists of, for each *EC action* associated with the active *EC step*, executing the associated algorithm, if any, and issuing an event at the associated event output, if any. The order in which the actions are performed corresponds to the order in which they appear graphically from top to bottom, or to the order in which they are declared following the textual syntax defined in B.2.1, or equivalently in the XML syntax defined in IEC 61499-2.

e All operations performed from an occurrence of transition $t_1$ to an occurrence of $t_2$ shall be implemented as a *critical region* with a lock on the function block instance.

### 5.2.2.5 Algorithm execution

Algorithm *execution* in a basic function block shall consist of the execution of a finite sequence of *operations* determined by **implementation-dependent** rules appropriate to the





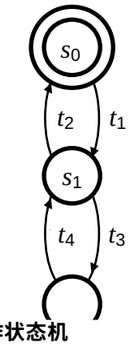



**图12 ECC操作状态机**

**表1 ECC操作状态机的状态和转换**

| State | | Operations |
|---|---|---|
| $s_0$ | | |
| $s_1$ | | 评估转换[c e] |
| $s_2$ | | 执行动作[d e] |
| **Transition** | | |
| $t_1$ | 输入事件发生 | 样本输入[b e] |
| $t_2$ | 没有过渡 | |
| $t_3$ | 跨越了一个过渡 | |
| | 已完成的操作 | |

a 资源应确保在任何给定时刻不超过一个输入事件发生。

b 该操作包括通过5.2.1.2中描述的WITH声明对与当前输入事件关联的输入变量进行采样（或其功能等效项）。

c 此操作包括评估活动EC状态之后的EC转换处的转换条件，并跨越第一个EC转换（如果有），其中定义为TRUEguard_condition
B.2.1找到，按以下规则：

1 “跨越EC过渡”应包括停用其前任EC状态和激活其后继EC状态。

2 评估转换条件的顺序应对应于B.2.1中定义的声明转换的顺序，或等效于IEC61499-2中定义的XML语法。

3 仅包含event_input_name的转换条件的guard_condition应具有默认值TRUE。

4 如果通过t1进入状态s1，则仅应评估通过B.2.1中定义的event_input_name与当前输入事件关联的转换条件，或没有事件关联的转换条件。

如果通过t4进入状态s1，则只应评估没有事件关联的转换条件。

d 对于与活动EC步骤相关联的每个EC操作，此操作包括执行相关算法（如果有）以及在相关事件输出（如果有）处发出事件。执行动作的顺序对应于它们从上到下以图形方式出现的顺序，或者对应于按照B.2.1中定义的文本语法或等效地在IEC中定义的XML语法中声明它们的顺序61499-2。

e 从发生t1到发生t2执行的所有操作都应实现为具有锁定功能块实例的关键区域。

**算法执行**

基本功能块中的算法执行应包括有限操作序列的执行，这些操作由适用于

language in which the algorithm is written, the *resource* in which it executes, and the domain to which it applies. Algorithm execution terminates after execution of the last operation in this sequence.

If an algorithm implements a state machine, repeated executions of the algorithm are necessary to recognize or perform state changes. Normally there is no association between those state changes and the completion of the algorithm. Such associations have to be created by the event output generation facilities described in 5.2.2.2.

## 5.3 Composite function blocks

### 5.3.1 Type specification

The declaration of *composite function block types* shall follow the rules given in 5.2.1 with the exception that *event inputs* and *event outputs* of the *component function blocks* can be interconnected with the event inputs and event outputs of the composite function block to represent the sequencing and causality of function block invocations. The following rules shall apply to this usage:

a) Each event input of the composite function block is connected to exactly one event input of exactly one component function block, or to exactly one event output of the composite function block, with the exception that the graphical shorthand for event splitting shown in Figure A.1 may be employed.

b) Each event input of a component function block is connected to no more than one event output of exactly one other component function block, or to no more than one event input of the composite function block, with the exception that the graphical shorthand for event merging shown in Figure A.1 may be employed.

c) Each event output of a component function block is connected to no more than one event input of exactly one other component function block, or to no more than one event output of the composite function block, with the exception that the graphical shorthand for event splitting shown in Figure A.1 may be employed.

d) Each event output of the composite function block is connected from exactly one event output of exactly one component function block, or from exactly one event input of the composite function block, with the exception that the graphical shorthand for event merging shown in Figure A.1 may be employed.

e) Use of the `WITH` qualifier in the declaration of event inputs of composite function block types is required. Use of the `WITH` qualifier may result in the *sampling* of the associated data inputs as in the case of basic or service interface function blocks, or software tools may provide means of elimination of redundant sampling in the implementation phase.

f) *Instances* of *subapplication types* as defined in 5.4 shall not be used in the specification of a composite function block type.

*Data inputs* and *data outputs* of the *component function blocks* can be interconnected with the data inputs and data outputs of the composite function block to represent the flow of data within the composite function block. The following rules shall apply to this usage:

- Each data input of the composite function block can be connected to zero or more data inputs of zero or more component function blocks, or to zero or more data outputs of the composite function block, or both.
- Each data input of a component function block can be connected to no more than one data output of exactly one other component function block, or to no more than one data input of the composite function block.
- Each data output of a component function block can be connected to zero or more data inputs of zero or more component function blocks, or to zero or more data outputs of the composite function block, or both.
- Each data output of the composite function block shall be connected from exactly one data output of exactly one component function block, or from exactly one data input of the composite function block.



编写算法的语言、执行算法的资源以及应用的领域。算法执行在此序列中的最后一个操作执行后终止。

如果算法实现了状态机，则需要重复执行该算法以识别或执行状态更改。通常，这些状态变化与算法完成之间没有关联。此类关联必须由5.2.2.2中描述的事件输出生成设施创建。

## 5.3 复合功能块

### 型号规格

复合功能块类型的声明应遵循5.2.1中给出的规则，但组件功能块的事件输入和事件输出可以与复合功能块的事件输入和事件输出互连，以表示顺序和功能块调用的因果关系。以下规则应适用于这种用法：

a)复合功能块的每个事件输入都连接到恰好一个组件功能块的一个事件输入，或者连接到复合功能块的一个事件输出，除了图A中所示的事件拆分的图形简写.1可以使用。

b)一个组件功能块的每个事件输入连接到恰好一个其他组件功能块的不超过一个事件输出，或连接到复合功能块的不超过一个事件输入，除了事件的图形简写可以采用图A.1所示的合并。

c)一个组件功能块的每个事件输出连接到恰好一个其他组件功能块的不超过一个事件输入，或连接到复合功能块的不超过一个事件输出，除了事件的图形简写可以采用图A.1所示的分割。

d)复合功能块的每个事件输出与恰好一个组件功能块的一个事件输出相连接，或从复合功能块的一个事件输入连接，但图A所示的事件合并的图形简写除外.1可以使用。

e)需要在复合功能块类型的事件输入声明中使用WITH限定符。使用WITH限定符可能会导致相关数据输入的采样，如在基本或服务接口功能块的情况下，或者软件工具可以提供在实现阶段消除冗余采样的方法。

f) 5.4中定义的子应用类型的实例不得用于复合功能块类型的规范中。

组件功能块的数据输入和数据输出可以与复合功能块的数据输入和数据输出互连，以表示复合功能块内的数据流。以下规则应适用于这种用法：

- 复合功能块的每个数据输入可以连接到零个或多个组件功能块的零个或多个数据输入，或连接到复合功能块的零个或多个数据输出，或两者。
- 一个组件功能块的每个数据输入可以连接到不超过一个恰好另一个组件功能块的数据输出，或者连接到复合功能块的不超过一个数据输入。
- 组件功能块的每个数据输出可以连接到零个或多个组件功能块的零个或多个数据输入，或连接到复合功能块的零个或多个数据输出，或两者。
- 复合功能块的每个数据输出应与恰好一个组件功能块的恰好一个数据输出相连接，或者与复合功能块的恰好一个数据输入相连接。

NOTE 1 If an element declared in a VAR_INPUT...END_VAR or VAR_OUTPUT...END_VAR construct is associated with an input or output event, respectively, by a WITH construct, this will result in the creation of an associated input or output variable, respectively, as in the case of basic function block types. If such an element is not associated with an input or output event, then the associated data flow is passed directly to or from the component function blocks via the connections described above.

NOTE 2 The rules for interconnection of the event and variable inputs and outputs of *plugs* and *sockets* in the body of the composite function block are the same as for the interconnection of the inputs and outputs of the *component function blocks*. See 5.5 for further requirements regarding *adapter interfaces*.

Figure 13 illustrates the application of these rules to the example PI_REAL function block. Figure 13a shows the graphical representation of the external interfaces and 13b shows the graphical construction of its body. Figure 14 shows the interfaces and execution control for the function block type PID_CALC used in the body of the PI_REAL example.

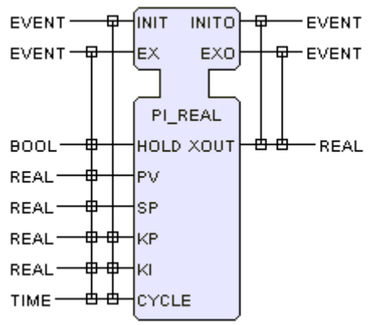



**Figure 13a – External interface**

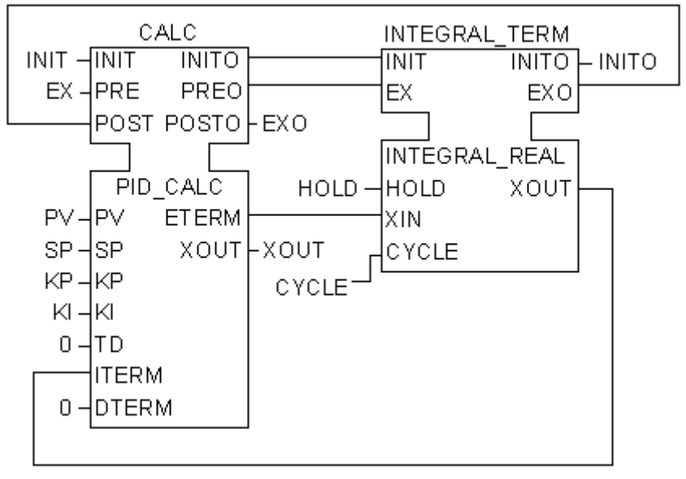



**Figure 13b – Graphical body**

NOTE 1 A full textual declaration of this function block type is given in Annex F.

NOTE 2 This example is illustrative only. Details of the specification are not normative.

**Figure 13 – Composite function block PI_REAL example**

注1如果在VAR_INPUT…END_VAR或VAR_OUTPUT…END_VAR构造中声明的元素分别通过WITH构造与输入或输出事件相关联，这将导致创建关联的输入或输出变量，分别，如在基本功能块类型的情况下。如果这样的元素不与输入或输出事件相关联，则相关联的数据流通过上述连接直接传入或传出组件功能块。

注2：复合功能块主体中插头和插座的事件和变量输入输出互连规则与组件功能块输入输出互连规则相同。有关适配器接口的进一步要求，请参见5.5。

图13说明了这些规则在示例PI_REAL功能块中的应用。图13a显示了外部接口的图形表示，13b显示了其主体的图形结构。图14显示了PI_REAL示例主体中使用的功能块类型PID_CALC的接口和执行控制。

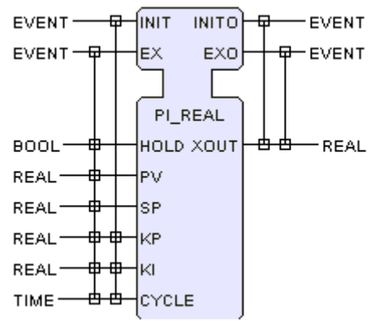



**图13a 外部接口**

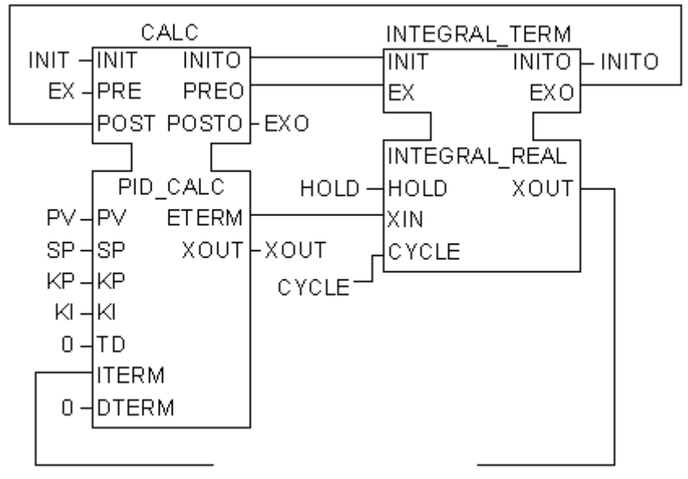



**图13b 图形主体**

注1该功能块类型的全文声明在附录F中给出。

注2这个例子只是说明性的。规范的细节不是规范性的。

**图13 复合功能块PI_REAL示例**

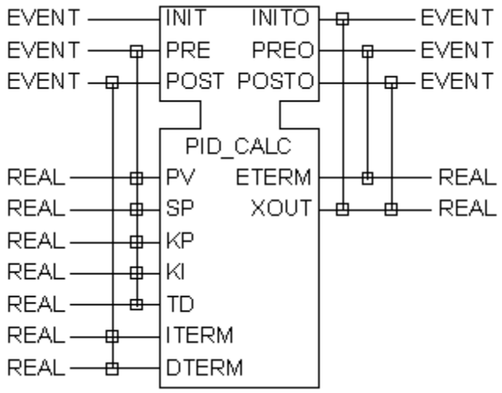



**Figure 14a – External interface**

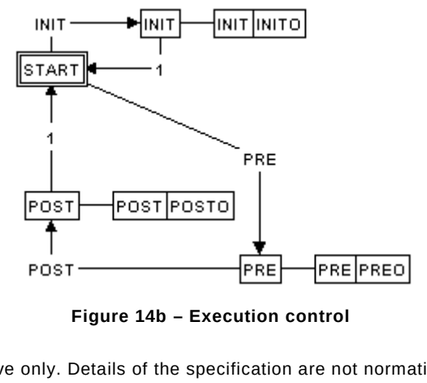



**Figure 14b – Execution control**

NOTE This example is illustrative only. Details of the specification are not normative.

**Figure 14 – Basic function block `PID_CALC` example**

### 5.3.2 Behavior of instances

*Invocation* and *execution* of *component function blocks* in composite function blocks shall be accomplished as follows.

a) If an *event input* of the composite function block is connected to an *event output* of the block, occurrence of an *event* at the event input shall cause the generation of an event at the associated event output.

b) If an event input of the composite function block is connected to an event input of a component function block, occurrence of an event at the event input of the composite function block shall cause the scheduling of an invocation of the execution control function of the component function block, with an occurrence of an event at the associated event input of the component function block.

c) If an event output of a component function block is connected to an event input of a second component function block, occurrence of an event at the event output of the first block shall cause the scheduling of an invocation of the execution control function of the second block, with an occurrence of an event at the associated event input of the second block.

d) If an event output of a component function block is connected to an event output of the composite function block, occurrence of an event at the event output of the component block shall cause the generation of an event at the associated event output of the composite function block.





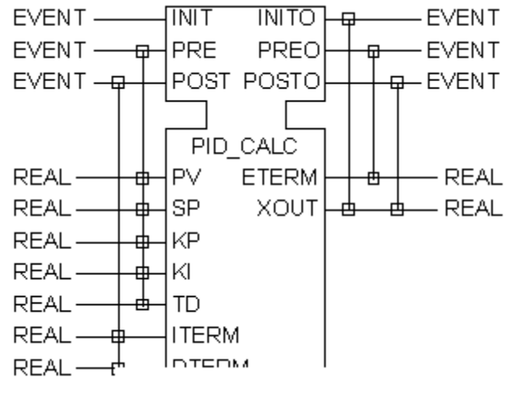



**图14a 外部接口**

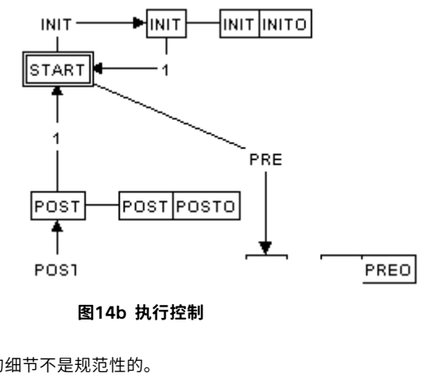



**图14b 执行控制**

注意这个例子只是说明性的。规范的细节不是规范性的。

**图14 基本功能块PID_CALC示例**

**实例的行为**

复合功能块中组件功能块的调用和执行应按如下方式完成。

a)如果复合功能块的事件输入连接到块的事件输出，则在事件输入处发生事件应导致在相关事件输出处产生事件。

b)如果复合功能块的事件输入连接到组件功能块的事件输入，在复合功能块的事件输入处发生事件将导致调度调用的执行控制功能组件功能块，在组件功能块的相关事件输入处发生事件。

c)如果一个组件功能块的事件输出连接到第二个组件功能块的事件输入，则在第一个功能块的事件输出处发生事件将导致调度该组件的执行控制功能的调用。第二个块，在第二个块的关联事件输入处发生事件。

d)如果组件功能块的事件输出连接到复合功能块的事件输出，则在组件块的事件输出处发生事件应导致在复合的相关事件输出处产生事件功能块。

Initialization of instances of composite function blocks shall be equivalent to initialization of their component function blocks according to the provisions of 5.2.2.1.

## 5.4 Subapplications

### 5.4.1 Type specification

The declaration of *subapplication types* is similar to the declaration of *composite function block types* as defined in 5.3.1, with the exception that the delimiting keywords shall be SUBAPPLICATION..END_SUBAPPLICATION. The following rules shall apply to this usage:

a) The WITH qualifier is not used in the declaration of event inputs and event outputs of *subapplication types*.

b) Each event input of the subapplication shall be connected to exactly one event input of exactly one component function block or component subapplication, or to exactly one event output of the subapplication.

c) Each event input of a component function block or component subapplication is connected to no more than one event output of exactly one other component function block or component subapplication, or to no more than one event input of the subapplication.

d) Each event output of a component function block or component subapplication is connected to no more than one event input of exactly one other component function block or component subapplication, or to no more than one event output of the subapplication.

e) Each event output of the subapplication is connected from exactly one event output of exactly one component function block or component subapplication, or from exactly one event input of the subapplication.

NOTE 1 Component function blocks can include instances of the event processing blocks defined in Annex A, for example to "split" events using instances of the E_SPLIT block, to "merge" events using instances of the E_MERGE block, or for both cases, using the equivalent graphical shorthand.

*Data inputs* and *data outputs* of the *component function blocks* or *component subapplications* can be interconnected with the data inputs and data outputs of the subapplication to represent the flow of data within the subapplication. The following rules shall apply to this usage:

- Each data input of the subapplication can be connected to zero or more data inputs of zero or more component function blocks or component subapplications, or to zero or more data outputs of the subapplication, or both.
- Each data input of a component function block or component subapplication can be connected to no more than one data output of exactly one other component function block or component subapplication, or to no more than one data input of the subapplication.
- Each data output of a component function block or component subapplication can be connected to zero or more data inputs of zero or more component function blocks or component subapplications, or to zero or more data outputs of the subapplication, or both.
- Each data output of the subapplication shall be connected from exactly one data output of exactly one component function block or component subapplication, or from exactly one data input of the subapplication.

NOTE 2 Although the VAR_INPUT...END_VAR and VAR_OUTPUT...END_VAR constructs are used for the declaration of the data inputs and outputs of subapplication types, this does not result in the creation of input and output variables; the data flow is instead passed to the component function blocks or component subapplications via the connections described above.

NOTE 3 The rules for interconnection of the event and variable inputs and outputs of *plugs* and *sockets* in the body of the subapplication are the same as for the interconnection of the inputs and outputs of the *component function blocks*. See 5.5 for further requirements regarding *adapter interfaces*.

EXAMPLE Figure 15 illustrates the application of these rules to the example PI_REAL_APPL subapplication. Figure 15a shows the graphical representation of its external interfaces and Figure 15b shows the graphical construction of its body. The body of the PI_REAL_APPL subapplication example uses the function block type PID_CALC from the composite function block example in 5.3.1, which is shown in Figure 14.



复合功能块实例的初始化应等同于按5.2.2.1的规定对其组成功能块的初始化。

## 5.4 S

### 型号规格

子应用类型的声明类似于5.3.1中定义的复合功能块类型的声明，不同之处在于定界关键字应为SUBAPPLICATION..END_SUBAPPLICATION。以下规则应适用于这种用法：

a)WITH限定符不用于子应用程序类型的事件输入和事件输出的声明。

b)子应用程序的每个事件输入应连接到恰好一个组件功能块或组件子应用程序的一个事件输入，或连接到子应用程序的一个事件输出。

c)组件功能块或组件子应用程序的每个事件输入连接到恰好一个其他组件功能块或组件子应用程序的不超过一个事件输出，或连接到子应用程序的不超过一个事件输入。

d)组件功能块或组件子应用程序的每个事件输出连接到恰好另一个组件功能块或组件子应用程序的不超过一个事件输入，或连接到子应用程序的不超过一个事件输出。

e)子应用程序的每个事件输出与恰好一个组件功能块或组件子应用程序的一个事件输出相连接，或者与子应用程序的一个事件输入相连接。

注1组件功能块可以包括附件A中定义的事件处理块的实例，例如使用E_SPLIT块的实例"拆分"事件，使用E_MERGE块的实例"合并"事件，或在这两种情况下，使用等效的图形速记。

组件功能块或组件子应用程序的数据输入和数据输出可以与子应用程序的数据输入和数据输出互连，以表示子应用程序内的数据流。以下规则应适用于这种用法：

- 子应用程序的每个数据输入可以连接到零个或多个组件功能块或组件子应用程序的零个或多个数据输入，或连接到子应用程序的零个或多个数据输出，或两者。
- 组件功能块或组件子应用程序的每个数据输入可以连接到恰好一个其他组件功能块或组件子应用程序的不超过一个数据输出端，或者连接到子应用程序的不超过一个数据输入端。
- 组件功能块或组件子应用程序的每个数据输出可以连接到零个或多个组件功能块或组件子应用程序的零个或多个数据输入，或连接到子应用程序的零个或多个数据输出，或两者。
- 子应用程序的每个数据输出应连接到恰好一个组件功能块或组件子应用程序的一个数据输出端，或者从子应用程序的一个数据输入端连接。

注2虽然VAR_INPUT…END_VAR和VAR_OUTPUT…END_VAR结构用于声明子应用程序类型的数据输入和输出，但这不会导致输入和输出变量的创建；而是通过上述连接将数据流传递给组件功能块或组件子应用程序。

注3：子应用程序主体中插头和插座的事件和变量输入输出互连规则与组件功能块输入输出互连规则相同。有关适配器接口的进一步要求，请参见5.5。

示例图15说明了将这些规则应用于示例PI_REAL_APPL子应用程序。图15a显示了其外部接口的图形表示，图15b显示了其主体的图形结构。PI_REAL_APPL子应用示例的主体使用5.3.1中复合功能块示例中的功能块类型PID_CALC，如图14所示。

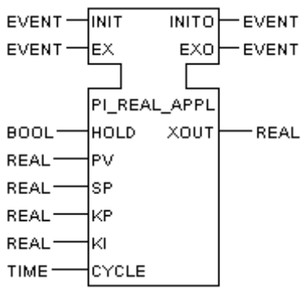



**Figure 15a – External interface**

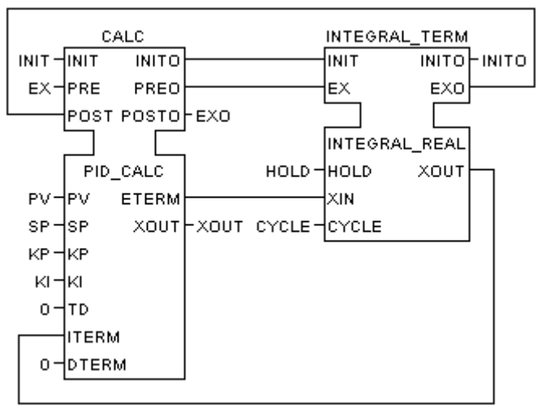



**Figure 15b – Graphical body**

NOTE 1 A full textual declaration of this subapplication type is given in Annex F.

NOTE 2 This example is illustrative only. Details of the specification are not normative.

**Figure 15 – Subapplication PI_REAL_APPL example**

### 5.4.2 Behavior of instances

*Invocation* of the *operations* of *component function blocks* or *component subapplications* within subapplications shall be accomplished as follows:

a) If an *event input* of the subapplication is connected to an *event output* of the block, occurrence of an *event* at the event input shall cause the generation of an event at the associated event output.

b) If an event input of the subapplication is connected to an event input of a component function block or component subapplication, occurrence of an event at the event input of the subapplication shall cause the scheduling of an invocation of the execution control function of the component function block or component subapplication, with an occurrence of an event at the associated event input of the component function block or component subapplication.

c) If an event output of a component function block or component subapplication is connected to an event input of a second component function block or component subapplication, occurrence of an event at the event output of the first block shall cause the scheduling of an invocation of the execution control function of the second block, with an occurrence of an event at the associated event input of the second block.





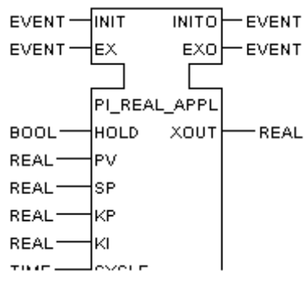



**图15a 外部接口**

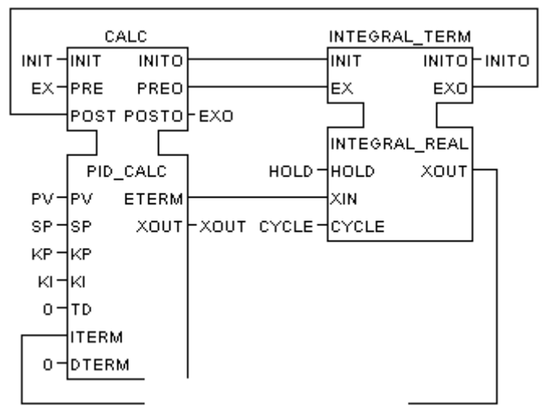



**图15b 图形主体**

注1：附件F中给出了该子应用类型的全文声明。

注2这个例子只是说明性的。规范的细节不是规范性的。

**图15 子应用PI_REAL_APPL示例**

**实例的行为**

在子应用程序内调用组件功能块或组件子应用程序的操作应完成如下：

a)如果子应用程序的事件输入连接到块的事件输出，则在事件输入处发生事件应导致在相关事件输出处生成事件。

b)如果子应用程序的事件输入连接到组件功能块或组件子应用程序的事件输入，则子应用程序的事件输入处发生事件将导致调度组件的执行控制功能的调用功能块或组件子应用程序，在组件功能块或组件子应用程序的关联事件输入处发生事件。

c)如果组件功能块或组件子应用程序的事件输出连接到第二个组件功能块或组件子应用程序的事件输入，则在第一个块的事件输出处发生事件应导致调度调用第二块的执行控制功能，在第二块的相关事件输入处发生事件。

d) If an event output of a component function block or component subapplication is connected to an event output of the subapplication, occurrence of an event at the event output of the component block shall cause the generation of an event at the associated event output of the subapplication.

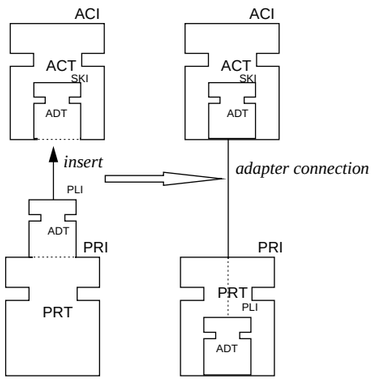
Since subapplications do not explicitly create variables, no specific initialization procedures are applicable to subapplication instances.

## 5.5 Adapter interfaces

### 5.5.1 General principles

*Adapter interfaces* can be used to provide a compact representation of a specified set of event and data flows. As illustrated in Figure 16, an *adapter interface type* provides a means for defining a subset (the *plug adapter)* of the *inputs* and *outputs* of a *provider* function block which can be inserted into a matching subset of corresponding *outputs* and *inputs* (the *socket adapter)* of an *acceptor* function block. Thus, the adapter interface represents the event and data paths by which the provider supplies a *service* to the acceptor, or vice versa, depending on the patterns of provider/acceptor interactions, which may be represented by sequences of *service primitives* as described in 6.1.3.

NOTE A given *function block type* might function as a *provider,* an *acceptor*, or both, or neither, and may contain more than one *plug* or *socket instance* of one or more *adapter interface types.*




**Key**

PRT Provider type
PRI Provider instance
ACT Acceptor type
ACI Acceptor instance
ADT Adapter type
PLI Plug instance
SKI Socket instance

NOTE This figure is illustrative only. The graphical representation is not normative.

**Figure 16 – Adapter interfaces – Conceptual model**

### 5.5.2 Type specification

An *adapter interface type declaration* shall define only the *interface type* name and its contained *event* and *data interfaces.* These are defined graphically or textually in the same manner as the *type name, event interfaces* and *data interfaces* of a *basic function block type* as defined in 5.2.1.1 and 5.2.1.2, with the exception that the keywords for beginning and ending the textual type declaration shall be `ADAPTER...END_ADAPTER`. Textual syntax for the declaration of adapter interfaces is given in Clause B.7.



d)如果组件功能块或组件子应用程序的事件输出连接到子应用程序的事件输出，则在组件块的事件输出处发生事件应导致在相关联的事件输出处生成事件。子应用程序。

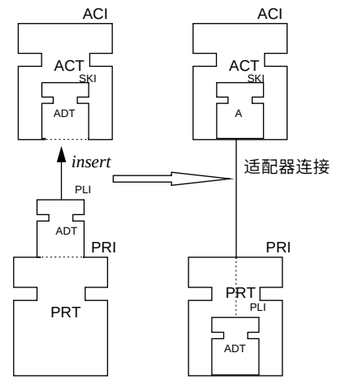
由于子应用程序没有显式创建变量，因此没有特定的初始化过程适用于子应用程序实例。

## 5.5 适配器接口

### 一般原则

适配器接口可用于提供一组指定事件和数据流的紧凑表示。如图16所示，适配器接口类型提供了一种方法，用于定义提供程序功能块的输入和输出的子集（插头适配器），该功能块可以插入到相应输出和输入的匹配子集中（插座适配器）接受器功能块。因此，适配器接口表示提供者向接受者提供服务的事件和数据路径，反之亦然，这取决于提供者接受者交互的模式，这可以由服务原语序列表示，如6.1.3中所述.

注：一个给定的功能块类型可能作为提供者、接受者，或两者兼有，或两者都不是，并且可能包含一种或多种适配器接口类型的多个插头或插座实例。




**Key**

PRT 提供者类型
PRI 提供者实例
ACT
ACI nce
ADT 适配器类型
PLI 插件实例
套接字实例

注意此图仅用于说明。图形表示不规范。

**图16 适配器接口 概念模型**

### 型号规格

适配器接口类型声明应仅定义接口类型名称及其包含的事件和数据接口。它们以与5.2.1.1和5.2.1.2中定义的基本功能块类型的类型名称、事件接口和数据接口相同的方式以图形或文本的方式定义，除了用于开始和结束文本类型的关键字声明应为ADAPTER...END_ADAPTER。适配器接口声明的文本语法在条款B.7中给出。

EXAMPLE The adapter interface illustrated in Figure 17 represents the operation of transferring a workpiece from an "upstream" piece of transfer equipment represented by a *provider* of the *plug* adapter to a "downstream" piece of equipment represented by an *acceptor* with a corresponding *socket* adapter. As illustrated in Figure 17b, the typical operation of this interaction consists of the following sequence:

a) An event in the upstream equipment, e.g., arrival of a workpiece at the unload position, causes a LD event, typically interpreted as a "load" command, to be transmitted to the downstream equipment. Associated with this event is a sensor value WO, indicating whether a workpiece is actually present for transfer, plus some measured property or set of properties of the workpiece, in this case its color.

b) A subsequent event in the downstream equipment, e.g., completion of the load setup, causes an UNLD event, typically interpreted as a command to release the workpiece, to be sent to the upstream equipment.

c) Subsequently a CNF event, typically interpreted as confirmation of the workpiece release, is passed from the upstream to the downstream equipment to complete the operation. At this point the WO output is typically FALSE and the value of the WKPC output has no significance.

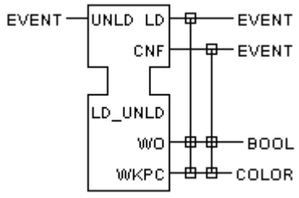



**Figure 17a – Interface**

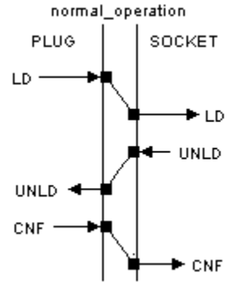



**Figure 17b – Service sequence**

NOTE 1 A full textual declaration of this adapter type is given in Annex F.

NOTE 2 This example is illustrative only. Details of the specification are not normative.

NOTE 3 See 6.1.2 for an explanation of service sequences.

**Figure 17 – Adapter type declaration – graphical example**

### 5.5.3 Usage

The usage of *adapter interface types* and *instances* shall be according to the following rules:

a) Adapter interface instances to be used as *plugs* in instances of a *function block type* shall be declared in its *type declaration* in a `PLUGS...END_PLUGS` block, declaring the *instance name* and *adapter interface type* of each plug. In the graphical representation of *function block types* and *instances*, plugs are shown as *output variables* with specialized textual or graphical indication to show that they are not ordinary output variables.

b) Adapter interface instances to be used as *sockets* in instances of a *function block type* shall be declared in its *type declaration* in a `SOCKETS...END_SOCKETS` block, declaring the *instance name* and *adapter interface type* of each socket. In the graphical representation of *function block types* and *instances*, sockets are shown as *input variables* with specialized textual or graphical indication to show that they are not ordinary input variables.

c) *Inputs* and *outputs* of a *plug* shall be used within its *function block type declaration* in the same manner as inputs and outputs of the function block.

d) *Inputs* and *outputs* of a *socket* shall be used within its *function block type declaration* in the same manner as *outputs* and *inputs* of the function block, respectively.

e) Insertion of *plugs* into *sockets* shall be specified in an `ADAPTER_CONNECTIONS ... END_CONNECTIONS` block in the *declaration* of the *application, subapplication, resource type, resource instance,* or *composite function block type* containing the respective *provider* and *acceptor* instances.



示例图17所示的适配器接口表示将工件从由插头适配器的提供者代表的"上游"传输设备件转移到由具有相应插座适配器的接受器代表的"下游"设备件的操作。如图17b所示，这种交互的典型操作包括以下序列：

a) 上游设备中的事件（例如，工件到达卸载位置）会导致LD事件（通常解释为"加载"命令）传输到下游设备。与该事件相关联的是传感器值WO，指示工件是否实际存在以供传送，加上工件的一些测量属性或一组属性，在本例中为它的颜色。

b) 下游设备中的后续事件（例如，加载设置的完成）会导致UNLD事件（通常解释为释放工件的命令）发送到上游设备。

c) 随后，CNF事件（通常解释为确认工件释放）从上游设备传递到下游设备以完成操作。此时，WO输出通常为FALSE，并且WKPC输出的值没有意义。

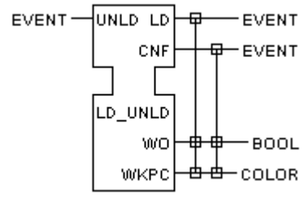



**图17a 界面**

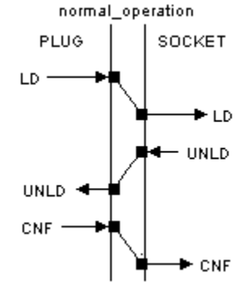



**图17b 服务顺序**

注1：附件F中给出了该适配器类型的全文声明。

注2这个例子只是说明性的。规范的细节不是规范性的。

注3见6.1.2对服务序列的解释。

**图17 适配器类型声明 图形示例**

适配器接口类型和实例的使用应遵循以下规则：

a)在功能块类型实例中用作插件的适配器接口实例应在PLUGS...END_PLUGS块的类型声明中声明，声明每个插件的实例名称和适配器接口类型。在功能块类型和实例的图形表示中，插头显示为输出变量，带有专门的文本或图形指示，以表明它们不是普通的输出变量。

b)在功能块类型实例中用作套接字的适配器接口实例应在SOCKETS...END_SOCKETS块的类型声明中声明，声明每个套接字的实例名称和适配器接口类型。在功能块类型和实例的图形表示中，套接字显示为输入变量，带有专门的文本或图形指示，以表明它们不是普通的输入变量。

c)插头的输入和输出应在其功能块类型声明中以与功能块的输入和输出相同的方式使用。

d)套接字的输入和输出应在其功能块类型声明中分别以与功能块的输出和输入相同的方式使用。

e)应在应用程序、子应用程序、资源类型、资源实例或包含相应提供者和接受者实例的复合功能块类型的声明中的ADAPTER_CONNECTIONS...END_CONNECTIONS块中指定插头插入插座。

f) In the body of a *composite function block type* or *subapplication,* a *socket* is represented as a *function block* with the same inputs and outputs as the corresponding *adapter interface type*. Similarly, in this case a *plug* is represented as a function block with the inputs and outputs of the corresponding adapter interface type reversed.

g) Insertion of plugs into sockets shall be subject to the following constraints:

1) a plug can only be inserted into a socket of the same *adapter interface type;*
2) a plug can only be inserted into zero or one socket at a time;
3) a socket can only accept zero or one plug at a time;
4) a plug can only be inserted in a socket if both are in the same *composite function block, resource*, *application* or *subapplication*.

A connection from a plug to a socket may be shown in an *application* or *subapplication* even though the corresponding function block instances may be *mapped* to separate *resources*. In this case appropriate means, such as communication service interface function blocks as described in 6.2, shall be used to implement the corresponding transfer of events and data among resources.

*Management function blocks* as described in 6.3 may provide facilities for the dynamic creation, deletion, and querying of adapter connections.

EXAMPLE 1 An instance of the XBAR_MVCA type illustrated in Figure 18 acts as both a provider of a plug interface (LDU_PLG) and an acceptor with a socket interface (LDU_SKT). In so doing, it serves to abstract and encapsulate the interactions of an instance of the XBAR_MVC type with "upstream" and "downstream" functional units.

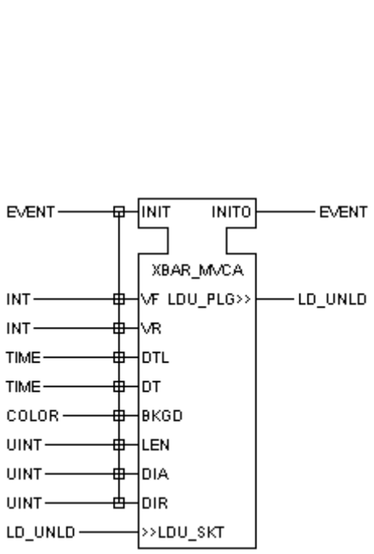



**Figure 18a – Interface**

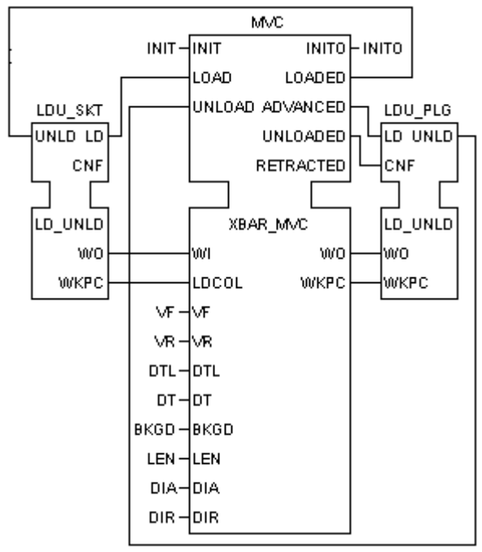



**Figure 18b – Body**

NOTE 1 A full textual declaration of this example is given in Annex F.

NOTE 2 This example is illustrative only. Details of the specification are not normative.

NOTE 3 Although this example presents only a composite type, *provider* and *acceptor* function block *types* can be either *basic* or *composite*.

**Figure 18 – Illustration of provider and acceptor function block type declarations**

f) 在复合功能块类型或子应用程序的主体中，套接字表示为具有与相应适配器接口类型相同的输入和输出的功能块。类似地，在这种情况下，插头被表示为一个功能块，其相应适配器接口类型的输入和输出颠倒了。

g)将插头插入插座应受以下约束：

1)插头只能插入同种适配器接口类型的插座；

2)一个插头一次只能插入零个或一个插座；

3)一个插座一次只能接零个或一个插头；

4)只有在同一个复合功能块、资源、应用程序或子应用程序中，插头才能插入插座。

从插头到插座的连接可能会显示在应用程序或子应用程序中，即使相应的功能块实例可能映射到单独的资源。在这种情况下，应使用适当的手段，例如6.2中描述的通信服务接口功能块，来实现资源之间相应的事件和数据传输。

6.3中描述的管理功能块可以为适配器连接的动态创建、删除和查询提供便利。

示例1图18中所示的XBAR_MVCA类型的实例既充当插头接口(LDU_PLG)的提供者，又充当具有套接字接口(LDU_SKT)的接受者。这样做时，它用于抽象和封装XBAR_MVC类型的实例与“上游”和“下游”功能单元的交互。

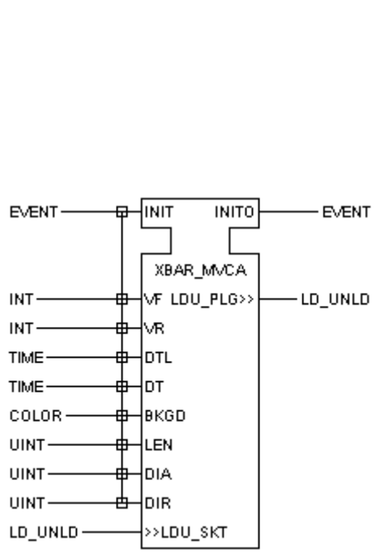



**图18a 界面**

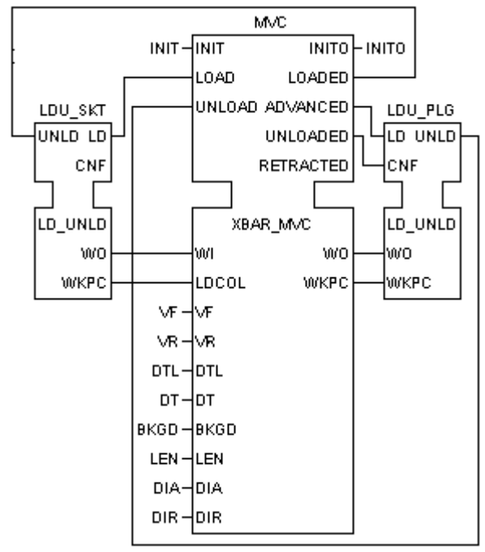



**图18b 主体**

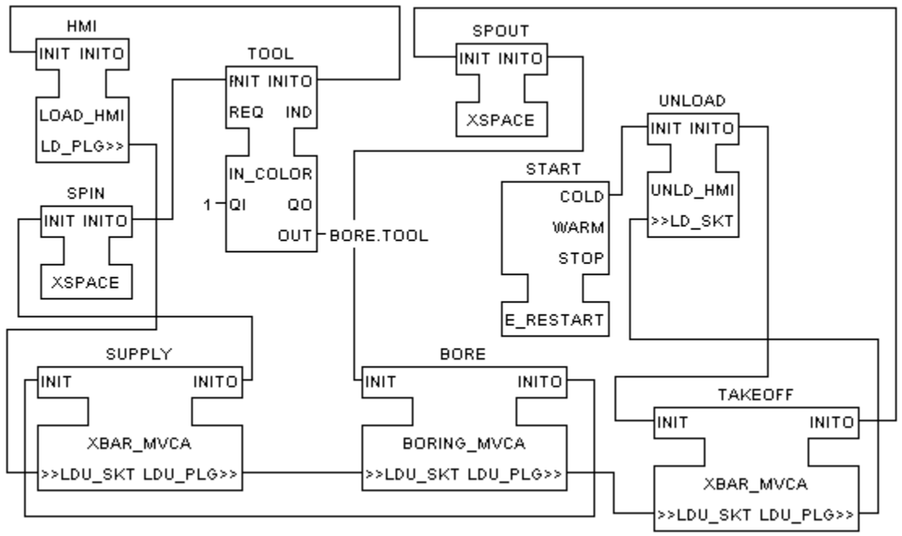
注1：附录F中给出了该示例的全文声明。

注2这个例子只是说明性的。规范的细节不是规范性的。

注3虽然这个例子只展示了一种复合类型，但提供者和接受者功能块类型可以是基本的或复合的。

**图18 提供者和接受者功能块类型声明的说明**

EXAMPLE 2
Figure 19 illustrates a resource configuration containing two instances of the XBAR_MVCA type illustrated in Figure 18. The SUPPLY instance acts as an acceptor ("downstream unit") for the HMI block and a provider ("upstream unit") for the BORE block, while the TAKEOFF instance fulfills corresponding roles for the BORE and UNLOAD blocks, respectively.




NOTE 1 This example is illustrative only. Details of the specification are not normative.

NOTE 2 *Parameter* connections are omitted in this diagram for clarity.

NOTE 3 Type declarations for blocks other than the XBAR_MVCA type are not given in Annex F.

**Figure 19 – Illustration of adapter connections**

### 5.6 Exception and fault handling

Additional facilities for the prevention, recognition and handling of *exceptions* and *faults* may be provided by *resources.* Such capabilities may be modeled as *service interface function blocks*. The definition of specific function block types for prevention, recognition and handling of exceptions and faults is beyond the scope of this standard. However, `INIT-`, `CNF-` and `IND-` outputs of service interface function blocks, and the associated `STATUS` values, may be used to indicate the occurrence and type of exceptions and faults, as noted in 6.1.3.

## 6 Service interface function blocks

### 6.1 General principles

#### 6.1.1 General

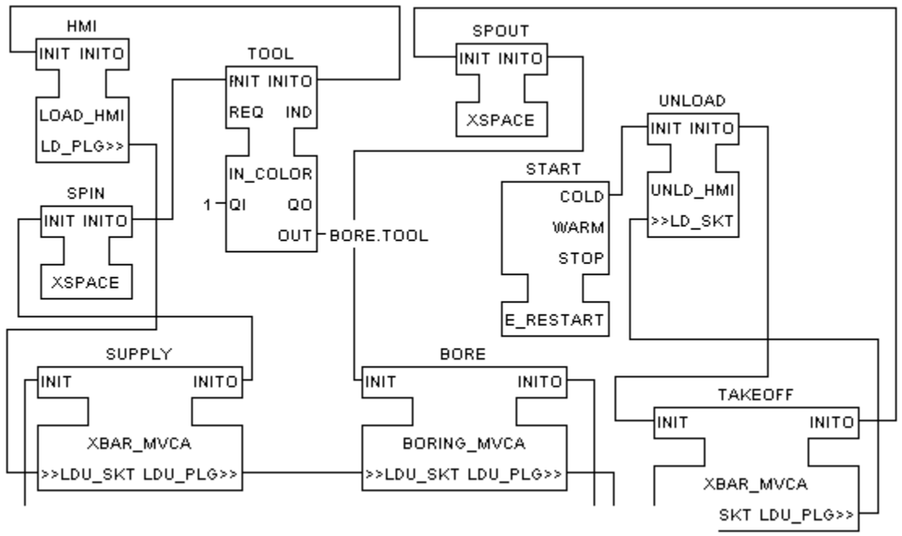
A *service interface function block* provides one or more *services* to an application, based on a *mapping* of *service primitives* to the function block's *event inputs, event outputs, data inputs* and *data outputs*.

The external interfaces of *service interface function block types* have the same general appearance as *basic function block types*. However, some inputs and outputs of service interface function block types have specialized semantics, and the behavior of *instances* of these types is defined through a specialized graphical notation for sequences of *service primitives.*





图19说明了包含两个XBAR_MVCA类型实例的资源配置，如图18所示。SUPPLY实例充当HMI块的接受者（“下游单元”）和BORE块的提供者（“上游单元”），而TAKEOFF实例分别履行BORE和UNLOAD块的相应角色。




注1这个例子只是说明性的。规范的细节不是规范性的。

注2为清楚起见，此图中省略了参数连接。

注3附录F中没有给出XBAR_MVCA类型以外的块的类型声明。

**图19 适配器连接示意图**

#### 异常和故障处理

资源可以提供用于预防、识别和处理异常和故障的额外设施。这种能力可以建模为服务接口功能块。用于预防、识别和处理异常和故障的特定功能块类型的定义超出了本标准的范围。然而，服务接口功能块的INIT−、CNF和IND输出，以及相关的STATUS值，可用于指示异常和故障的发生和类型，如6.1.3中所述。

## 6 服务接口功能块

### 6.1 一般原则

服务接口功能块基于服务原语到功能块的事件输入、事件输出、数据输入和数据输出的映射向应用程序提供一个或多个服务。

服务接口功能块类型的外部接口与基本功能块类型具有相同的外观。但是，服务接口功能块类型的某些输入和输出具有专门的语义，并且这些类型的实例的行为是通过服务原语序列的专门图形符号定义的。

NOTE The specification of the internal operations of service interface function blocks is beyond the scope of this standard.

### 6.1.2 Type specification

*Declaration* of *service interface function block types* may use the standard *event inputs, event outputs, data inputs* and *data outputs* listed in Table 2, as appropriate to the particular service provided. When these are used, their semantics shall be as defined in 6.1.2. The name of the function block *type* shall indicate the provided service.

EXAMPLE Figure 20a and Figure 20b show examples of service interface function blocks in which the primary interaction is initiated by the application and by the resource, respectively.

NOTE 1 Services can provide both resource- and application-initiated interactions in the same service interface function block.

NOTE 2 Service interface types can also utilize inputs and outputs, including *plugs* and *sockets*, with names different from those given here; in such case their usage is defined in terms of appropriate sequences of service primitives.

**Table 2 – Standard inputs and outputs for service interface function blocks** (1 of 2)

| **Event inputs** |
|---|
| **INIT**<br>This event input shall be *mapped* to a *request primitive* which requests an initialization of the service provided by the function block instance, e.g., local initialization of a *communication connection* or a process interface module. |
| **REQ**<br>This event input shall be mapped to a *request primitive* of the service provided by the function block instance. |
| **RSP**<br>This event input shall be mapped to a *response primitive* of the service provided by the function block instance. |
| **Event outputs** |
| **INITO**<br>This event output shall be mapped to a *confirm primitive* which indicates completion of a service initialization procedure. |
| **CNF**<br>This event output shall be mapped to a *confirm primitive* of the service provided by the function block instance. |
| **IND**<br>This event output shall be mapped to an *indication primitive* of the service provided by the function block instance. |
| **Data inputs** |
| **QI**: **BOOL**<br>This input represents a qualifier on the *service primitives* mapped to the *event inputs*. For instance, if this input is TRUE upon the occurrence of an INIT event, initialization of the service is requested; if it is FALSE, termination of the service is requested. |
| **PARAMS**: **ANY**<br>This input contains one or more *parameters* associated with the service, typically as elements of an *instance* of a *structured data type.* When this input is present, the *function block type* specification shall define its *data type* and default initial value(s).<br>A service interface function block type specification may substitute one or more service parameter inputs for this input. |
| **SD_1, ..., SD_m: ANY**<br>These inputs contain the data associated with *request* and *response primitives*. The *function block type* specification shall define the *data types* and default values of these inputs, and shall define their associations with event inputs in an event sequence diagram as illustrated in 6.1.3.<br>The function block type specification may define other names for these inputs. |



注：服务接口功能块的内部操作规范超出了本标准的范围。

### 型号规格

服务接口功能块类型的声明可以使用表2中列出的标准事件输入、事件输出、数据输入和数据输出，以适用于所提供的特定服务。当使用这些时，它们的语义应如6.1.2中所定义。功能块类型的名称应表明所提供的服务。

示例图20a和图20b显示了服务接口功能块的示例，其中主要交互分别由应用程序和资源启动。

注1服务可以在同一个服务接口功能块中提供资源和应用程序发起的交互。

注2服务接口类型也可以使用输入和输出，包括插头和插座，其名称与此处给出的名称不同；在这种情况下，它们的使用是根据适当的服务原语序列来定义的。

表2 服务接口功能块的标准输入和输出（2个中的1个）

| **事件输入** |
|---|
| 此事件输入应映射到请求初始化功能块实例提供的服务的请求原语，例如，通信连接或过程接口模块的本地初始化。 |
| 该事件输入应映射到功能块实例提供的服务的请求原语。 |
| 该事件输入应映射到功能块实例提供的服务的响应原语。 |
| **事件输出** |
| 该事件输出应映射到一个确认原语，该原语指示服务初始化过程的完成。 |
| 该事件输出应映射到功能块实例提供的服务的确认原语。 |
| 该事件输出应映射到功能块实例提供的服务的指示原语。 |
| **数据输入** |
| 此输入表示映射到事件输入的服务原语的限定符。例如，如果在发生INIT事件时此输入为TRUE，则请求初始化服务；如果为FALSE，则请求终止服务。 |
| 此输入包含一个或多个与服务关联的参数，通常作为结构化数据类型实例的元素。当此输入出现时，功能块类型规范应定义其数据类型和默认初始值。<br>服务接口功能块类型规范可以用一个或多个服务参数输入代替该输入。 |
| 这些输入包含与请求和响应原语相关的数据。功能块类型规范应定义这些输入的数据类型和默认值，并应在事件序列图中定义它们与事件输入的关联，如6.1.3所示。<br>功能块类型规范可以为这些输入定义其他名称。 |

**Table 2** (2 of 2)

| Data outputs |
|---|
| **QO: BOOL**<br>This variable represents a qualifier on the *service primitives* mapped to the *event outputs*. For instance, a TRUE value of this output upon the occurrence of an INITO event indicates successful initialization of the service; a FALSE value indicates unsuccessful initialization. |
| **STATUS: ANY**<br>This output shall be of a *data type* appropriate to express the status of the service upon the occurrence of an event output.<br>A service specification may indicate that the value of this output is irrelevant for some situations,for instance, for INITO+, IND+ and CNF+ as described in 6.1.3. |
| **RD_1, ..., RD_n: ANY**<br>These outputs contain the data associated with *confirm* and *indication primitives*. The function block *type specification* shall define the *data types* and initial values of these outputs, and shall define their associations with event outputs in an event sequence diagram as described in 6.1.3.<br>The function block type specification may define other names for these outputs. |

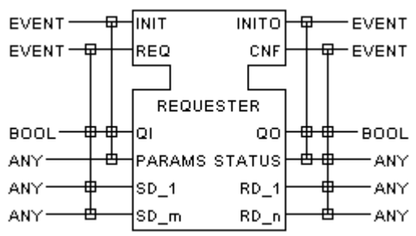



**Figure 20a – Application-initiated interactions**

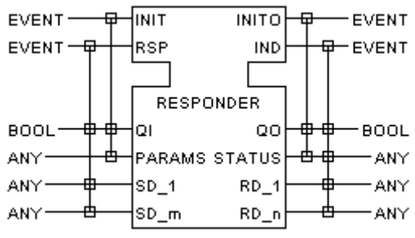



**Figure 20b – Resource-initiated interactions**

NOTE 1 REQUESTER and RESPONDER represent the particular services provided by instances of the function block types.

NOTE 2 The *data types* of the SD_1,...,SD_n inputs and RD_1,...RD_m outputs will typically be fixed as some non-generic data type, for instance INT or WORD, in concrete implementations of the generic function block types illustrated here.

NOTE 3 See Annex F for a full textual declaration of the REQUESTER function block type.

**Figure 20 – Example service interface function blocks**

### 6.1.3 Behavior of instances

The behavior of *instances* of *service interface function blocks* shall be defined in the corresponding *function block type* specification, which can utilize *service sequence diagrams* subject to the following rules:

a) The following semantics shall apply:
   1) Time increases in the downward direction.
   2) Events which are sequentially related are linked together across or within resources.
   3) If there is no specific relationship between events, in that it is impossible to foresee which will occur first but both shall occur within a finite period of time, a tilde (~) or similar textual notation is used.





表2（2个中的2个）

| 数据输出 |
|---|
| 此变量表示映射到事件输出的服务原语的限定符。例如，在发生INITO事件时此输出的TRUE值表示服务初始化成功；FALSE值表示初始化不成功。 |
| 该输出的数据类型应适合于在事件输出发生时表达服务的状态。<br>服务规范可能表明此输出的值与某些情况无关，例如，对于INITO+、IND+和CNF+，如6.1.3中所述。 |
| 这些输出包含与确认和指示原语相关的数据。功能块类型规范应定义这些输出的数据类型和初始值，并应在事件序列图中定义它们与事件输出的关联，如6.1.3所述。<br>功能块类型规范可以为这些输出定义其他名称。 |

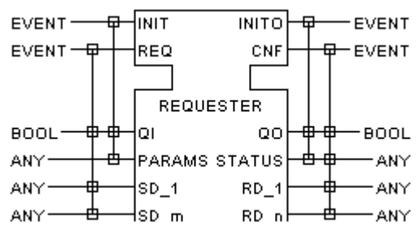



**图20a 应用程序发起的交互**

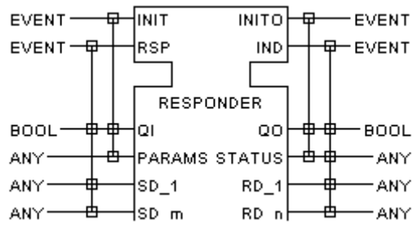



**图20b 资源发起的交互**

注1REQUESTER和RESPONDER代表功能块类型实例提供的特定服务。

注2在通用功能块类型的具体实现中，SD_1 … SD_n输入和RD_1 …RD_m输出的数据类型通常会固定为一些非通用数据类型，例如INT或WORD此处说明。

注3REQUESTER功能块类型的全文声明见附录F。

**图20 示例服务接口功能块**

#### 实例的行为

服务接口功能块实例的行为应在相应的功能块类型规范中定义，它可以使用服务序列图，但须遵守以下规则：

a)应适用以下语义：
   1)时间向下增加。
   2)顺序相关的事件跨资源或在资源内链接在一起。
   3)如果事件之间没有特定关系，即无法预见哪个会先发生，但两者都将在有限的时间内发生，则使用波浪号(~)或类似的文本符号。

b) In the case where the service is represented by a single service interface function block, the diagram shall be partitioned by a single vertical line into two fields as illustrated in Figure 21:

   1) In the case where the service is provided primarily by an application-initiated interaction, the *application* shall be in the left-hand field and the *resource* in the right-hand field, as illustrated in Figure 21a.

   2) In the case where the service is provided primarily by a resource-initiated interaction, the *resource* shall be in the left-hand field and the *application* in the right-hand field, as illustrated in Figure 21b.

c) In the case where the service is represented by two or more service interface function blocks, the notation illustrated in E.2.2 and E.2.3 can be used.

d) *Service primitives* shall be indicated by horizontal arrows. The name of the *event* representing the service primitive shall be written adjacent to the arrow, and means shall be provided to determine the names of the input and/or output *variables* representing the *data* associated with the primitive.

e) When a QI input is present in the function block type definition, the suffix "+" shall be used in conjunction with an *event input* name to indicate that the value of the QI input is TRUE upon the occurrence of the associated event, and the suffix "-" shall be used to indicate that it is FALSE.

f) When a QO output is present in the function block type definition, the suffix "+" shall be used in conjunction with an *event output* name to indicate that the value of the QO output is TRUE upon the occurrence of the associated event, and the suffix "-" shall be used to indicate that it is FALSE.

g) The standard semantics of asserted (+) and negated (-) events shall be as specified in Table 3.

Figure 21 illustrates normal sequences of service initiation, data transfer, and service termination. *Service interface function block type* specifications can utilize similar diagrams to specify all relevant sequences of service primitives and their associated data under both normal and abnormal conditions.

NOTE Sequence diagrams can also be used to document the externally observable behaviors of basic and composite function block types.

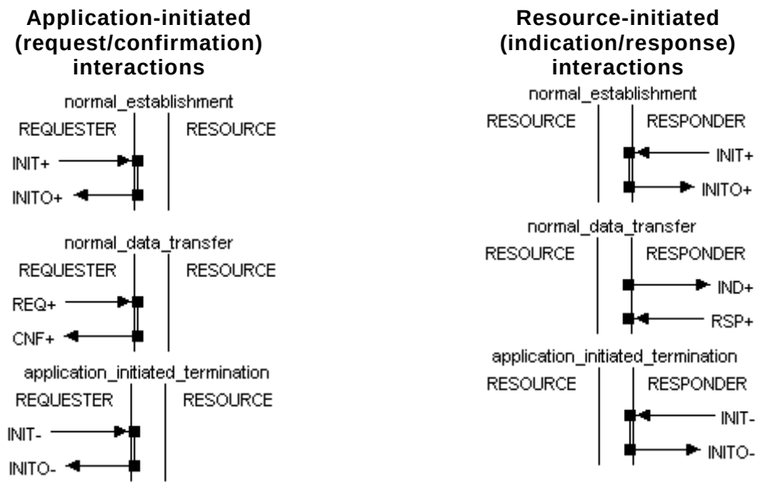



**Figure 21 – Example service sequence diagrams**

b)在服务由单个服务接口功能块表示的情况下，该图应由一条垂直线划分为两个字段，如图21所示：

1)在服务主要由应用程序发起的交互提供的情况下，应用程序应位于左侧字段中，资源应位于右侧字段中，如图21a所示。

2)在服务主要由资源发起的交互提供的情况下，资源应在左侧字段中，应用程序在右侧字段中，如图21b所示。

c)在服务由两个或多个服务接口功能块表示的情况下，可以使用E.2.2和E.2.3中说明的符号。

d)服务原语应由水平箭头表示。代表服务原语的事件名称应写在箭头旁边，并且应提供方法来确定表示与原语相关的数据的输入和/或输出变量的名称。

e)当功能块类型定义中存在QI输入时，后缀“+”应与事件输入名称一起使用，以指示在相关事件发生时QI输入的值为TRUE，并且应使用后缀“-”来表示它是FALSE。

f) 当功能块类型定义中存在QO输出时，后缀“+”应与事件输出名称一起使用，以指示在相关事件发生时QO输出的值为TRUE，并且后缀“-”用于表示为FALSE。

g)断言(+)和否定(-)事件的标准语义应在表3中指定。

图21说明了服务启动、数据传输和服务终止的正常顺序。服务接口功能块类型规范可以利用类似的图表来指定服务原语的所有相关序列及其在正常和异常条件下的相关数据。

注意序列图还可用于记录基本和复合功能块类型的外部可观察行为。

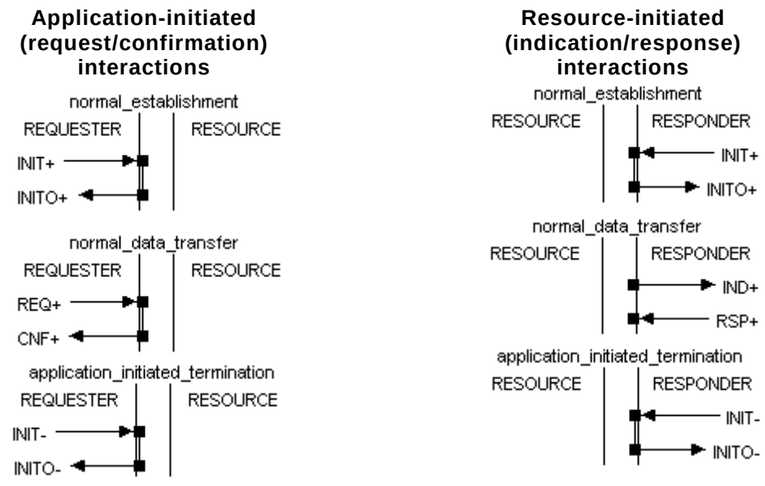



**图21 示例服务序列图**

**Table 3 – Service primitive semantics**

| Primitive | Semantics |
|---|---|
| **INIT+** | Request for service establishment |
| **INIT-** | Request for service termination |
| **INITO+** | Indication of establishment of normal service |
| **INITO-** | Rejection of service establishment request or indication of service termination |
| **REQ+** | Normal request for service |
| **REQ-** | Disabled request for service |
| **CNF+** | Normal confirmation of service |
| **CNF-** | Indication of abnormal service condition |
| **IND+** | Indication of normal service arrival |
| **IND-** | Indication of abnormal service condition |
| **RSP+** | Normal response by application |
| **RSP-** | Abnormal response by application |

## 6.2 Communication function blocks

### 6.2.1 Type specification

Communication function blocks provide *interfaces* between *applications* and the "communication mapping" functions of *resources* as defined in 4.3; hence, they are *service interface function blocks* as described in 6.1.

Like other service interface function blocks, a communication function block may be of either *basic* or *composite* type, as long its operation can be represented by a *mapping* of *service primitives* to the function block's *event inputs, event outputs, data inputs* and *data outputs*.

This subclause provides rules for the *declaration* of *communication function block types.* 6.2.2 provides rules for the behavior of *instances* of such function block types. Clause E.2 defines generic communication function block types for *unidirectional* and *bidirectional transactions,* and gives rules for the implementation-dependent customization of these types.

*Declaration* of *communication function block types* shall utilize the means defined in 6.1 for the declaration of *service interface function block types*, with the specialized semantics shown in Table 4 for *input* and *output variables*.





**表3 服务原语语义**

| Primitive | antics |
|---|---|
| **INIT+** | 请求建立服务 |
| | 请求终止服务 |
| INITO+正常服务建立指示 | |
| INITOR拒绝服务建立请求或服务终止指示 | |
| **REQ+** | 正常的服务请求 |
| **REQ-** | 已禁用服务请求 |
| **CNF+** | 服务正常确认 |
| **CNF-** | 异常服务状态指示 |
| **IND+** | 正常服务到达的指示 |
| **IND-** | 异常服务状态指示 |
| **RSP+** | 应用程序的正常响应 |
| **RSP-** | 应用程序的异常响应 |

## 6.2 通讯功能块

### 型号规格

以及4.3中定义的资源的“通信映射”功能；因此，它们是6.1中描述的服务接口功能块。

与其他服务接口功能块一样，通信功能块可以是基本类型或复合类型，只要它的操作可以通过服务原语到功能块的事件输入、事件输出、数据输入和数据输出的映射来表示。

本子条款为通信功能块类型的声明提供了规则。6.2.2规定了此类功能块类型实例的行为规则。条款E.2定义了用于单向和双向事务的通用通信功能块类型，并为这些类型的实现相关定制提供了规则。

通信功能块类型的声明应使用6.1中定义的方法来声明服务接口功能块类型，输入和输出变量的特殊语义如表4所示。

**Table 4 – Variable semantics for communication function blocks**

| Variable | Semantics |
|---|---|
| **PARAMS** | This input provides *parameters* of the *communication connection* associated with the *communication function block instance*. This shall include means of identifying the communication protocol and communication connection, and may include other parameters of the communication connection such as timing constraints, etc. |
| **SD_1,..., SD_m** | These inputs represent *data* to be transferred along the *communication connection* specified by the PARAMS input upon the occurrence of a REQ+ or RSP+ *primitive*, as appropriate. [a] |
| **STATUS** | This output represents the status of the *communication connection*, for instance:<br>- Normal completion of initiation, termination, or data transfer<br>- Reasons for abnormal initiation, termination, or data transfer |
| **RD_1,..., RD_n** | These outputs represent *data* received along the *communication connection* specified by the PARAMS input upon the occurrence of an IND+ or CNF+ primitive, as appropriate. [a] |
| NOTE Communication function block type declarations can define constraints between RD_1,...,RD_n outputs and the SD_1,..., SD_m inputs of corresponding function block instances. For example. the number and types of the RD outputs might be constrained to match the number and types of the corresponding SD inputs. | |
| a *Communication function block type declarations* define the number and type of the SD_1,..., SD_m inputs and RD_1,...,RD_n outputs, and can assign them other names. | |

### 6.2.2 Behavior of instances

As illustrated in Clause E.2, the behavior of *instances* of *communication function block types* shall be defined in the corresponding communication function block type *declaration*, utilizing the means specified for *service interface function blocks* in 6.1 with the specialized service primitive semantics given in Table 5. Such specification shall include *service primitive* sequences for:

- normal and abnormal establishment and release of *communication connections;*
- normal and abnormal data transfer.

**Table 5 – Service primitive semantics for communication function blocks**

| Primitive | Semantics |
|---|---|
| **INIT+** | Request for communication connection establishment |
| **INIT-** | Request for communication connection release |
| **INITO+** | Indication of communication connection establishment |
| **INITO-** | Rejection of communication connection establishment request or indication of communication connection release |
| **REQ+** | Normal request for data transfer |
| **REQ-** | Disabled request for data transfer |
| **CNF+** | Normal confirmation of data transfer |
| **CNF-** | Indication of abnormal data transfer |
| **IND+** | Indication of normal data arrival |
| **IND-** | Indication of abnormal data arrival |
| **RSP+** | Normal response by application to data arrival |
| **RSP-** | Abnormal response by application to data arrival |





**表4 通信功能块的变量语义**

| Variable | |
|---|---|
| **PARAMS** | 该输入提供与通信功能块实例相关的通信连接参数。这应包括识别通信协议和通信连接的方式，并且可能包括通信连接的其他参数，例如时序约束等。 |
| **SD_1,..., SD_m** | 这些输入表示在适当的REQ+或RSP+原语出现时要沿着由PARAMS输入指定的通信连接传输的数据。一个 |
| **STATUS** | 此输出表示通信连接的状态，例如：启动、终止或数据传输正常完成启动、终止或数据传输异常的原因 |
| **RD_1,..., RD_n** | 这些输出表示在出现IND+或CNF+原语时，在适当的情况下，沿着由PARAMS输入指定的通信连接接收的数据。一个 |
| 注意通信功能块类型声明可以定义RD_1 … RD_n输出和相应功能块实例的SD_1 … SD_m输入之间的约束。例如。RD输出的数量和类型可能会受到限制，以匹配相应SD输入的数量和类型。 | |
| a通信功能块类型声明定义了SD_1、…、SD_m输入和RD_1、…、RD_n输出的数量和类型，并且可以为它们分配其他名称。 | |

**实例的行为**

如第E.2节所示，通信功能块类型实例的行为应在相应的通信功能块类型声明中定义，使用6.1中为服务接口功能块指定的方法以及表5中给出的专用服务原语语义.此类规范应包括服务原语序列：

- 正常和异常建立和释放通信连接；
- 正常和异常数据传输。

**表5 通信功能块的服务原语语义**

| Primitive | |
|---|---|
| **INIT+** | 请求建立通信连接 |
| | 通信连接释放请求 |
| INITO+通信连接建立指示 | |
| INITOR拒绝通信连接建立请求<br>或通信连接释放指示 | |
| **REQ+** | 数据传输的正常请求 |
| **REQ-** | 数据传输请求被禁用 |
| **CNF+** | 数据传输的正常确认 |
| **CNF-** | 异常数据传输的指示 |
| **IND+** | 正常数据到达指示 |
| **IND-** | 异常数据到达指示 |
| **RSP+** | 应用程序对数据到达的正常响应 |
| **RSP-** | 应用程序对数据到达的异常响应 |

### 6.3 Management function blocks

#### 6.3.1 Requirements

Extending the functional requirements for "application management" in subclause 8.3.2 of ISO/IEC 7498-1:1994 to the distributed application model of this standard indicates that *services* for management of resources and applications in IPMCSs should be able to perform the following *functions:*

a) In a *resource,* create, initialize, start, stop, delete, query the existence and *attributes* of, and provide notification of changes in availability and status of:
   1) data types
   2) function block types and instances
   3) connections among function block instances
b) In a *device,* create, initialize, start, stop, delete, query the existence and *attributes* of, and provide notification of changes in availability and status of *resources*.

NOTE 1 The provisions of this standard are not intended to meet the requirements for *system management* addressed in ISO/IEC 7498-4 and ISO/IEC 10040, except as such requirements are addressed by the above listed functions.

NOTE 2 This standard only deals with item a) above, i.e., the management of *applications* in *resources*. A framework for device management is described in IEC 61499-2.

NOTE 3 The associations among *resources, applications,* and *function block instances* are defined in *system configurations* as described in 7.3.

NOTE 4 Starting and termination of a distributed *application* is performed by an appropriate *software tool*.

#### 6.3.2 Type specification

Figure 22 illustrates the general form of *management function block types* whose *instances* meet the application management requirements defined above.

NOTE 1 In particular implementations, the type name (MANAGER in this example) might represent the type of the managed resource.

NOTE 2 For these function block types, the specific CMD and OBJECT inputs and RESULT output replace the generic SD_1 and SD_2 inputs and RD_1 output described in 6.1.

NOTE 3 The INIT and PARAMS inputs and INITO output might or might not be present in a particular implementation.

NOTE 4 When present, the type and values of the PARAMS input are **implementation-dependent** parameters of the resource type.

NOTE 5 A full textual specification of this function block type, including all service sequences, is given in Annex F.

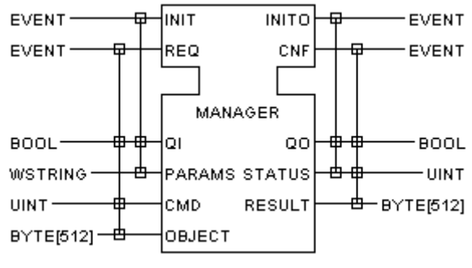



**Figure 22 – Generic management function block type**

The behavior of instances and input/output semantics of management function block types shall follow the rules given in 6.1 for *service interface function block types* with application-





### 6.3 管理功能块

将ISOIEC7498-1:1994的第8.3.2小节中“应用程序管理”的功能要求扩展到本标准的分布式应用程序模型，表明IPMCS中用于管理资源和应用程序的服务应该能够执行以下功能：

a)在一个资源中，创建、初始化、启动、停止、删除、查询其存在和属性，并提供可用性和状态变化的通知：
   1)数据类型
   2)功能块类型和实例
   3)功能块实例之间的连接
b)在设备中，创建、初始化、启动、停止、删除、查询资源的存在和属性，并提供资源可用性和状态变化的通知。

注1：本标准的规定并非旨在满足ISOIEC7498-4和ISOIEC10040中提出的系统管理要求，除非上述功能解决了这些要求。

注2本标准仅涉及上述a)项，即资源中应用程序的管理。IEC61499-2中描述了设备管理框架。

注3资源、应用程序和功能块实例之间的关联在系统配置中定义，如7.3所述。

注4：分布式应用程序的启动和终止由适当的软件工具执行。

#### 型号规格

图22说明了管理功能块类型的一般形式，其实例满足上面定义的应用程序管理要求。

注1在特定实现中，类型名称（本例中为MANAGER）可能表示受管资源的类型。

注2对于这些功能块类型，特定的CMD和OBJECT输入和RESULT输出替换了6.1中描述的通用SD_1和SD_2输入和RD_1输出。

注3：INIT和PARAMS输入以及INITO输出可能出现在特定实现中，也可能不出现。

**注4：当存在时，PARAMS输入的类型和值是资源类型的实现相关参数。**

注5该功能块类型的完整文本规范，包括所有服务序列，在
Annex F.

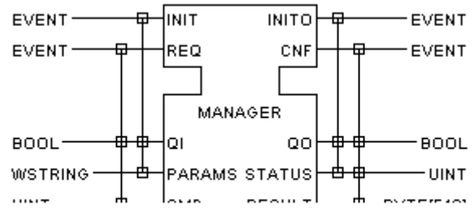
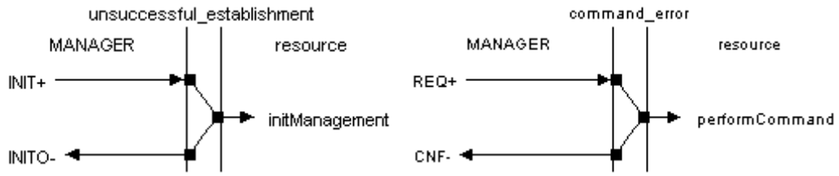



**图22 通用管理功能块类型**

管理功能块类型的实例行为和输入输出语义应遵循6.1中针对具有应用程序的服务接口功能块类型给出的规则。

initiated interactions, with the additional behaviors shown in Figure 23 for unsuccessful service initiation and requests.




NOTE A full textual specification of this function block type, including all service sequences, is given in Annex F.

**Figure 23 – Service primitive sequences for unsuccessful service**

The management *operation* to be *executed* shall be expressed by the value of the CMD input of a management function block according to the semantics defined in Table 6.

**Table 6 – CMD input values and semantics**

| Value | Command | Semantics |
|---|---|---|
| 0 | CREATE | Create specified object |
| 1 | DELETE | Delete specified object |
| 2 | START | Start specified object |
| 3 | STOP | Stop specified object |
| 4 | READ | Read parameter data |
| 5 | WRITE | Write parameter data |
| 6 | KILL | Make specified object unrunnable |
| 7 | QUERY | Request information on specified object |
| 8 | RESET | Reset specified object |

The values and corresponding semantics of the STATUS output of a management function block shall be as described in Table 7 to express the result of performing the specified command.

**Table 7 – STATUS output values and semantics**

| Value | Status | Semantics |
|---|---|---|
| 0 | RDY | No errors |
| 1 | BAD_PARAMS | Invalid PARAMS input value |
| 2 | LOCAL_TERMINATION | Application-initiated termination |
| 3 | SYSTEM_TERMINATION | System-initiated termination |
| 4 | NOT_READY | Manager is not able to process the command |
| 5 | UNSUPPORTED_CMD | Requested command is not supported |
| 6 | UNSUPPORTED_TYPE | Requested object type is not supported |
| 7 | NO_SUCH_OBJECT | Referenced object does not exist |
| 8 | INVALID_OBJECT | Invalid object specification syntax |
| 9 | INVALID_OPERATION | Commanded operation is invalid for specified object |
| 10 | INVALID_STATE | Commanded operation is invalid for current object state |
| 11 | OVERFLOW | Previous transaction still pending |



发起的交互，以及图23中针对不成功的服务发起和请求显示的附加行为。


注：该功能块类型的全文规范，包括所有服务序列，在附录F中给出。

**图23 不成功服务的服务原语序列**

根据表6中定义的语义，要执行的管理操作应由管理功能块的CMD输入值表示。

**表6 CMD输入值和语义**

| Value | Command | emantics |
|---|---|---|
| 0 | CREATE | 创建指定对象 |
| 1 | DELETE | 删除指定对象 bject |
| 2 | START | 启动指定对象 bject |
| 3 | STOP | 停止指定对象 ject |
| 4 | READ | 读取参数数据 ta |
| 5 | WRITE | 写入参数数据 |
| 6 | KILL | 使指定的对象不可运行 e |
| 7 | QUERY | 请求指定对象的信息 ɔd object |
| 8 | RESET | 重置指定对象 ɔject |

管理功能块的STATUS输出的值和对应的语义应如表7所述，以表达执行指定命令的结果。

**表7 STATUS输出值和语义**

| Value | Status | Semantics |
|---|---|---|
| 0 | RDY | 没有错误 |
| 1 | BAD_PARAMS | 无效的参数输入值 'alue |
| 2 | LOCAL_TERMINATION | |
| 3 | SYSTEM_TERMINATION | |
| 4 | NOT_READY | 经理无法处理命令 command |
| 5 | UNSUPPORTED_CMD | 不支持请求的命令 rted |
| 6 | UNSUPPORTED_TYPE | 不支持请求的对象类型 upported |
| 7 | NO_SUCH_OBJECT | 引用的对象不存在 |
| 8 | INVALID_OBJECT | 无效的对象规范语法 |
| 9 | INVALID_OPERATION | 命令操作对指定对象无效 |
| 10 | INVALID_STATE | 命令操作对当前对象状态无效 |
| 11 | OVERFLOW | 之前的交易仍在等待中 |

The actual lengths of the OBJECT input and RESULT output of management function block instances are **implementation-dependent**.

The OBJECT input shall specify the object to be operated on according to the CMD input, and the RESULT output shall contain a description of the object resulting from the operation if successful. The contents of these strings shall consist of **implementation-dependent** encodings of objects defined as non-terminal symbols in Annex B and referenced in Table 8.

NOTE 6 The maximum allowable length of the OBJECT input and RESULT output is an **implementation-dependent parameter**; the value of 512 given in Figure 22 is illustrative.

**Table 8 – Command syntax**

| CMD | OBJECT | RESULT |
|---|---|---|
| **CREATE** | type_declaration | data_type_name |
| | fb_type_declaration | fb_type_name |
| | fb_instance_definition | fb_instance_reference |
| | connection_definition | connection_start_point |
| **DELETE** | data_type_name | data_type_name |
| | fb_type_name | fb_type_name |
| | fb_instance_reference | fb_instance_reference |
| | connection_definition | connection_definition |
| **START** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **STOP** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **KILL** | fb_instance_reference | fb_instance_reference |
| **QUERY** | all_data_types | data_type_list |
| | all_fb_types | fb_type_list |
| | data_type_name | type_declaration |
| | fb_type_name | fb_type_declaration |
| | fb_instance_reference | fb_status |
| | connection_start_point | connection_end_points |
| | application_name | fb_instance_list |
| **READ** | parameter_reference | parameter |
| **WRITE** | referenced_parameter | parameter_reference |
| **RESET** | fb_instance_reference | fb_status |

NOTE See Table 6 for the integer values of the CMD input corresponding to the commands listed above.

It shall be an **error**, resulting in a STATUS code of INVALID_OBJECT, if a CREATE command attempts to create

- a *function block* whose *instance name* duplicates that of an existing function block within the same *resource*,
- a duplicate *connection*, or
- multiple connections to a *data input.*

The single exception to the above rule is that a CREATE command can replace a connection of a *parameter* to a *data input* with a new parameter connection.



管理功能块实例的OBJECT输入和RESULT输出的实际长度取决于实现。

OBJECT输入应根据CMD输入指定要操作的对象，如果成功，RESULT输出应包含对操作产生的对象的描述。这些字符串的内容应由附件B中定义为非终结符号并在表8中引用的对象的实现相关编码组成。

**注6：OBJECT输入和RESULT输出的最大允许长度是一个与实现相关的参数；图22中给出的512的值是说明性的。**

**表8 命令语法**

| CMD | OBJECT | RESULT |
|---|---|---|
| **CREATE** | type_declaration | data_type_name |
| | fb_type_declaration | fb_type_name |
| | fb_instance_definition | fb_instance_reference |
| | connection_definition | connection_start_point |
| **DELETE** | data_type_name | data_type_name |
| | fb_type_name | fb_type_name |
| | fb_instance_reference | fb_instance_reference |
| | connection_definition | connection_definition |
| **START** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **STOP** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **KILL** | fb_instance_reference | fb_instance_reference |
| **QUERY** | all_data_types | data_type_list |
| | all_fb_types | fb_type_list |
| | data_type_name | type_declaration |
| | fb_type_name | fb_type_declaration |
| | fb_instance_reference | fb_status |
| | connection_start_point | connection_end_points |
| | application_name | fb_instance_list |
| **READ** | parameter_reference | parameter |
| **WRITE** | referenced_parameter | parameter_reference |
| | | |

注意与上面列出的命令相对应的CMD输入的整数值参见表6。

如果CREATE命令尝试创建

- 一个功能块，其实例名称与同一资源中现有功能块的名称重复，
- 重复连接，或
  到数据输入的多个连接。

上述规则的唯一例外是CREATE命令可以用新的参数连接替换参数到数据输入的连接。

It shall be an **error**, resulting in a STATUS code of UNSUPPORTED_TYPE, if a CREATE command attempts to create a function block instance or parameter of a *type* which is not known to the management function block.

It shall be an **error**, resulting in a STATUS code of INVALID_OPERATION, if a DELETE command attempts to delete a *function block type*, function block instance, *data type* or connection which is defined in the *type specification* of the managed *resource*.

The semantics of the START and STOP commands shall be as follows:

- START and STOP of a *function block instance* shall be as defined in 6.3.2;
- START and STOP of an *application* shall be equivalent to START and STOP, respectively, of all *function block instances* in the application contained within the managed *resource*;
- STOP of a *management function block instance* shall be equivalent to STOP of all *function block instances* within the managed *resource*;
- START of a *management function block instance* shall be equivalent to START of all *function block instances* within the managed *resource.* If the managed resource was previously stopped, this shall be followed by issuing of an event at the appropriate output of each instance of the E_RESTART function block type defined in Annex A. These events shall occur at the WARM outputs of the E_RESTART blocks if the resource was stopped due to a previous STOP command, and at the COLD outputs otherwise.

Specialized semantics for the QUERY command shall be as follows:

- when the OBJECT input specifies an *event input, event output* or *data output,* the RESULT output shall contain zero or more opposite end points;
- when the OBJECT input specifies a *data input,* the RESULT output shall list zero or one opposite end point;
- when the OBJECT input specifies the name of an *application,* the RESULT output shall list the names of all function blocks in the application contained within the managed *resource*.

### 6.3.3 Behavior of managed function blocks

Function blocks that are under the control of a *management function block* shall exhibit operational behaviors equivalent to that shown in the state transition diagram of Figure 24, subject to the following rules.

a) The capitalized transition conditions in Figure 24 refer to a value of the CMD input, as specified in Table 6, of the management function block upon the occurrence of a REQ+ service primitive.

b) The command_error sequence of primitives for the MANAGER function block type shall occur, with the indicated value of the STATUS output as defined in Table 7, under the following conditions:

   1) UNSUPPORTED_CMD: No state exists in Figure 24 with a transition condition for the specified CMD value;

   2) INVALID_STATE: The currently active state does not have a transition condition for the specified CMD value;

   3) UNSUPPORTED_TYPE: The CMD value is CREATE, and the function block instance does not exist, but the function block type is unknown to the MANAGER instance, i.e., the guard condition **type_defined** is FALSE;

   4) INVALID_OPERATION: The CMD value is DELETE, and the function block instance is in the STOPPED or KILLED state, but the function block instance is *declared* in the *device* or *resource type* specification, i.e., the guard condition **is_deletable** is FALSE.



如果CREATE命令试图创建管理功能块不知道的类型的功能块实例或参数，这将是一个错误，导致状态代码为UNSUPPORTED_TYPE。

如果DELETE命令试图删除托管资源的类型规范中定义的功能块类型、功能块实例、数据类型或连接，则应为错误，导致状态代码为INVALID_OPERATION。

START和STOP命令的语义如下：

- 功能块实例的START和STOP应如6.3.2中所定义；
- 应用程序的START和STOP应分别等效于托管资源中包含的应用程序中所有功能块实例的START和STOP；
- 一个管理功能块实例的停止应等同于被管理资源内所有功能块实例的停止；
- 管理功能块实例的START应等同于被管理资源内所有功能块实例的START。如果受管资源先前已停止，则应在附件A中定义的每个E_RESTART功能块类型实例的适当输出处发出事件。如果资源由于先前的STOP命令而停止，则这些事件应在E_RESTART块的WARM输出处发生，否则在COLD输出处发生。

QUERY命令的专用语义如下：

- 当OBJECT输入指定事件输入、事件输出或数据输出时，RESULT输出应包含零个或多个相反的端点；
- 当OBJECT输入指定数据输入时，RESULT输出应列出零个或一个相反的端点；
- 当OBJECT输入指定应用程序的名称时，RESULT输出应列出托管资源中包含的应用程序中所有功能块的名称。

**托管功能块的行为**

受管理功能块控制的功能块应表现出与图24状态转换图中所示的操作行为等效的操作行为，但须遵守以下规则。

a)图24中大写的转换条件是指在REQ+服务原语出现时管理功能块的CMD输入值，如表6中指定的那样。

b)MANAGER功能块类型的原语command_error序列应在以下条件下发生，并具有表7中定义的STATUS输出的指示值：

1)UNSUPPORTED_CMD：图24中不存在具有指定CMD值的转换条件的状态；

2)INVALID_STATE：当前活动状态没有指定CMD值的转换条件；

3)UNSUPPORTED_TYPE：CMD值为CREATE，功能块实例不存在，但功能块类型对MANAGER实例未知，即保护条件type_defined为FALSE；

4)INVALID_OPERATION：CMD值为DELETE，功能块实例处于STOPPED或KILLED状态，但功能块实例在设备或资源类型规范中声明，即保护条件is_deletable为FALSE。

c) The normal_command_sequence of primitives shown for the MANAGER function block type shall follow a CMD+ service primitive under all other conditions, with a value of RDY for the STATUS output as defined in Table 7, and a corresponding value for the RESULT output as defined in Table 8.

d) The semantics of the actions shown in Figure 24 shall be as shown in Table 9 for managed *basic* and *service interface function blocks*.

e) The actions described in the previous rule apply recursively to all *component function blocks* of managed *composite function blocks*.

NOTE 1 The behaviors of function blocks that are not under the control of management function blocks are beyond the scope of this standard.

NOTE 2 Specification of the behavior of managed function blocks under conditions of power loss and restoration is beyond the scope of this standard. Such behavior can be specified by the manufacturer of a compliant device, for example by reference to an appropriate standard.

NOTE 3 *Applications* can utilize *instances* of the E_RESTART block described in Annex A to generate events that can be used to trigger appropriate algorithms upon power loss and restoration.

NOTE 4 As described in 5.4.2, execution control in *subapplications* is entirely deferred to the execution control mechanisms of their component function blocks and component subapplications.

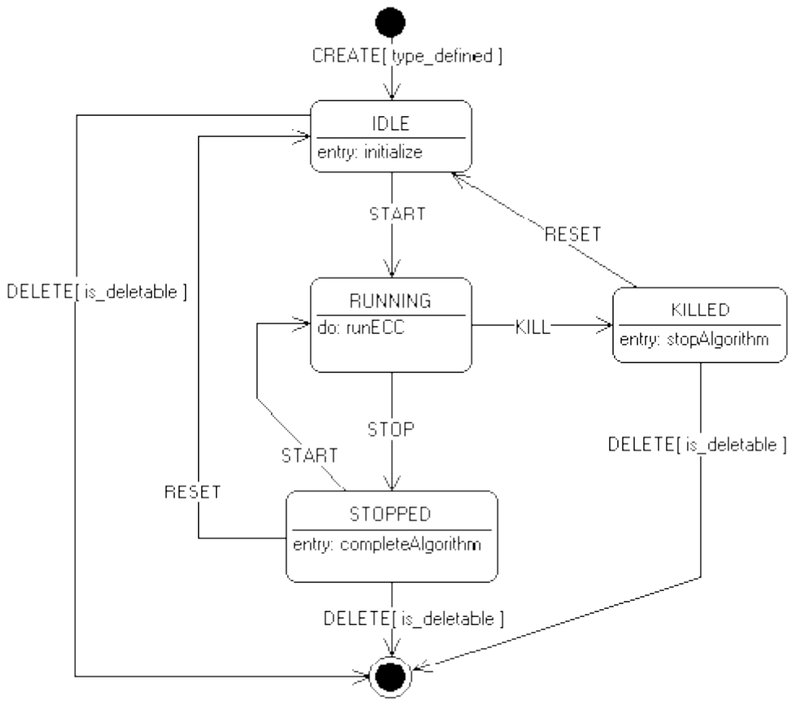



**Figure 24 – Operational state machine of a managed function block**



c)在所有其他条件下，为MANAGER功能块类型显示的原语的normal_command_sequence应遵循CMD+服务原语，表7中定义的STATUS输出值为RDY，而RESULT输出的值如表7中定义。表8。

d)图24所示动作的语义应如表9所示，用于管理的基本和服务接口功能块。

e)前面规则中描述的动作递归地应用于托管复合功能块的所有组件功能块。

注1：不受管理功能块控制的功能块的行为超出了本标准的范围。

注2：管理功能块在断电和恢复条件下的行为规范超出了本标准的范围。这种行为可以由兼容设备的制造商指定，例如通过参考适当的标准。

注3应用程序可以利用附件A中描述的E_RESTART块的实例来生成可用于在断电和恢复时触发适当算法的事件。

注4如5.4.2所述，子应用程序中的执行控制完全依赖于其组件功能块和组件子应用程序的执行控制机制。

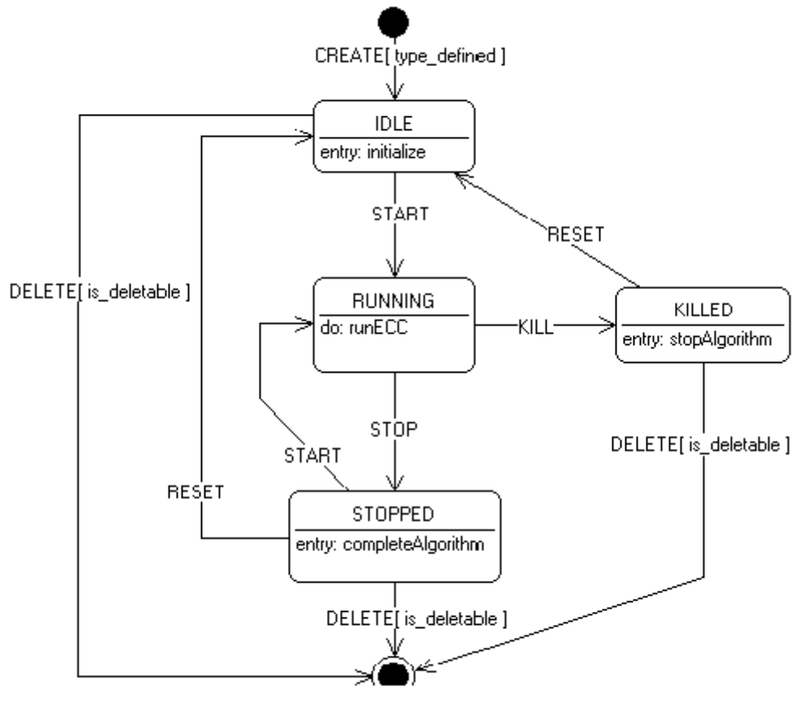



**图24 托管功能块的操作状态机**

**Table 9 – Semantics of actions in Figure 24**

| Action | Basic function blocks | Service interface function block |
|---|---|---|
| **initialize** | Initialize all variables as defined in 5.2.2.1. | |
| | Perform other initialization operations as defined in 5.2.2.1. | Place service in the proper state to respond correctly to an INIT+ primitive. |
| **runECC** | Enable operation of the ECC state machine defined in 5.2.2.2. | Enable invocation of service primitives by events at event inputs, and generation of events at event outputs. |
| **completeAlgorithm** | Allow the currently active algorithm (if any) without further generation of output events. | Allow the currently active service primitive to complete. |
| **stopAlgorithm** | Terminate the operations of the currently active algorithm (if any) immediately. | Terminate all operations of the service immediately. |

# 7 Configuration of functional units and systems

## 7.1 Principles of configuration

Clause 7 contains rules for the *configuration* of industrial-process measurement and control *systems* (IPMCSs) according to the following model:

a) an IPMCS consists of interconnected *devices;*

b) a *device* is an *instance* of a corresponding device *type;*

c) the functional capabilities of a *device type* are described in terms of its associated *resources;*

d) a *resource* is an *instance* of a corresponding resource *type*;

e) the functional capabilities of a *resource type* are described in terms of the *function block types* which can be *instantiated,* and the particular *function block instances* which exist, in all *instances* of the resource *type.*

The *configuration* of an IPMCS is thus considered to consist of the *configuration* of its associated *devices* and *applications,* including the allocation of *function block instances* in each *application* to the *resources* associated with the *devices*. Clause 7 defines the following sets of rules to support this process:

- rules for the functional specification of *types* of *resources* and *devices* are defined in 7.2;
- rules for the *configuration* of an IPMCS in terms of its associated *devices* and *applications* are defined in 7.3.

## 7.2 Functional specification of resource, device and segment types

### 7.2.1 Functional specification of resource types

The functional specification of a *resource type* includes:

- the resource *type name*;
- the *instance name, data type*, and initialization of each of the resource *parameters*;
- a declaration of the *data types* and *function block types* that each *instance* of the *resource type* is capable of *instantiating*;
- the instance names, types, and initial values of any function block instances that are always present in each instance of the resource type;
- any *data connections, adapter connections* and *event connections* that are always present in each instance of the resource type.





**表9 图24中动作的语义**

| Action | 基本功能块 | 服务接口功能块 |
|---|---|---|
| **initialize** | 初始化5.2.2.1中定义的所有变量。 | |
| | 执行5.2.2.1中定义的其他初始化操作。 | 将服务置于正确的状态以正确响应INIT+原语。 |
| **runECC** | 启用5.2.2.2中定义的ECC状态机的操作。 | 在事件输入处启用事件调用服务原语，并在事件输出处生成事件。 |
| completeAlgorithm允许当前活动的算法（如果有）而不进一步生成输出事件。 | | 允许当前活动的服务原语完成。 |
| **stopAlgorithm** | 立即终止当前活动算法（如果有）的操作。 | 立即终止服务的所有操作。 |

# 7 功能单元和系统的配置

**配置原则**

第7条包含根据以下模型配置工业过程测量和控制系统(IPMCS)的规则：

a)IPMCS由互连的设备组成；

b)设备是相应设备类型的实例；

c)设备类型的功能能力根据其相关资源进行描述；

d)资源是相应资源类型的实例；

e)一种资源类型的功能能力是根据可以被实例化的功能块类型和存在于该资源类型的所有实例中的特定功能块实例来描述的。

因此，IPMCS的配置被认为包括其关联设备和应用程序的配置，包括将每个应用程序中的功能块实例分配给与设备关联的资源。第7条定义了以下规则集以支持此过程：

- 资源和设备类型的功能规范规则在7.2中定义；
- 7.3中定义了根据其相关设备和应用程序配置IPMCS的规则。

## 7.2 资源、设备和段类型的功能规范

**资源类型的功能规范**

资源类型的功能规范包括：

- 资源类型名称；
- 每个资源参数的实例名称、数据类型和初始化；
- 资源类型的每个实例能够实例化的数据类型和功能块类型的声明；
- 始终存在于资源类型的每个实例中的任何功能块实例的实例名称、类型和初始值；
- 始终存在于资源类型的每个实例中的任何数据连接、适配器连接和事件连接。

NOTE 1 Additional information can be supplied with resource type specifications, including:

– the maximum numbers of *data connections, adapter connections* and *event connections* that can exist in an instance of the resource type;
– the time (identified as $T_{alg}$ in Figure 7) required for *execution* of each *algorithm* of function blocks of a specified type in an instance of the resource;
– the maximum number of instances of specified function block types that can exist in each instance of the resource;
– trade-offs among function block instances, e.g., whether two instances of function block type "A" can be traded for one instance of type "B", etc.

NOTE 2 The functional specifications of a resource's communication and process *interfaces*, including the kind and degree of compliance to applicable standards, is beyond the scope of this standard except as such interfaces are represented by *service interface function blocks.*

### 7.2.2 Functional specification of device types

The functional specification of a *device type* includes:

a) the device *type name*;
b) the *instance name, data type*, and initialization of each of the device *parameters*;
c) the instance name, type name, and initialization of each *function block instance* that is always present in each *instance* of the device type;
d) any *data connections, adapter connections* and *event connections* that are always present in each instance of the device type;
e) declarations of the *resource instances* which are present in each instance of the device type. Each such declaration shall contain:
   1) the resource instance name and type name;
   2) the instance name, type name, and initialization of each *function block instance* that is always present in the resource instance in each instance of the device type;
   3) any *data connections, adapter connections* and *event connections* that are always present in the resource instance in each instance of the device type.

NOTE 1 Items (2) and (3) above are considered to be in addition to the corresponding elements declared in the resource type specification as defined in 7.2.1.

NOTE 2 The functional specifications of a device's communication and process *interfaces*, including the kind and degree of compliance to applicable standards, is beyond the scope of this standard except as such interfaces are represented by *service interface function blocks*.

NOTE 3 A device type can contain a function block network only when it is considered to consist of a single (undeclared) resource; in such a case the device type does not contain any declarations of resource instances.

### 7.2.3 Functional specification of segment types

The functional specification of a *segment type* includes:

- the segment *type name*;
- the *instance name, data type*, and initialization of each of the segment *parameters*.

## 7.3 Configuration requirements

### 7.3.1 Configuration of systems

The configuration of a *system* includes:

- the *name* of the system;
- the specification of each *application* in the system, as specified in 7.3.2;
- the configuration of each *device* and its associated *resources*, as specified in 7.3.3;



注1可以随资源类型规范提供附加信息，包括：

– 资源类型实例中可以存在的最大数据连接数、适配器连接数和事件连接数；
– 执行一个功能块的每个算法所需的时间（在图7中标识为Talg）
资源实例中的指定类型；
– 资源的每个实例中可以存在的指定功能块类型的最大实例数；

功能块实例之间的权衡，例如，功能块类型“A”的两个实例是否可以交换一个“B”类型的实例等。

注2资源的通信和过程接口的功能规范，包括符合适用标准的种类和程度，超出了本标准的范围，除非这些接口由服务接口功能块表示。

#### 设备类型的功能规范

设备类型的功能规范包括：

a)设备类型名称；
b)每个设备参数的实例名称、数据类型和初始化；
c)实例名称、类型名称和每个功能块实例的初始化，这些实例始终存在于设备类型的每个实例中；

d)始终存在于设备类型的每个实例中的任何数据连接、适配器连接和事件连接；

e)存在于设备类型的每个实例中的资源实例的声明。每份此类声明应包含：

1）资源实例名和类型名；

2)设备类型的每个实例中的资源实例中始终存在的每个功能块实例的实例名称、类型名称和初始化；

3)任何数据连接、适配器连接和事件连接始终存在于设备类型的每个实例中的资源实例中。

注1上述(2)和(3)项被认为是对7.2.1中定义的资源类型规范中声明的相应元素的补充。

注2设备的通信和过程接口的功能规范，包括符合适用标准的种类和程度，超出了本标准的范围，除非这些接口由服务接口功能块表示。

注3：一个设备类型只能包含一个功能块网络，当它被认为是由一个单一的（未声明的）资源组成时；在这种情况下，设备类型不包含任何资源实例的声明。

#### 段类型的功能规范

段类型的功能规范包括：

- 段类型名称；
- 每个段参数的实例名称、数据类型和初始化。

## 7.3 配置要求

#### 系统配置

一个系统的配置包括：

- 系统名称；
- 系统中每个应用程序的规范，如7.3.2中所述；
- 7.3.3中规定的每个设备及其相关资源的配置；

- the configuration of each *network segment* and its associated *links* to devices or resources, as specified in 7.3.4.

### 7.3.2 Specification of applications

The specification of an *application* consists of:

- its name in the form of an *identifier*;
- the *instance name, type name, data connections, event connections* and *adapter connections* of each *function block* and *subapplication* in the application.

It shall be an **error** if the name of an application is not unique within the scope of the *system*.

### 7.3.3 Configuration of devices and resources

The configuration of a *device* consists of:

- the *instance name* and *type name* of the device;
- configuration-specific values for the device *parameters*;
- the *resource types* supported by the device *instance* in addition to those specified for the device *type*;
- the *instance name* and *type name* of each *function block instance* that is present in the device instance in addition to those defined for the device type;
- any *data connections, adapter connections* and *event connections* that are present in the device instance in addition to those defined for the device type;
- the *resource types* supported by the device *instance* in addition to those specified for the device *type*;
- the configuration of each of the *resources* in the device. These consist of any resource instances defined in the device *type* specification, plus any additional resources associated with the specific device *instance*.

NOTE A device instance can contain a function block network only when it is considered to consist of a single (undeclared) resource; in such a case the declaration of the device instance does not contain any declarations of resource instances.

It shall be an **error** if the instance name of each device is not unique within the scope of the system.

The configuration of a *resource* consists of:

a) its *instance name* and *type name*;

b) the *data types* and *function block types* supported by the resource *instance;*

c) the *instance name, type name*, and initialization of each function block instance that is present in the resource instance;

d) any *data connections, event connections* and *adapter connections* that are present in the resource instance.

Resource configuration is subject to the following rules:

– Items b), c), and d) above are considered to be in addition to the corresponding elements declared in the device and resource type specifications as defined in 7.2.2 and 7.2.1, respectively.

– Items c) and d) include *function block instances, data connections, adapter connections* and *event connections* from those portions of *applications* allocated to the resource.

– Items c) and d) include *communication function blocks, data connections, event connections* and *adapter connections* as necessary to establish and maintain the data and event flows for any associated *applications*.



- 每个网段的配置及其与设备或资源的关联链接，如7.3.4中所述。

#### 应用规范

应用程序的规范包括：

- 它的名称以标识符的形式；
- 应用程序中每个功能块和子应用程序的实例名称、类型名称、数据连接、事件连接和适配器连接。

**如果应用程序的名称在系统范围内不是唯一的，则为错误。**

#### 设备和资源的配置

设备的配置包括：

- 设备的实例名称和类型名称；
- 设备参数的配置特定值；
- 除了为设备类型指定的资源类型之外，设备实例支持的资源类型；
- 除了为设备类型定义的那些之外，设备实例中存在的每个功能块实例的实例名称和类型名称；
- 除了为设备类型定义的连接之外，设备实例中存在的任何数据连接、适配器连接和事件连接；
- 除了为设备类型指定的资源类型之外，设备实例支持的资源类型；
- 设备中每个资源的配置。这些包括在设备类型规范中定义的任何资源实例，以及与特定设备实例关联的任何附加资源。

注：一个设备实例只有在被认为由一个单一的（未声明的）资源组成时才能包含一个功能块网络；在这种情况下，设备实例的声明不包含任何资源实例的声明。

如果每个设备的实例名称在系统范围内不是唯一的，则应为错误。

资源的配置包括：

a)它的实例名称和类型名称；

b)资源实例支持的数据类型和功能块类型；

c)资源实例中存在的每个功能块实例的实例名称、类型名称和初始化；

d)资源实例中存在的任何数据连接、事件连接和适配器连接。

资源配置遵循以下规则：

– 上面的b)、c)和d)项被认为是对分别在7.2.2和7.2.1中定义的设备和资源类型规范中声明的相应元素的补充。

– 项c)和d)包括来自分配给资源的那些应用程序部分的功能块实例、数据连接、适配器连接和事件连接。

– 项c)和d)包括必要的通信功能块、数据连接、事件连接和适配器连接，以建立和维护任何相关应用程序的数据和事件流。

– The items in Item c) may include the *mapping* of function block instances in the application to function block instances existing in the resource as a result of type definition as described in 7.2.1.
– It shall be an **error** if the instance name of a resource is not unique within the scope of the device containing it, or if any function block instance in an application is not allocated to exactly one resource.

Automated means may be provided to meet the above requirements. Providers of such means shall either provide unambiguous rules by which their operation can be determined, or shall provide means by which the results of the application of such means can be examined and modified.

### 7.3.4 Configuration of network segments and links

The configuration of a *network segment* consists of:

- the *instance name* and *type name* of the segment;
- configuration-specific values for the *parameters* of the network segment.

It shall be an **error** if the *instance name* of each network segment is not unique within the scope of the *system*, or if the declared values of the segment parameters are inconsistent with the declaration (if any) of the *segment type* defined in 7.2.3.

The configuration of a *link* consists of:

– the name of a *device* or the hierarchical name of a "communication *resource*" inside a device, and the name of the network segment to which the device or the resource is connected;
– configuration-specific values for the *parameters* of the link.





– c)项中的项目可能包括应用程序中的功能块实例到资源中存在的功能块实例的映射，这是7.2.1中描述的类型定义的结果。
– 如果资源的实例名称在包含它的设备范围内不是唯一的，或者如果应用程序中的任何功能块实例没有分配给一个资源，则将是一个错误。

可以提供自动化装置来满足上述要求。此类手段的提供者应提供可以确定其操作的明确规则，或者应提供可以检查和修改此类手段的应用结果的手段。

### 网段和链路的配置

一个网段的配置包括：

- 段的实例名称和类型名称；
  网段参数的配置特定值。

如果每个网段的实例名称在系统范围内不是唯一的，或者段参数的声明值与7.2.3中定义的段类型的声明（如果有）不一致，则为错误.

链路的配置包括：

– 设备名称或设备内部"通信资源"的分层名称，以及该设备或资源所连接的网段名称；
– 链接参数的配置特定值。

# Annex A
(normative)

# Event function blocks

*Instances* of the function block *types* shown in Table A.1 can be used for the generation and processing of *events* in *composite function blocks*; in *subapplications*; in the definition of *resource* and *device types*; and in the *configuration* of *applications*, *resources* and *devices*.

Those function block types shown in Annex A which utilize *execution control charts* are *basic function block types*. Where textual declarations of *algorithms* are given for these function block types, the language used is the Structured Text (ST) language defined in IEC 61131-3.

Reference implementations for some of the function block types in Annex A are given as *composite function block type* definitions. These implementations are normative only in the sense that the functional behaviors of compliant implementations shall be equivalent to those of the reference implementation, where the following considerations apply to the timing parameters defined in 4.5.3.

- The parameters $T_{\text{setup}}$, $T_{\text{start}}$ and $T_{\text{finish}}$ are considered to be zero (0) for all *component function blocks* in the reference implementation.
- The parameter $T_{\text{alg}}$ is considered to be equal to the parameter DT for all instances of E_DELAY type used as *component function blocks* in the reference implementation, and to be zero (0) for all other component function blocks in the reference implementation.

All other function block types given in Annex A are *service interface function block types*.

NOTE Full textual specifications of all function block types shown in Table A.1 are given in Annex F.





# 事件功能块

表A.1所示功能块类型的实例可用于复合功能块中事件的生成和处理；在子应用程序中；在资源和设备类型的定义中；在应用程序、资源和设备的配置中。

附件A中所示的那些利用执行控制图的功能块类型是基本功能块类型。在为这些功能块类型给出算法的文本声明的情况下，使用的语言是IEC61131-3中定义的结构化文本(ST)语言。

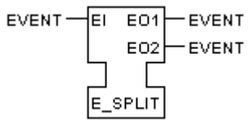
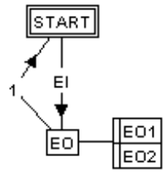
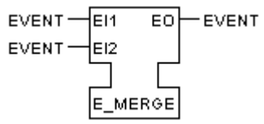
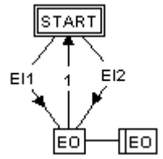
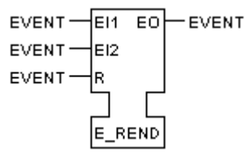
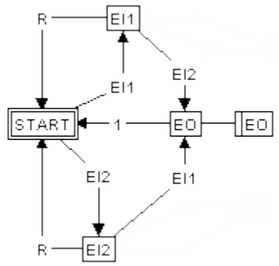
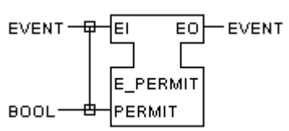
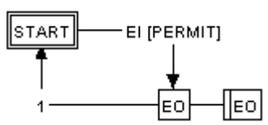
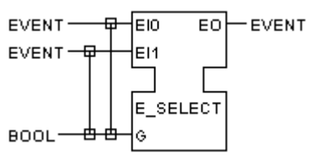
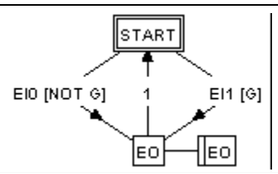
附件A中某些功能块类型的参考实现以复合功能块类型定义的形式给出。这些实现仅在兼容实现的功能行为应等同于参考实现的功能行为的意义上是规范的，其中以下考虑适用于4.5.3中定义的时序参数。

- 对于参考实现中的所有组件功能块，参数Tsetup、Tstart和Tfinish被认为为零(0)。
- 对于在参考实现中用作组件功能块的所有E_DELAY类型的实例，参数Talg被认为等于参数DT，而对于参考实现中的所有其他组件功能块，参数Talg被认为等于参数DT。

附件A中给出的所有其他功能块类型都是服务接口功能块类型。

注：附录F中给出了表A.1中所有功能块类型的全文规范。

**Table A.1 – Event function blocks** (1 of 6)

| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **1** | **Split an event** | |
| |  |  |
| The occurrence of an event at EI causes the occurrence of events at EO1, EO2,...,EOn (n=2 in the above example). | | |
| **2** | **Merge (OR) of multiple events** | |
| |  |  |
| The occurrence of an event at any of the inputs EI1, EI2,...,EIn causes the occurrence of an event at EO (n=2 in the above example). | | |
| **3** | **Rendezvous of two events** | |
| |  |  |
| **4** | **Permissive propagation of an event** | |
| |  |  |
| **5** | **Selection between two events** | |
| |  |  |





表A.1 事件功能块（6个中的1个）

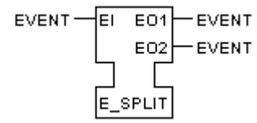
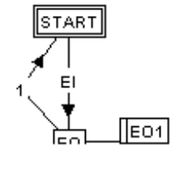
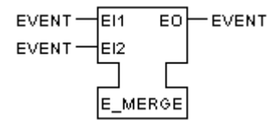
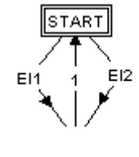
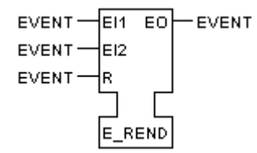
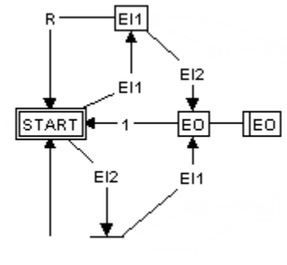
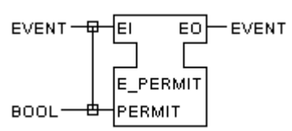
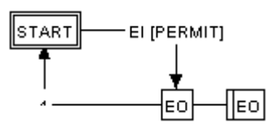
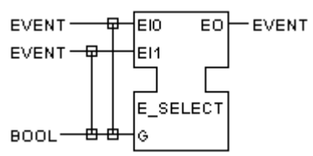
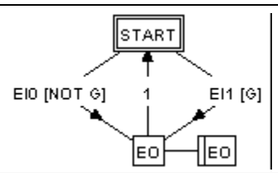
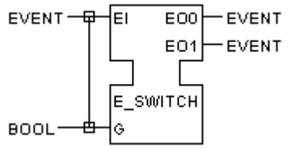
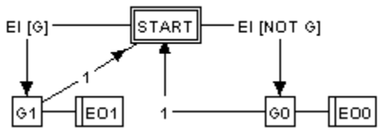
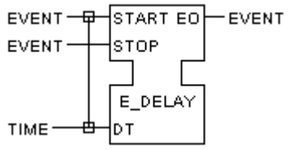
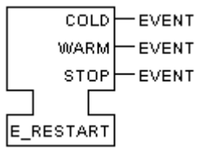
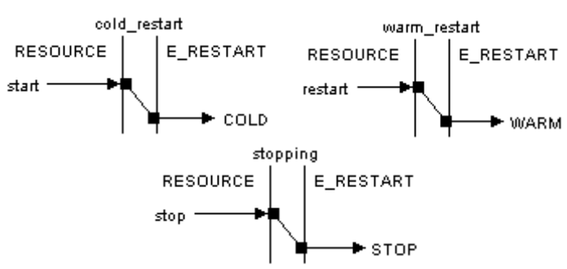
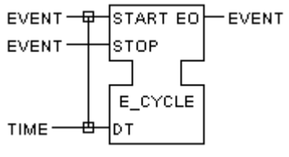
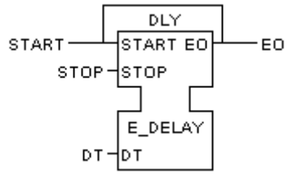
| No. | Description | |
|---|---|---|
| | **Interface** | **gorithms/Service sequences** |
| **1** | **拆分事件** | |
| |  |  |
| 在EI发生事件会导致在EO1、EO2、…、EOn发生事件（上例中的n=2）。 | | |
| **2** | **多个事件的合并(OR)** | |
| |  |  |
| 在任何输入EI1、EI2、…、EIn处发生事件会导致在EO处发生事件（在上面的示例中n=2）。 | | |
| **3** | **两个事件的约会** | |
| |  |  |
| **4** | **事件的许可传播** | |
| |  |  |
| **5** | **两个事件之间的选择** | |
| |  |  |

**Table A.1** (2 of 6)

| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **6** | **Switching (demultiplexing) an event** | |
| |  |  |
| **7** | **Delayed propagation of an event** | |
| |  | An event at EO is generated at a time interval DT after the occurrence of an event at the START input. The event delay is cancelled by an occurrence of an event at the STOP input. If multiple events occur at the START input before the occurrence of an event at EO, only a single event occurs at EO, at a time DT after the first event occurrence at the START input. No event delay will be initiated if an event occurs at the START input with a value of DT which is not greater than t#0s. |
| **8** | **Generation of restart events** | |
| |  |  |
| a) An event is issued at the COLD output upon "cold restart" of the associated resource.<br>b) An event is issued at the WARM output upon "warm restart" of the associated resource.<br>c) An event is issued at the STOP output (if possible) prior to "stopping" of the associated resource.<br>NOTE 1 See IEC 61131-1 for a discussion of "cold restart" and "warm restart". | | |
| **9** | **Periodic (cyclic) generation of an event** | |
| |  <br>An event occurs at EO at an interval DT after the occurrence of an event at START, and at intervals of DT thereafter until the occurrence of an event at STOP. |  |





表A.1（6个中的2个）

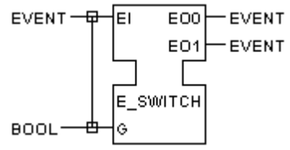
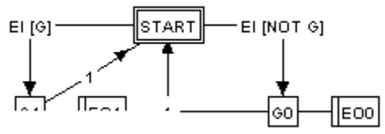
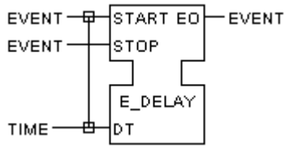
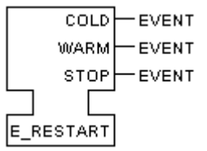
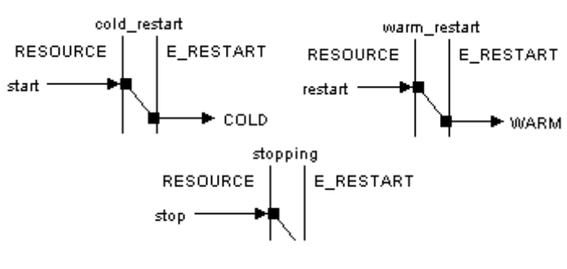
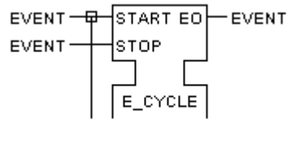
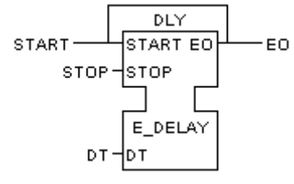
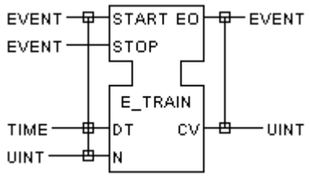
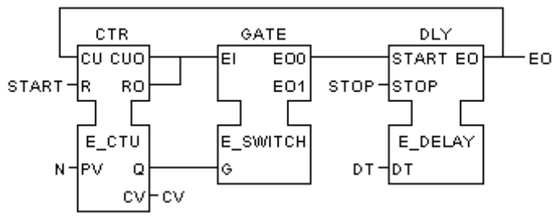
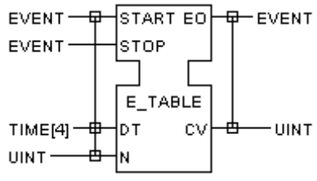
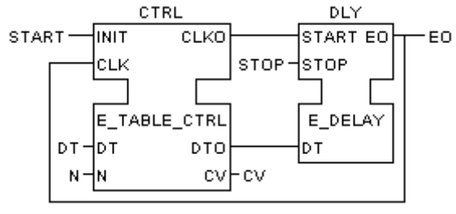
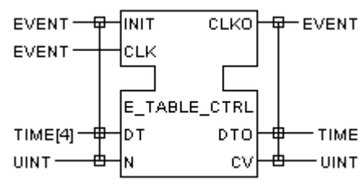
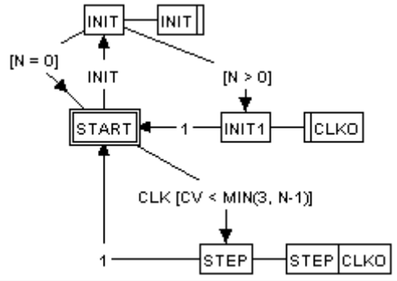
| No. | Description | |
|---|---|---|
| | **Interface** | **ervice sequences** |
| **6** | **切换（解复用）事件** | |
| |  |  |
| **7** | **事件的延迟传播** | |
| |  | 在START输入处发生事件后，在时间间隔DT处生成EO处的事件。事件延迟会因STOP输入发生事件而取消。如果在EO处发生事件之前在START输入处发生了多个事件，则在EO处仅发生一个事件，在START输入处发生第一个事件之后的时间DT。如果事件发生在<br><br>带有不大于t#0s的DT值的START输入。 |
| **8** | **重启事件的产生** | |
| |  |  |
| a) 在关联资源"冷启动"时，在COLD输出处发出事件。<br>b) 在关联资源"热重启"时，在WARM输出处发出一个事件。<br>在"停止"相关资源之前，在STOP输出（如果可能）处发出一个事件。<br>注1有关"冷启动"和"热启动"的讨论，请参见IEC61131-1。 | | |
| **9** | **定期（循环）生成事件** | |
| |  <br>在START事件发生后，以DT间隔在EO发生事件，之后以DT间隔发生，直到在STOP事件发生。 |  |

**Table A.1** (3 of 6)

| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **10** | **Generation of a finite train of events** | |
| |  |  <br>NOTE 2 See table entry 18 for a definition of the E_CTU type. |
| An event occurs at EO at an interval DT after the occurrence of an event at START, and at intervals of DT thereafter, until N occurrences have been generated or an event occurs at the STOP input.<br><br>NOTE 3 The count CV is reset whenever an event occurs at the START interface, but the delay does not restart unless it is already stopped. This behavior maintains the inter-EO interval when restarting the count. | | |
| **11** | **Generation of a finite train of events (table driven)** | |
| |  |  |
| An event occurs at EO at an interval DT[0] after the occurrence of an event at START. A second event occurs at an interval DT[1] after the first, etc., until N occurrences have been generated or an event occurs at the STOP input. The current event count is maintained at the CV output.<br><br>NOTE 4 In this example implementation, N <= 4.<br><br>NOTE 5 Implementation using the E_TABLE_CTRL function block type illustrated below is not a normative requirement. Equivalent functionality can be implemented by various means. | | |
| |  |  |
| ALGORITHM INIT IN ST:<br>  CV:= 0;<br>  DTO:= DT[0];<br>END_ALGORITHM | ALGORITHM STEP IN ST:<br>  CV:= CV+1;<br>  DTO:= DT[CV];<br>END_ALGORITHM | |





表A.1（3个，共6个）

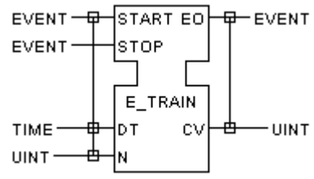
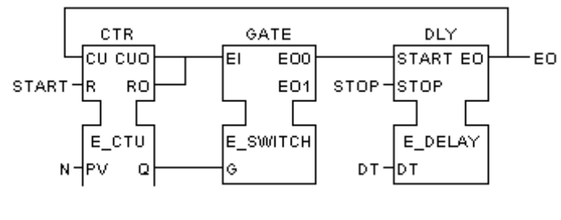
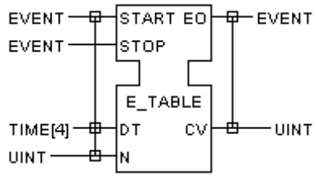
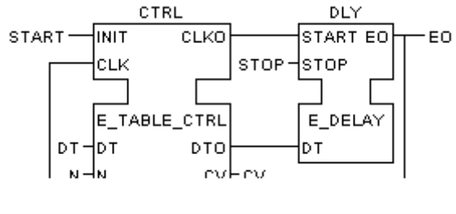
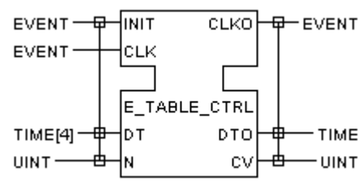
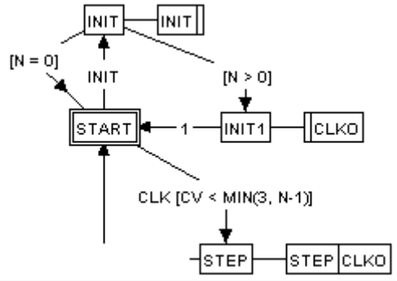
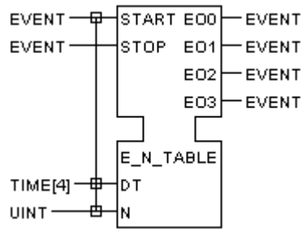
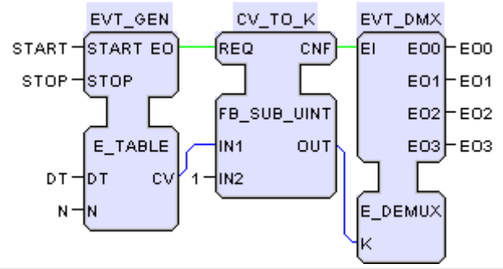
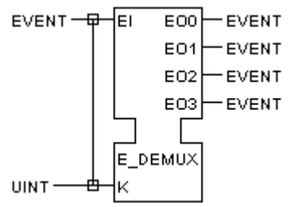
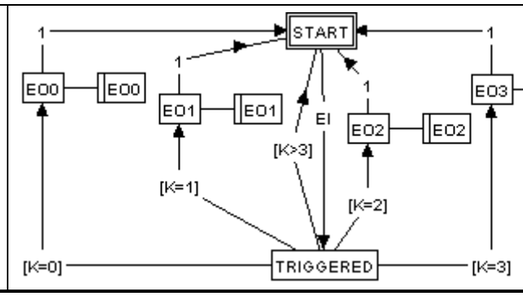
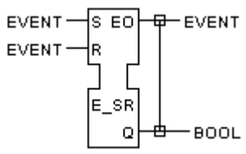
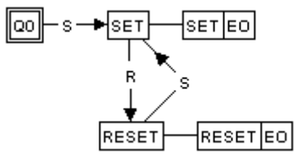
| No. | Description | |
|---|---|---|
| | **Interface** | **rvice sequences** |
| **10** | **生成有限的事件序列** | |
| |  |  <br>注2有关E_CTU类型的定义，请参见表条目18。 |
| 在START处发生事件后，在间隔DT处发生事件，并在其后间隔DT处发生事件，直到已生成N次事件或在STOP输入处发生事件。<br><br>注3每当START接口发生事件时，计数CV都会重置，但延迟不会重新开始，除非它已经停止。此行为在重新开始计数时保持interEO间隔。 | | |
| **11** | **生成有限的事件序列（表驱动）** | |
| |  |  |
| 在START发生事件之后，在EO处以间隔DT[0]发生事件。第二个事件在第一个事件之后的间隔DT[1]处发生，依此类推，直到发生N次或在STOP输入处发生事件。当前事件计数保持在CV输出。<br><br>注4在此示例实现中，N<=4。<br><br>注5使用如下所示的E_TABLE_CTRL功能块类型的实现不是标准要求。等效功能可以通过多种方式实现。 | | |
| |  |  |
| ST中的算法初始化：<br>  CV:= 0;<br>  DTO:= DT[0];<br>END_ALGORITHM | ST中的算法步骤：<br>  CV:= CV+1;<br>  DTO:= DT[CV];<br>END_ALGORITHM | |

**Table A.1** (4 of 6)

| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **12** | **Generation of a finite train of separate events (table driven)** | |
|  |  | |
| An event occurs at EO0 at an interval DT[0] after the occurrence of an event at START. An event occurs at EO1 an interval DT[1] after the occurrence of the event at EO0, etc., until N occurrences have been generated or an event occurs at the STOP input.<br><br>NOTE 6 In this example implementation, N <= 4.<br><br>NOTE 7 Implementation using the E_DEMUX function block type illustrated below is not a normative requirement. Equivalent functionality can be implemented by various means. | | |
|  |  | |
| **13** | **Event-driven bistable** | |
|  |  | |
| The output Q is set to 1 (TRUE) upon the occurrence of an event at the S input, and is reset to 0 (FALSE) upon the occurrence of an event at the R input. An event is issued at the EO output when the value of Q changes. | | |
| ALGORITHM SET IN ST: (* Set Q *)<br>  Q:= TRUE;<br>END_ALGORITHM | ALGORITHM RESET IN ST: (* Reset Q *)<br>  Q:= FALSE;<br>END_ALGORITHM | |





表A.1（4个，共6个）

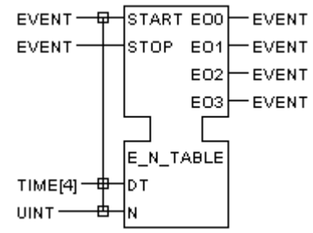
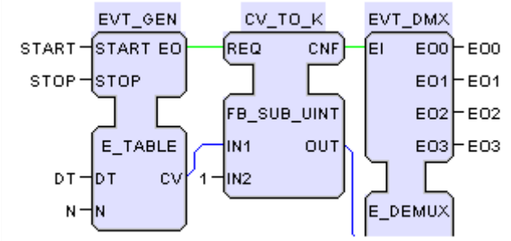
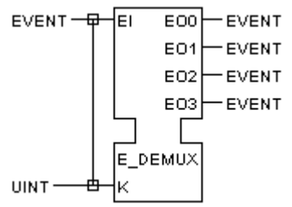
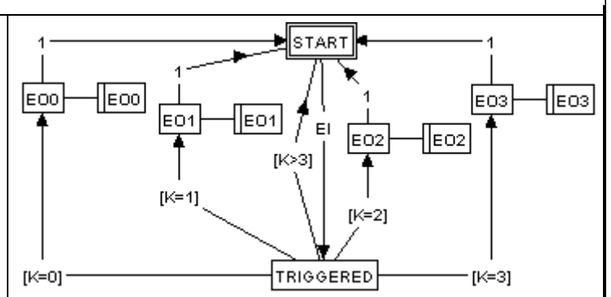
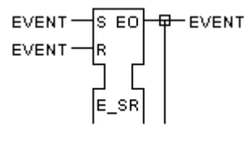
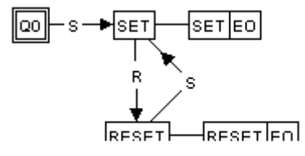
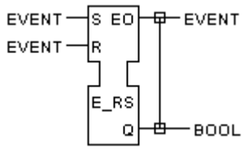
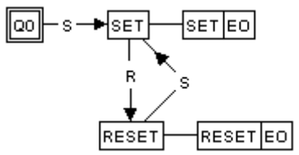
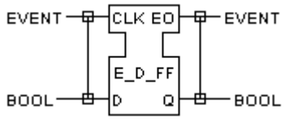
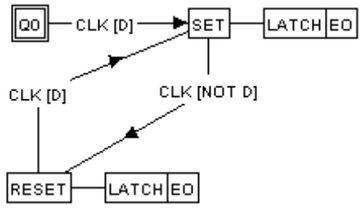
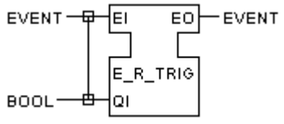
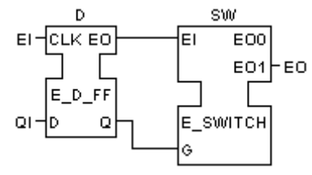
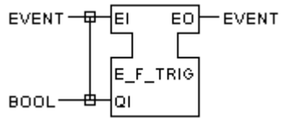
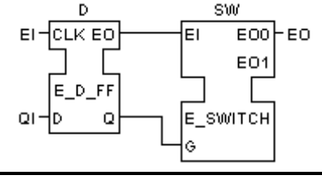
| No. | Description | |
|---|---|---|
| | **Interfac** | **nces** |
| **12** | **生成有限序列的单独事件（表驱动）** | |
|  |  | |
| 在START发生事件之后，在EO0以间隔DT[0]发生事件。在EO0处发生事件之后的间隔DT[1]处，事件在EO1处发生，依此类推，直到已生成N次事件或在STOP输入处发生事件。<br><br>注6在此示例实现中，N<=4。<br><br>注7使用如下所示的E_DEMUX功能块类型的实现不是标准要求。等效功能可以通过多种方式实现。 | | |
|  |  | |
| **13** | **Event-driven bistable** | |
|  |  | |
| 输出Q在S输入发生事件时设置为1(TRUE)，并在R输入发生事件时重置为0(FALSE)。当Q的值改变时，EO输出会发出一个事件。 | | |
| ST中的算法集：（*集Q*）<br>  Q:= TRUE;<br>END_ALGORITHM | ST中的算法重置：（*重置Q*）<br>  Q:= FALSE;<br>END_ALGORITHM | |

**Table A.1** (5 of 6)

| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **14** | **Event-driven bistable** | |
| |  |  |
| | The output Q is set to 1 (TRUE) upon the occurrence of an event at the S input, and is reset to 0 (FALSE) upon the occurrence of an event at the R input. An event is issued at the EO output when the value of Q changes.<br><br>NOTE 8 The implementation of this function block type is identical to E_SR. Both E_SR and E_RS are implemented for consistency with the SR and RS types of IEC 61131-3, although there is no "dominance" of events as there would be for level-controlled R and S inputs. | |
| **15** | **D (Data latch) bistable** | |
| |  |  |
| | | ALGORITHM LATCH IN ST:<br>  Q:= D;<br>END_ALGORITHM |
| **16** | **Boolean rising edge detection** | |
| |  |  |
| **17** | **Boolean falling edge detection** | |
| |  |  |





表A.1（5个，共6个）

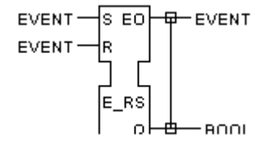
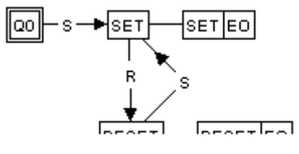
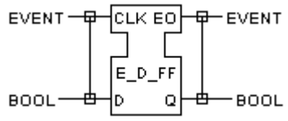
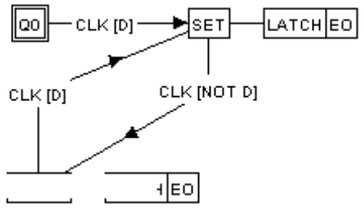
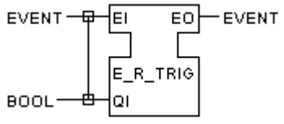
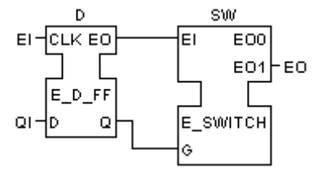
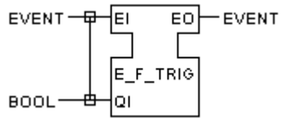
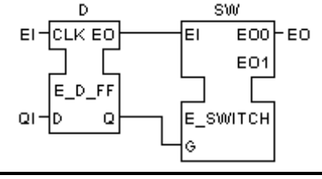
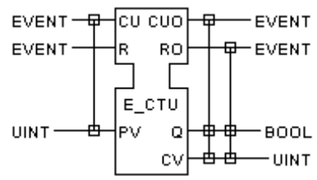
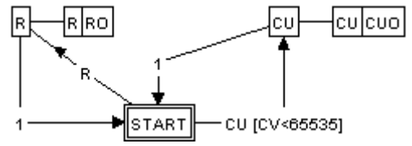
| No. | Description | |
|---|---|---|
| | **Interface** | **ECC/Algorithms/Service sequences** |
| **14** | **Event-driven bistable** | |
| |  |  |
| | 输出Q在S输入发生事件时设置为1(TRUE)，并在R输入发生事件时重置为0(FALSE)。当Q的值改变时，EO输出会发出一个事件。<br><br>注8此功能块类型的实现与E_SR相同。E_SR和E_RS都是为了与IEC61131-3的SR和RS类型保持一致而实现的，尽管没有像电平控制的R和S输入那样的事件"主导"。 | |
| **15** | **D (Data latch) bistable** | |
| |  |  |
| | | ST中的算法锁存器： |
| **16** | **布尔上升沿检测** | |
| |  |  |
| **17** | **布尔下降沿检测** | |
| |  |  |

**Table A.1** (6 of 6)

| No. | Description | |
|---|---|---|
| | Interface | ECC/Algorithms/Service sequences |
| 18 | Event-driven up counter | |
| |  |  |
| | ALGORITHM R IN ST: (* Reset *)<br>CV:= 0;<br>Q:= 0;<br>END_ALGORITHM | ALGORITHM CU IN ST: (* Count Up *)<br>CV:= CV + 1;<br>Q:= (CV >= PV);<br>END_ALGORITHM |

Graphical shorthand notations may be substituted for the E_SPLIT and E_MERGE blocks defined in Table A.1. For example, the shorthand (implicit) representation shown in Figure A.1b is equivalent to the explicit representation in Figure A.1a.

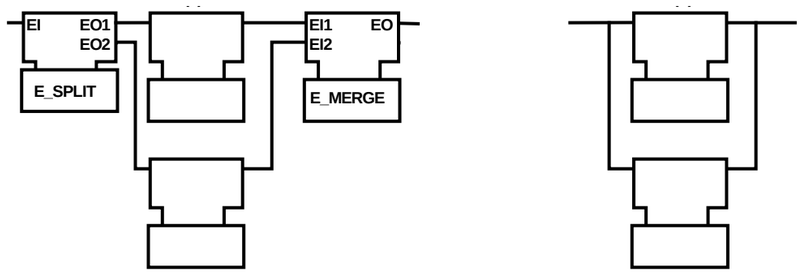



**Figure A.1a – Explicit representation**



**Figure A.1b – Implicit representation**

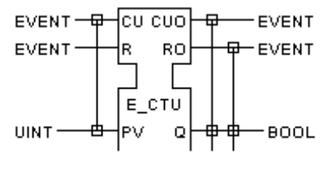
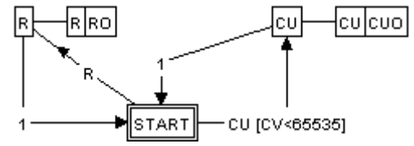
NOTE Irrelevant details are suppressed in the above figure.

**Figure A.1 – Event split and merge**

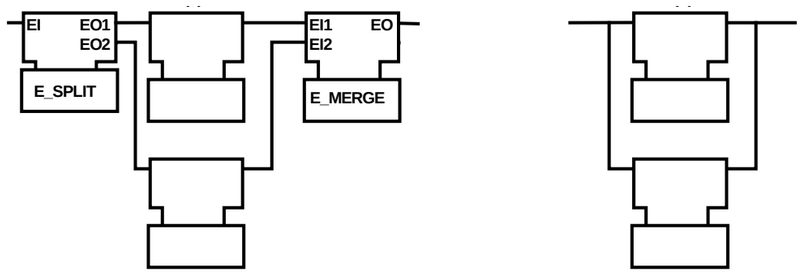




表A.1（6个中的6个）

| No. | Description | |
|---|---|---|
| | Interface | ms/Service sequences |
| 18 | 事件驱动的计数器 | |
| |  |  |
| | ST中的算法R: （*重置*）<br>CV:= 0;<br>Q:= 0;<br>END_ALGORITHM | ST中的算法CU: （*向上计数*）<br>CV:= CV + 1;<br>Q:= (CV >= PV);<br>END_ALGORITHM |

图形简写符号可以代替表A.1中定义的E_SPLIT和E_MERGE块。例如，图A.1b中所示的简写（隐式）表示等效于图A.1a中的显式表示。




**图A.1a 显式表示**



**图A.1b 隐式表示**

注意不相关的细节在上图中被隐藏。

**图A.1 事件拆分和合并**

# Annex B
(normative)

# Textual syntax

## B.1 Syntax specification technique

The textual constructs in Annex B are specified in terms of a *syntax*, which specifies the allowable combinations of symbols which can be used to define a program; and a set of *semantics*, which specify the meanings of the symbol combinations defined by the syntax.

A **syntax** is defined by a set of *terminal symbols* to be utilized for program specification; a set of *non-terminal symbols* defined in terms of the terminal symbols; and a set of *production rules* specifying those definitions.

The **terminal symbols** for textual specifications of entities defined in this standard consist of combinations of the characters in the character set given as Table 2 – Row 00 of the "Basic Latin to CJK Compatibility" table linked to Clause 33 defined in ISO/IEC 10646:2003.

For the purposes of this standard, terminal textual symbols consist of the appropriate character string enclosed in paired single or double quotes. For example, a terminal symbol represented by the character string ABC can be represented by either `"ABC"` or `'ABC'`.

This allows the representation of strings containing either single or double quotes; for instance, a terminal symbol consisting of the double quote itself would be represented by `'"'`

A special terminal symbol utilized in this syntax is the "null string", that is, a string containing no characters. This is represented by the terminal symbol `NIL`.

**Non-terminal** textual symbols are represented by strings of lower-case letters, numbers, and the underline character (_), beginning with a lower-case letter. For instance, the strings `nonterm1` and `non_term_2` are valid nonterminal symbols, while the strings `3nonterm` and `_nonterm4` are not.

The **production rules** given in this standard form an *extended grammar* in which each rule has the form

```
non_terminal_symbol::= extended_structure
```

This rule can be read as:

"A `non_terminal_symbol` can consist of an `extended_structure`."

Extended structures can be constructed according to the following rules:

a) The null string, `NIL`, is an extended structure.
b) A terminal symbol is an extended structure.
c) A non-terminal symbol is an extended structure.
d) If S is an extended structure, then the following expressions are also extended structures:
   - `(S)`, meaning S itself.
   - `{S}`, *closure*, meaning zero or more concatenations of S.





# 文本语法

## 语法规范技术

附件B中的文本结构是根据语法规定的，它规定了可用于定义程序的允许的符号组合；和一组语义，它们指定由语法定义的符号组合的含义。

语法由一组用于程序规范的终端符号定义；根据终端符号定义的一组非终端符号；以及一组指定这些定义的生产规则。

本标准中定义的实体的文本规范的终端符号由字符集中的字符组合组成，如表2 与ISOIEC10646:2003中定义的第33条相关联的“基本拉丁语到CJK兼容性”表的第00行.

就本标准而言，终端文本符号由包含在成对单引号或双引号中的适当字符串组成。例如，由字符串ABC表示的终结符号可以用"ABC"或'ABC'来表示。

这允许表示包含单引号或双引号的字符串；例如，由双引号本身组成的终结符号将由'"'表示

此语法中使用的特殊终端符号是“空字符串”，即不包含字符的字符串。这由终端符号NIL表示。

非终结文本符号由小写字母、数字和下划线字符(_)组成的字符串表示，以小写字母开头。例如，字符串nonterm1和non_term_2是有效的非终结符，而字符串3nonterm和_nonterm4不是。

本标准中给出的产生式规则形成了一个扩展语法，其中每个规则都有以下形式

```
_terminal_symbol::= extended_structure
```

这条规则可以理解为：

“一个non_terminal_symbol可以由一个extended_structure组成。”

可以根据以下规则构建扩展结构：

a)空字符串NIL是一个扩展结构。
b)终端符号是一个扩展结构。
c)非终结符号是扩展结构。
d)如果S是扩展结构，则以下表达式也是扩展结构：
   - (S)，表示S本身。
   - {S ，闭包，表示S的零个或多个串联。

- `[S]`, *option*, meaning zero or one occurrence of S.

e) If `S1` and `S2` are extended structures, then the following expressions are extended structures:

- `S1 | S2`, *alternation*, meaning a choice of `S1` or `S2`.
- `S1 S2`, *concatenation*, meaning `S1` followed by `S2`.

f) Concatenation *precedes* alternation, that is, `S1 | S2 S3` is equivalent to `S1 | (S2 S3)`, and `S1 S2 | S3` is equivalent to `(S1 S2) | S3`.

**Semantics** are defined in this standard by appropriate natural language text, accompanying the production rules, which references the descriptions provided in the appropriate clauses. Standard options available to the user and vendor are specified in these semantics.

In some cases it is more convenient to embed semantic information in an extended structure. In such cases, this information is delimited by paired angle brackets, for example, `<semantic information>`.

## B.2 Function block and subapplication type specification

### B.2.1 Function block type specification

The syntax defined in B.2.1 can be used for the textual specification of *function block types* according to the rules given in Clauses 5 and 6 of this standard.

**SYNTAX:**

```
fb_type_declaration::=
     'FUNCTION_BLOCK' fb_type_name
     fb_interface_list
     [fb_internal_variable_list] <only for basic FB>
     [fb_instance_list]      <only for composite FB>
     [plug_list]
     [socket_list]
     [fb_connection_list]     <only for composite FB>
     [fb_ecc_declaration]     <only for basic FB>
     {fb_algorithm_declaration}  <only for basic FB>
     [fb_service_declaration]
     'END_FUNCTION_BLOCK'


fb_interface_list::=
     [event_input_list]
     [event_output_list]
     [input_variable_list]
     [output_variable_list]


event_input_list::=
     'EVENT_INPUT'
     {event_input_declaration}
     'END_EVENT'


event_output_list::=
     'EVENT_OUTPUT'
     {event_output_declaration}
     'END_EVENT'


event_input_declaration::= event_input_name [ ':' event_type ]
     ['WITH' input_variable_name {',' input_variable_name}] ';'


event_output_declaration::= event_output_name [ ':' event_type ]
     ['WITH' output_variable_name {',' output_variable_name}] ';'
```





[S]，option，表示S出现零次或一次。

e)如果S1和S2是扩展结构，则以下表达式是扩展结构：

- S1S2，交替，意思是选择S1或S2。

  S1S2，串联，意思是S1后跟S2。

f) 串联先于交替，即S1S2S3等价于S1(S2S3)，S1S2S3等价于(S1S2)S3。

语义在本标准中由适当的自然语言文本定义，伴随生产规则，它引用了适当条款中提供的描述。这些语义中指定了用户和供应商可用的标准选项。

在某些情况下，将语义信息嵌入扩展结构中会更方便。在这种情况下，此信息由成对的尖括号分隔，例如<semanticinformation>。

## B.2 功能块和子应用类型规范

### 功能块类型规范

根据本标准第5章和第6章给出的规则，B.2.1中定义的语法可用于功能块类型的文本规范。

**SYNTAX:**

```
fb_t
     'FUNCTION_BLOCK'fb_type_namefb_interface_list[fb_internal_variable
     _list]<仅适用于基本FB>[fb_instance_list]<仅适用于复合FB>[plug_list][s
     ocket_list][fb_connection_list]<仅适用于复合FB>[fb_ecc_declaration]<
     仅适用于基本FB>{fb_algorithm_declaration <仅适用于基本FB>[fb_servi
     ce_declaration]'END_FUNCTION_BLOCK'


fb_interface_list::=
     [event_input_list]
     [event_output_list]
     [input_variable_list]
     [output_variable_list]


event_input_list::=
     'EVENT_INPUT'
     {event_input_declaration}
     'END_EVENT'


event_output_list::=
     'EVENT_OUTPUT'
     {event_output_declaration}
     'END_EVENT'


event_input_declaration::= event_input_name [ ':' event_type ]
     ['WITH' input_variable_name {',' input_variable_name}] ';'


event_output_declaration::= event_output_name [ ':' event_type ]
     ['WITH' output_variable_name {',' output_variable_name}] ';'
```

```
input_variable_list::=
     'VAR_INPUT' {input_var_declaration ';'} 'END_VAR'

output_variable_list::=
     'VAR_OUTPUT' {output_var_declaration ';'} 'END_VAR'

fb_internal_variable_list::=
     'VAR' {internal_var_declaration ';'} 'END_VAR'

input_var_declaration::=
     input_variable_name {',' input_variable_name} ':' var_spec_init

output_var_declaration::=
     output_variable_name {',' output_variable_name} ':' var_spec_init

internal_var_declaration::=
     internal_variable_name {',' internal_variable_name}
     ':' var_spec_init

var_spec_init::= located_var_spec_init <as specified in IEC 61131-3>

fb_instance_list::= 'FBS'
     {fb_instance_definition ';'}
     'END_FBS'

fb_instance_definition::= fb_instance_name ':' fb_type_name [parameters]

plug_list::= 'PLUGS'
     {plug_name ':' adapter_type_name [parameters] ';'}
     'END_PLUGS'

socket_list::= 'SOCKETS'
     {socket_name ':' adapter_type_name [parameters] ';'}
     'END_SOCKETS'

fb_connection_list::=   <may be empty, e.g. for basic FB>
     [event_conn_list]
     [data_conn_list]
     [adapter_conn_list]

event_conn_list::=
     'EVENT_CONNECTIONS'
     {event_conn}
     'END_CONNECTIONS'

event_conn::= event_conn_source 'TO' event_conn_destination ';'


event_conn_source::= ([plug_name '.'] event_input_name)
     | ((fb_instance_name | socket_name) '.' event_output_name)

event_conn_destination::= ([plug_name '.'] event_output_name)
     | ((fb_instance_name | socket_name) '.' event_input_name)

data_conn_list::=
     'DATA_CONNECTIONS'
     {data_conn}
     'END_CONNECTIONS'

data_conn::= data_conn_source 'TO' data_conn_destination ';'
```



61499-1 © IEC:2012 – 65 –

```
input_variable_list::=
     'VAR_INPUT' {input_var_declaration ';'} 'END_VAR'

output_variable_list::=
     'VAR_OUTPUT' {output_var_declaration ';'} 'END_VAR'

fb_internal_variable_list::=
     'VAR' {internal_var_declaration ';'} 'END_VAR'

input_var_declaration::=
     input_variable_name {',' input_variable_name} ':' var_spec_init

output_var_declaration::=
     output_variable_name {',' output_variable_name} ':' var_spec_init

internal_var_declaration::=
     internal_variable_name {',' internal_variable_name}
```

var_spec_init::=located_var_spec_init<按照IEC61131-3的规定>

```
fb_instance_list::= 'FBS'
     {fb_instance_definition ';'}
     'END_FBS'

fb_instance_definition::= fb_instance_name ':' fb_type_name [parameters]

plug_list::= 'PLUGS'
     {plug_name ':' adapter_type_name [parameters] ';'}
     'END_PLUGS'

socket_list::= 'SOCKETS'
     {socket_name ':' adapter_type_name [parameters] ';'}
```

fb_connection_list::=<可能为空，例如对于基本FB>[event_conn_list][data_conn_list]
[adapter_conn_list]

```
event_conn_list::=
     'EVENT_CONNECTIONS'
     {event_conn}
     'END_CONNECTIONS'

event_conn::= event_conn_source 'TO' event_conn_destination ';'


event_conn_source::= ([plug_name '.'] event_input_name)
     | ((fb_instance_name | socket_name) '.' event_output_name)

event_conn_destination::= ([plug_name '.'] event_output_name)
     | ((fb_instance_name | socket_name) '.' event_input_name)

data_conn_list::=
     'DATA_CONNECTIONS'
     {data_conn}
     'END_CONNECTIONS'

data_conn::= data_conn_source 'TO' data_conn_destination ';'
```

```
data_conn_source::= ([plug_name '.'] input_variable_name)
     | ((fb_instance_name | socket_name) '.' output_variable_name)

data_conn_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_instance_name | socket_name) '.' input_variable_name)

adapter_conn_list::=
     'ADAPTER_CONNECTIONS'
     {adapter_conn}
     'END_CONNECTIONS'

adapter_conn::=
     ((fb_instance_name '.' plug_name ) | socket_name)
     'TO' ((fb_instance_name '.' socket_name ) | plug_name) ';'

fb_ecc_declaration::=
     'EC_STATES'
     {ec_state}    <first state is initial state>
     'END_STATES'
     'EC_TRANSITIONS'
     {ec_transition}
     'END_TRANSITIONS'

ec_state::= ec_state_name
     [':' ec_action {',' ec_action}] ';'

ec_action::= algorithm_name | ('->' ec_action_output)
     | (algorithm_name '->' ec_action_output)

ec_action_output:= ([plug_name '.'] event_output_name)
     | (socket_name '.' event_input_name)

ec_transition::=
     ec_state_name
     'TO' ec_state_name
     ':=' ec_transition_condition ';'

ec_transition_condition::= '1'
     | ec_transition_event | '[' guard_condition ']'
     | ec_transition_event '[' guard_condition ']'


ec_transition_event::= ([plug_name '.'] event_input_name)
     | (socket_name '.' event_output_name)

guard_condition::= expression <over ec_expression_operand elements>
     <as defined in IEC 61131-3>
     <Shall evaluate to a BOOL value>

ec_expression_operand::=
     ([(plug_name | socket_name) '.'] input_variable_name)
     | ([(plug_name | socket_name) '.'] output_variable_name)
     | internal_variable_name
     | constant

fb_algorithm_declaration::=
     'ALGORITHM' algorithm_name 'IN' language_type ':'
     [temp_var_decls]
     algorithm_body
     'END_ALGORITHM'

temp_var_decls::= <as defined in IEC 61131-3>
```



– 66 – 61499-1 © IEC:2012

```
data_conn_source::= ([plug_name '.'] input_variable_name)
     | ((fb_instance_name | socket_name) '.' output_variable_name)

data_conn_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_instance_name | socket_name) '.' input_variable_name)

adapter_conn_list::=
     'ADAPTER_CONNECTIONS'
     {adapter_conn}
     'END_CONNECTIONS'

adapter_conn::=
     ((fb_instance_name '.' plug_name ) | socket_name)
                                                  lug_name) ';'
```

fb_ecc_declaration::='EC_STATES'{ec_state <第一个状态是初始状态>'EN
D_STATES''EC_TRANSITIONS'{ec_transition 'END_TRANSITIONS'

```
ec_state::= ec_state_name
     [':' ec_action {',' ec_action}] ';'

ec_action::= algorithm_name | ('->' ec_action_output)
     | (algorithm_name '->' ec_action_output)

ec_action_output:= ([plug_name '.'] event_output_name)
     | (socket_name '.' event_input_name)

ec_transition::=
     ec_state_name
     'TO' ec_state_name
     ':=' ec_transition_condition ';'

ec_transition_condition::= '1'
     | ec_transition_event | '[' guard_condition ']'
     | ec_transition_event '[' guard_condition ']'


ec_transition_event::= ([plug_name '.'] event_input_name)
```

guard_condition::=expression<overec_expression_operandelements><asdefinedinIEC61131−3><S
hallevaluatetoaBOOLvalue>

```
ec_expression_operand::=
     ([(plug_name | socket_name) '.'] input_variable_name)
     | ([(plug_name | socket_name) '.'] output_variable_name)
     | internal_variable_name
     | constant

fb_algorithm_declaration::=
     'ALGORITHM' algorithm_name 'IN' language_type ':'
     [temp_var_decls]
     algorithm_body
```

temp_var_decls::=<如IEC61131−3中定义>

```
algorithm_body::= <as defined in compliant standards>

fb_service_declaration::=
     'SERVICE' service_interface_name '/' service_interface_name
     {service_sequence}
     'END_SERVICE'

service_interface_name::= fb_type_name | 'RESOURCE'

service_sequence::=
     'SEQUENCE' sequence_name
     {service_transaction ';'}
     'END_SEQUENCE'

service_transaction::=
     [input_service_primitive] '->' output_service_primitive
     {'->' output_service_primitive}

input_service_primitive::= service_interface_name '.'
     ([plug_name '.'] event_input_name
      | socket_name '.' event_output_name)
     ['+' | '-']
     '(' [input_variable_name {',' input_variable_name}] ')'

output_service_primitive::= service_interface_name '.' ('NULL' |
     ([plug_name '.'] event_output_name
      | socket_name '.' event_input_name)
     ['+' | '-']
     '(' [output_variable_name {',' output_variable_name}] ')')

algorithm_name::= identifier

ec_state_name::= identifier

event_input_name::= identifier

event_output_name::= identifier

event_type::= identifier

fb_instance_name::= identifier

fb_type_name::= identifier

input_variable_name::= identifier

internal_variable_name::= identifier

language_type::= identifier

output_variable_name::= identifier

plug_name::= identifier

sequence_name::= identifier

socket_name::= identifier
```



algorithm_body::=<在兼容标准中定义>

```
fb_service_declaration::=
     'SERVICE' service_interface_name '/' service_interface_name
     {service_sequence}
     'END_SERVICE'

service_interface_name::= fb_type_name | 'RESOURCE'

service_sequence::=
     'SEQUENCE' sequence_name
     {service_transaction ';'}
     'END_SEQUENCE'

service_transaction::=
     [input_service_primitive] '->' output_service_primitive
     {'->' output_service_primitive}

input_service_primitive::= service_interface_name '.'
     ([plug_name '.'] event_input_name
      | socket_name '.' event_output_name)
     ['+' | '-']
     '(' [input_variable_name {',' input_variable_name}] ')'

output_service_primitive::= service_interface_name '.' ('NULL' |
     ([plug_name '.'] event_output_name
      | socket_name '.' event_input_name)
     ['+' | '-']
     '(' [output_variable_name {',' output_variable_name}] ')')

algorithm_name::= identifier

ec_state_name::= identifier

event_input_name::= identifier

event_output_name::= identifier

event_type::= identifier

fb_instance_name::= identifier

fb_type_name::= identifier

input_variable_name::= identifier

internal_variable_name::= identifier

language_type::= identifier

output_variable_name::= identifier

plug_name::= identifier

sequence_name::= identifier

socket_name::= identifier
```

### B.2.2 Subapplication type specification

The syntax defined in this subclause can be used for the textual specification of *subapplication types* according to the rules given in 5.4.1.

The productions given in B.2.1 also apply to this subclause.

**SYNTAX:**

```
subapplication_type_declaration::=
     'SUBAPPLICATION' subapp_type_name
          subapp_interface_list
          [fb_instance_list]
          [subapp_instance_list]
          [plug_list]
          [socket_list]
          [subapp_connection_list]
     'END_SUBAPPLICATION'


subapp_interface_list::=
     [subapp_event_input_list]
     [subapp_event_output_list]
     [input_variable_list]
     [output_variable_list]


subapp_event_input_list::=
     'EVENT_INPUT'
     {subapp_event_input_declaration}
     'END_EVENT'


subapp_event_output_list::=
     'EVENT_OUTPUT'
     {subapp_event_output_declaration}
     'END_EVENT'


subapp_event_input_declaration::=
     event_input_name [ ':' event_type ] ';'


subapp_event_output_declaration::=
     event_output_name [ ':' event_type ] ';'


subapp_instance_list::= 'SUBAPPS'
     {subapp_instance_definition ';'}
     'END_SUBAPPS'


subapp_instance_definition::= subapp_instance_name ':' subapp_type_name


subapp_connection_list::=
     [subapp_event_conn_list]
     [subapp_data_conn_list]
     [adapter_conn_list]


subapp_event_conn_list::=
     'EVENT_CONNECTIONS'
     {subapp_event_conn}
     'END_CONNECTIONS'


subapp_event_conn::= subapp_event_source 'TO' subapp_event_destination ';'


subapp_event_source:: = ([plug_name '.'] event_input_name)
     | ((fb_subapp_name | socket_name) '.' event_output_name
```





### 子应用类型规范

根据5.4.1中给出的规则，本子条款中定义的语法可用于子应用类型的文本规范。

B.2.1中给出的产生式也适用于本条。

**SYNTAX:**

```
subapplication_type_declaration::=
     'SUBAPPLICATION' subapp_type_name
          subapp_interface_list
          [fb_instance_list]
          [subapp_instance_list]
          [plug_list]
          [socket_list]
          [subapp_connection_list]
     'END_SUBAPPLICATION'


subapp_interface_list::=
     [subapp_event_input_list]
     [subapp_event_output_list]
     [input_variable_list]
     [output_variable_list]


subapp_event_input_list::=
     'EVENT_INPUT'
     {subapp_event_input_declaration}
     'END_EVENT'


subapp_event_output_list::=
     'EVENT_OUTPUT'
     {subapp_event_output_declaration}
     'END_EVENT'


subapp_event_input_declaration::=
     event_input_name [ ':' event_type ] ';'


subapp_event_output_declaration::=
     event_output_name [ ':' event_type ] ';'


subapp_instance_list::= 'SUBAPPS'
     {subapp_instance_definition ';'}
     'END_SUBAPPS'


subapp_instance_definition::= subapp_instance_name ':' subapp_type_name


subapp_connection_list::=
     [subapp_event_conn_list]
     [subapp_data_conn_list]
     [adapter_conn_list]


subapp_event_conn_list::=
     'EVENT_CONNECTIONS'
     {subapp_event_conn}
     'END_CONNECTIONS'


subapp_event_conn::= subapp_event_source 'TO' subapp_event_destination ';'


subapp_event_source:: = ([plug_name '.'] event_input_name)
     | ((fb_subapp_name | socket_name) '.' event_output_name
```

```
subapp_event_destination::= ([plug_name '.'] event_output_name)
     | ((fb_subapp_name | socket_name) '.' event_input_name)

fb_subapp_name::= fb_instance_name | subapp_instance_name

subapp_data_conn_list::=
     'DATA_CONNECTIONS'
     {subapp_data_conn}
     'END_CONNECTIONS'

subapp_data_conn::= subapp_data_source 'TO' subapp_data_destination ';'

subapp_data_source::= ([plug_name '.'] input_variable_name)
     | ((fb_subapp_name | socket_name) '.' output_variable_name)

subapp_data_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_subapp_name | socket_name) '.' input_variable_name)

subapp_type_name::= identifier

subapp_instance_name::= identifier
```

## B.3 Configuration elements

The syntax defined in this clause can be used for the textual specification of *resource types*, *device types*, *segment types*, *applications*, and *system configurations* according to the rules given in Clause 7.

The productions given in Clause B.2 also apply to this clause.

**SYNTAX:**

```
application_configuration::=
     'APPLICATION' application_name
     [fb_instance_list]
     [subapp_instance_list]
     [subapp_connection_list]
     'END_APPLICATION'

system_configuration::= 'SYSTEM' system_name
      {application_configuration}
      device_configuration
      {device_configuration}
      [mappings]
      [segments]
      [links]
     'END_SYSTEM'

segments::= 'SEGMENTS'
      segment
      {segment}
     'END_SEGMENTS'

segment::= segment_name ':' segment_type_name [parameters] ';'

links::= 'LINKS'
      link
      {link}
     'END_LINKS'
```



61499-1 © IEC:2012 – 69 –


```
subapp_event_destination::= ([plug_name '.'] event_output_name)
     | ((fb_subapp_name | socket_name) '.' event_input_name)

fb_subapp_name::= fb_instance_name | subapp_instance_name

subapp_data_conn_list::=
     'DATA_CONNECTIONS'
     {subapp_data_conn}
     'END_CONNECTIONS'

subapp_data_conn::= subapp_data_source 'TO' subapp_data_destination ';'

subapp_data_source::= ([plug_name '.'] input_variable_name)
     | ((fb_subapp_name | socket_name) '.' output_variable_name)

subapp_data_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_subapp_name | socket_name) '.' input_variable_name)

subapp_type_name::= identifier

subapp                          ifier
```


## 配置元素

根据第7章中给出的规则，本节中定义的语法可用于资源类型、设备类型、段类型、应用程序和系统配置的文本规范。

条款B.2中给出的产生式也适用于本条款。

**SYNTAX:**

```
application_configuration::=
     'APPLICATION' application_name
     [fb_instance_list]
     [subapp_instance_list]
     [subapp_connection_list]
     'END_APPLICATION'

system_configuration::= 'SYSTEM' system_name
      {application_configuration}
      device_configuration
      {device_configuration}
      [mappings]
      [segments]
      [links]
     'END_SYSTEM'

segments::= 'SEGMENTS'
      segment
      {segment}
     'END_SEGMENTS'

segment::= segment_name ':' segment_type_name [parameters] ';'

links::= 'LINKS'
      link
      {link}
     'END_LINKS'
```

```
link::= resource_hierarchy '=>' segment_name [parameters] ';'

parameters::= '(' parameter {',' parameter} ')'

parameter::= parameter_name ':='
     (constant | enumerated_value | array_initialization |
     structure_initialization) ';'
     <as defined in IEC 61131-3>

device_configuration::=
     'DEVICE' device_name ':' device_type_name [parameters]
     [resource_type_list]
     {resource_configuration}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE'

resource_type_list::= 'RESOURCE_TYPES'
     {resource_type_name ';'}
     'END_RESOURCE_TYPES'

resource_configuration::=
     'RESOURCE' resource_instance_name ':' resource_type_name [parameters]
     [fb_type_list]
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

fb_type_list::= 'FB_TYPES' {fb_type_name ';'} 'END_FB_TYPES'

config_connection_list::=
     [config_event_conn_list]
     [config_data_conn_list]
     [config_adapter_conn_list]

config_event_conn_list::= 'EVENT_CONNECTIONS'
     {config_event_conn}
     'END_CONNECTIONS'

config_event_conn::= fb_instance_name '.' event_output_name
     'TO' fb_instance_name '.' event_input_name ';'

config_data_conn_list::= 'DATA_CONNECTIONS'
     {config_data_conn}
     'END_CONNECTIONS'

config_data_conn::=
     (fb_instance_name '.' output_variable_name | input_variable_name)
     'TO'
     (fb_instance_name | resource_instance_name)'.' input_variable_name ';'
     <resource_instance_name only applies to connections within device_type or
     device_configuration declarations>

config_adapter_conn_list::= 'ADAPTER_CONNECTIONS'
     {config_adapter_conn}
     'END_CONNECTIONS'

config_adapter_conn::= fb_instance_name '.' plug_name
     'TO' fb_instance_name '.' socket_name ';'

fb_instance_reference::= [app_hierarchy_name] fb_instance_name

app_hierarchy_name:= application_name '.'{subapp_instance_name '.'}
```



– 70 – 61499-1 © IEC:2012

```
link::= resource_hierarchy '=>' segment_name [parameters] ';'

parameters::= '(' parameter {',' parameter} ')'

para
     （常量枚举_值数组_初始化结构_初始化）';'<如IEC61131-3中所定义>



device_configuration::=
     'DEVICE' device_name ':' device_type_name [parameters]
     [resource_type_list]
     {resource_configuration}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE'

resource_type_list::= 'RESOURCE_TYPES'
     {resource_type_name ';'}
     'END_RESOURCE_TYPES'

resource_configuration::=
     'RESOURCE' resource_instance_name ':' resource_type_name [parameters]
     [fb_type_list]
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

fb_type_list::= 'FB_TYPES' {fb_type_name ';'} 'END_FB_TYPES'

config_connection_list::=
     [config_event_conn_list]
     [config_data_conn_list]
     [config_adapter_conn_list]

config_event_conn_list::= 'EVENT_CONNECTIONS'
     {config_event_conn}
     'END_CONNECTIONS'

config_event_conn::= fb_instance_name '.' event_output_name
     'TO' fb_instance_name '.' event_input_name ';'

config_data_conn_list::= 'DATA_CONNECTIONS'
     {config_data_conn}
     'END_CONNECTIONS'

conf
     (fb_instance_name'.'output_variable_nameinput_variable_name)'TO'(fb_instance_nameresource_instance_
     name)'。输入变量名';'<resource_instance_name仅适用于device_type或device_configuration声明中的连接
     >



config_adapter_conn_list::= 'ADAPTER_CONNECTIONS'
     {config_adapter_conn}
     'END_CONNECTIONS'

config_adapter_conn::= fb_instance_name '.' plug_name
     'TO' fb_instance_name '.' socket_name ';'

fb_instance_reference::= [app_hierarchy_name] fb_instance_name

app_hierarchy_name:= application_name '.'{subapp_instance_name '.'}
```

```
device_type_specification::=
     'DEVICE_TYPE' device_type_name
     [input_variable_list]
     [resource_type_list] <if not given, defined by resource instances>
     {resource_instance}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE_TYPE'

resource_instance::=
     'RESOURCE' resource_instance_name ':' resource_type_name
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

resource_type_specification::= 'RESOURCE_TYPE' resource_type_name
     [input_variable_list]
     [fb_type_list] <if not given, defined by function block instances>
     [fb_instance_list]
     config_connection_list
     'END_RESOURCE_TYPE'

segment_type_specification::= 'SEGMENT_TYPE' segment_type_name
     {parameter_declaration}
     'END_SEGMENT_TYPE'

parameter_declaration:= parameter_name ':' var_spec_init ';'

mappings::= 'MAPPINGS' mapping {mapping} 'END_MAPPINGS'

mapping::= fb_instance_reference 'ON' fb_resource_reference ';'

fb_resource_reference::= resource_hierarchy ['.' fb_instance_name]
     <When the optional element ['.' fb_instance_name] is not given, the
     instance name of the FB in the resource is the same as its instance name
     in the corresponding fb_instance_reference of the mapping.>

resource_hierarchy::= device_name ['.' resource_instance_name]

segment_name::= identifier

segment_type_name::= identifier

parameter_name::= identifier

system_name::= identifier

device_name::= identifier

device_type_name::= identifier

application_name::= identifier

resource_instance_name::= identifier

resource_type_name::= identifier
```



61499-1 © IEC:2012 – 71 –

```
devi
     'DEVICE_TYPE'device_type_name[input_variable_list][resource_type_list]<如果没有给出，由资源
     实例定义>{resource_instance [fb_instance_list][config_connection_list]'END_DEVICE_TYPE'

resource_instance::=
     'RESOURCE' resource_instance_name ':' resource_type_name
     [fb_instance_list]
     [config_connection_list]

resource_type_specification::='RESOURCE_TYPE'resource_type_name[input_variable_list][fb_type_list]<
如果未给出，由功能块实例定义>[fb_instance_list]config_connection_list'END_RESOURCE_TYPE'

segment_type_specification::= 'SEGMENT_TYPE' segment_type_name
     {parameter_declaration}
     'END_SEGMENT_TYPE'

parameter_declaration:= parameter_name ':' var_spec_init ';'

mappings::= 'MAPPINGS' mapping {mapping} 'END_MAPPINGS'

mapping::= fb_instance_reference 'ON' fb_resource_reference ';'

fb_r
     <当可选元素['.'fb_instance_name]没有给出，资源中FB的实例名称与映射对应的fb_instance_reference中的
     实例名称相同。>

resource_hierarchy::= device_name ['.' resource_instance_name]

segment_name::= identifier

segment_type_name::= identifier

parameter_name::= identifier

system_name::= identifier

device_name::= identifier

device_type_name::= identifier

application_name::= identifier

resource_instance_name::= identifier

resource_type_name::= identifier
```

## B.4 Common elements

Where syntactic productions are not given for non-terminal symbols in Annex B, the syntactic productions and corresponding semantics given in Annex B of IEC 61131-3:2003 shall apply.

## B.5 Supporting productions for management commands

The syntax defined in this clause is referenced in Table 8.

**SYNTAX:**

```
data_type_list::= 'DATA_TYPES' {data_type_name ';'} 'END_DATA_TYPES'

connection_definition::=
     connection_start_point ' ' connection_end_point

connection_start_point::= fb_instance_reference '.' attachment_point

connection_end_points::=
     connection_end_point {',' connection_end_point}

connection_end_point::= fb_instance_reference '.' attachment_point

attachment_point::= identifier

referenced_parameter::=
     [(resource_instance_name | fb_instance_name)'.'] parameter
     <resource_instance_name refers to a resource located in the same device
     as the MANAGER block defined in 6.3.2>
     <fb_instance_name refers to an FB contained in the same device or
     resource as the <MANAGER> block>
     <if no resource or FB instance name is given, the parameter refers to a
     parameter of the device or resource containing the MANAGER block>

parameter_reference::=
     [(resource_instance_name | fb_instance_name)'.'] parameter_name
     <see above for semantics>

all_data_types::= 'ALL_DATA_TYPES'

all_fb_types::= 'ALL_FB_TYPES'

fb_status::= 'IDLE' | 'RUNNING' | 'STOPPED' | 'KILLED'
```





### 共同元素

如果附录B中的非终结符号没有给出句法产生式，则应适用IEC61131-3:2003附录B中给出的句法产生式和相应的语义。

### 支持管理命令的产生

本节中定义的语法在表8中引用。

**SYNTAX:**

```
data_type_list::= 'DATA_TYPES' {data_type_name ';'} 'END_DATA_TYPES'

connection_definition::=
     connection_start_point ' ' connection_end_point

connection_start_point::= fb_instance_reference '.' attachment_point

connection_end_points::=
     connection_end_point {',' connection_end_point}

connection_end_point::= fb_instance_reference '.' attachment_point

attachment_point::= identifier

refe
     [(resource_instance_namefb_instance_name)'.']参数<resource_instance_name指与6.3.2中定义的MANAG
     ER块位于同一设备中的资源><fb_instance_name指与<MANAGER>块包含在同一设备或资源中的FB><如果
     没有给出资源或FB实例名称，则参数指的是包含MANAGER块的设备或资源的参数>

para
     [(resource_instance_namefb_instance_name)'.']parameter_name<语义见上文>

all_data_types::= 'ALL_DATA_TYPES'

all_fb_types::= 'ALL_FB_TYPES'

fb_status::= 'IDLE' | 'RUNNING' | 'STOPPED' | 'KILLED'
```

## B.6 Tagged data types

The syntax defined below shall be used for the assignment of tags as defined in ISO/IEC 8824-1 to derived data types defined as specified in Annex B and Annex E. As defined in ISO/IEC 8824-1, the class tags APPLICATION and PRIVATE shall be used except for types to be used only in context-specific tagging.

**SYNTAX:**

```
tagged_type_declaration::=
     'TYPE'
     asn1_tag type_declaration ';'
     {asn1_tag type_declaration ';'}
     'END_TYPE'

asn1_tag::= '[' ['APPLICATION' | 'PRIVATE'] (integer | hex_integer) ']'
```

## B.7 Adapter interface types

See 5.5 for the semantics associated with the following syntax.

**SYNTAX:**

```
adapter_type_declaration::=
     'ADAPTER' adapter_type_name
          fb_interface_list
          [fb_service_declaration]
     'END_ADAPTER'

adapter_type_name::= identifier
```





## 标记数据类型

下面定义的语法应用于将ISOIEC8824-1中定义的标签分配给附录B和附录E中定义的派生数据类型。如ISOIEC8824-1中定义的，类标签APPLICATION和PRIVATE应除了仅在特定于上下文的标记中使用的类型外，都可以使用。

**SYNTAX:**

```
tagged_type_declaration::=
     'TYPE'
     asn1_tag type_declaration ';'
     {asn1_tag type_declaration ';'}
     'END_TYPE'

asn1_t                        ' | 'PRIVATE'] (integer | hex_integer) ']'
```

## 适配器接口类型

有关与以下语法相关的语义，请参见5.5。

**SYNTAX:**

```
adapter_type_declaration::=
     'ADAPTER' adapter_type_name
          fb_interface_list
          [fb_service_declaration]
     'END_ADAPTER'

adapter_type_name::= identifier
```

# Annex C
(informative)

# Object models

## C.1 Model notation

Annex C presents object models for some of the classes which may be used in Engineering Support Systems (ESS) to support the design, implementation, commissioning and operation of Industrial-Process Measurement and Control Systems (IPMCSs) constructed according to the architecture defined in this standard.

The notation used in Annex C is the Unified Modeling Language (UML). References to extensive documentation of this notation can be found on the Internet at the Uniform Resource Locator (URL) **http://www.omg.org/uml/**.

## C.2 ESS models

### C.2.1 ESS overview

Figure C.1 presents an overview of the major classes in the ESS (Engineering Support System) for an industrial-process measurement and control system (IPMCS), and their correspondence to the classes of objects in the IPMCS. Descriptions of the classes in Figure C.1 are given in Table C.1.

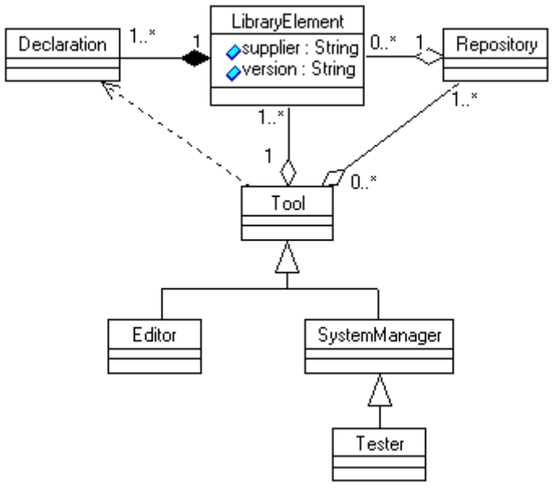



**Figure C.1 – ESS overview**





# 对象模型

## 型号符号

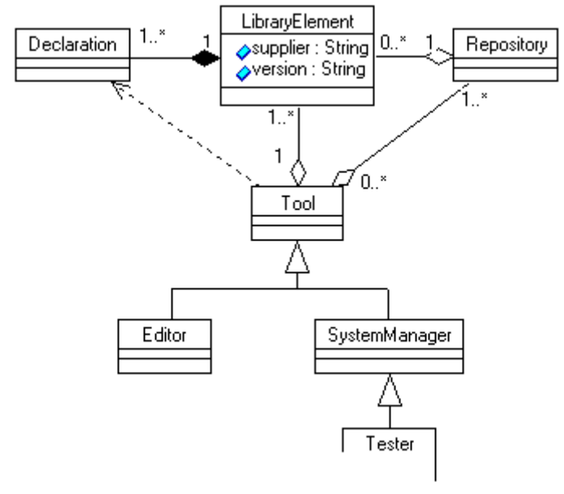
附录C提供了一些可用于工程支持系统(ESS)的类的对象模型，以支持根据本文定义的架构构建的工业过程测量和控制系统(IPMCS)的设计、实施、调试和操作标准。

附录C中使用的符号是统一建模语言(UML)。可以在Internet上的统一资源定位器(URL)http:www.omg.org uml上找到对该表示法的大量文档的引用。

## C.2 ESS型号

### ESS概述

图C.1概述了工业过程测量和控制系统(IPMCS)的ESS（工程支持系统）中的主要类别，以及它们与IPMCS中对象类别的对应关系。图C.1中的类的描述在表C.1中给出。




**图C.1 ESS概览**

**Table C.1 – ESS class descriptions**

| | |
|---|---|
| **Declaration** | This is the abstract superclass for *declarations*. |
| **Editor** | Instances of this class provide the editing functions on *declarations* necessary to support the EDIT use case. |
| **LibraryElement** | This is the abstract superclass of objects which may be stored in repositories and which may be imported and exported in the textual syntax defined in Annex B, or the XML syntax defined in IEC 61499-2. Such objects have supplier (vendor, programmer, etc.) and version(version number, date, etc.) attributes to assist in management, in addition to a name (inherited from **NamedDeclaration –** see C.2.2) as a key attribute. |
| **Repository** | Instances of this class provide persistent storage and retrieval of library elements. They may also provide version control services. |
| **SystemManager** | Instances of this class provide the functions necessary to support the INSTALL and OPERATE use cases. |
| **Tester** | This class extends the capabilities of the SystemManager class to support the operations of the TEST use case. |
| **Tool** | This class models the generic behaviors of *software tools* for engineering support of IPMCSs. |

### C.2.2 Library elements

The subclasses of **LibraryElement** are shown in Figure C.2. The syntactic production in Annex B corresponding to each subclass is listed in Table C.2.

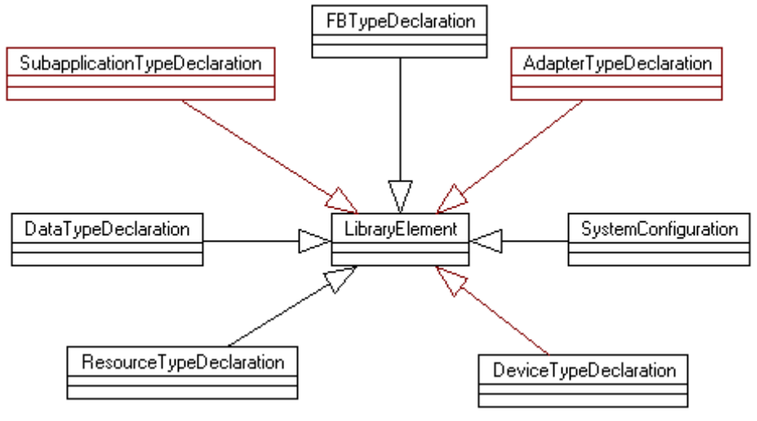



**Figure C.2 – Library elements**

**Table C.2 – Syntactic productions for library elements**

| Class | Syntactic production |
|---|---|
| **DataTypeDeclaration** | type_declaration |
| **FBTypeDeclaration** | fb_type_declaration |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **SubapplicationTypeDeclaration** | subapplication_type_declaration |
| **ResourceTypeDeclaration** | resource_type_specification |
| **DeviceTypeDeclaration** | device_type_specification |
| **SystemConfiguration** | system_configuration |



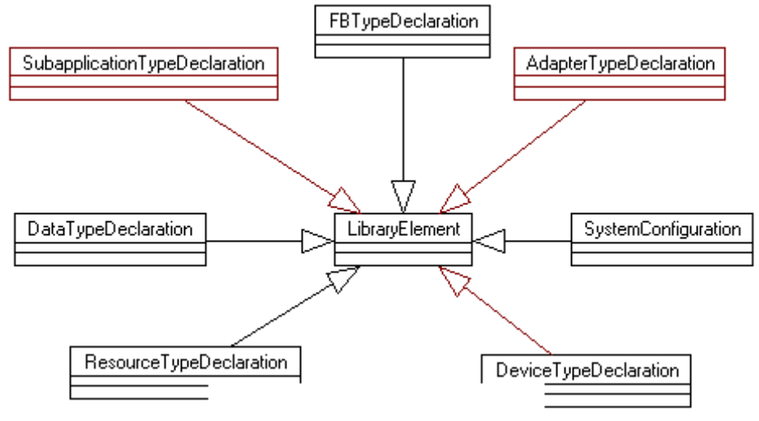


**表C.1 ESS类描述**

| | |
|---|---|
| **Declaration** | 这是声明的抽象超类。 |
| **Editor** | 此类的实例提供了对支持EDIT用例所必需的声明的编辑功能。 |
| **LibraryElement** | 这是对象的抽象超类，可以存储在存储库中，可以按照附件B中定义的文本语法或IEC61499-2中定义的XML语法导入和导出。此类对象具有供应商（供应商、程序员等）和版本（版本号、日期等）属性以协助管　　理，此外还有名称（继承自NamedDeclaration 参见C.2.2）作为关键属性。 |
| **Repository** | 此类的实例提供库元素的持久存储和检索。他们还可能提供版本控制服务。 |
| **SystemManager** | 此类的实例提供了支持INSTALL和<br>操作用例。 |
| **Tester** | 此类扩展了SystemManager类的功能以支持TEST用例的操作。 |
| **Tool** | 此类对用于工程支持的软件工具的通用行为进行建模<br>IPMCSs. |

### 库元素

LibraryElement的子类如图C.2所示。中的句法产生
每个子类对应的附录B列于表C.2中。



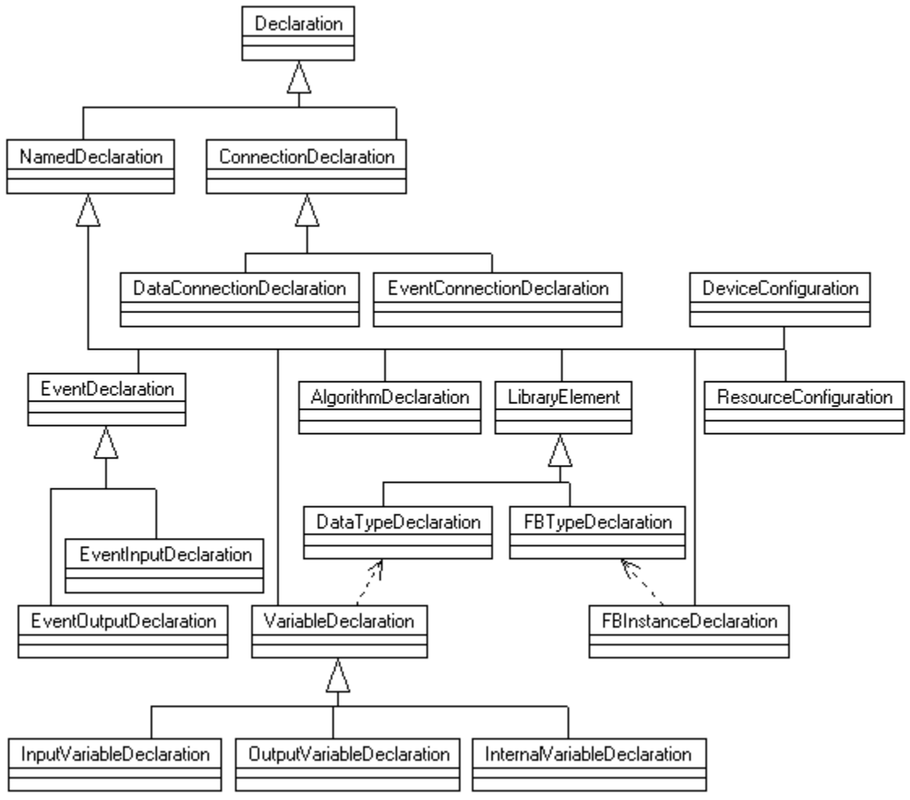

**图C.2 库元素**

**表C.2 库元素的句法产生式**

| Class | 句法产生 |
|---|---|
| **DataTypeDeclaration** | type_declaration |
| **FBTypeDeclaration** | fb_type_declaration |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **SubapplicationTypeDeclaration** | subapplication_type_declaration |
| **ResourceTypeDeclaration** | resource_type_specification |
| **DeviceTypeDeclaration** | device_type_specification |
| **SystemConfiguration** | system_configuration |

### C.2.3 Declarations

Figure C.3 shows the class hierarchy of *declarations* which may be manipulated by *software tools*. The syntactic productions in Annex B corresponding to each of these subclasses are listed in Table C.3.



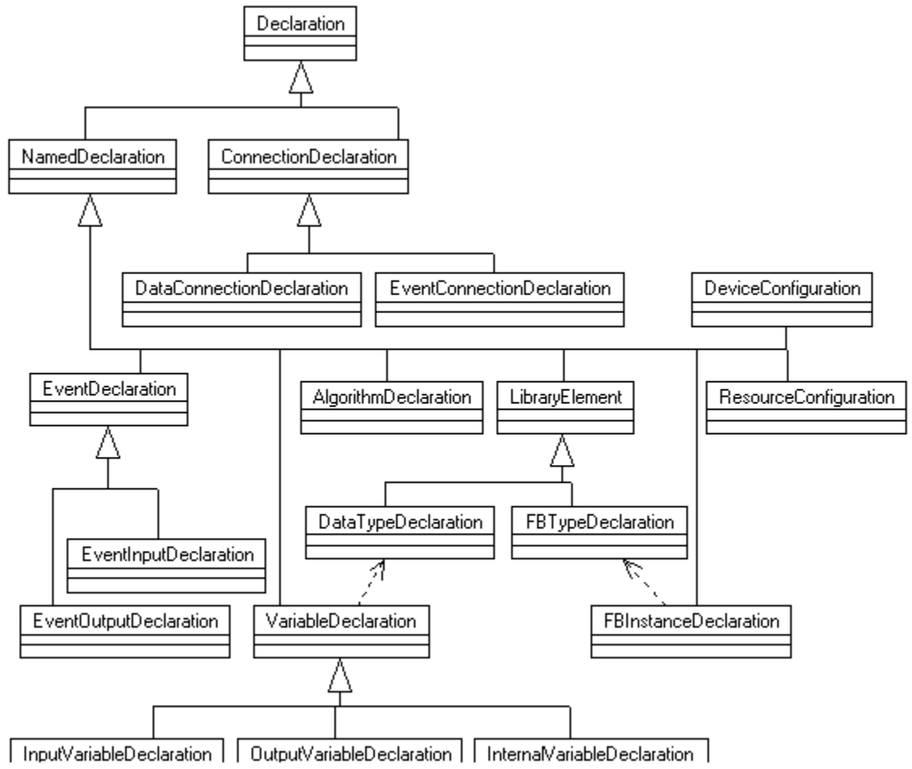

NOTE To avoid clutter, classes related to *adapter types, instances* and *connections* are not shown in this Figure; however, they are listed in Table C.3 for reference.

**Figure C.3 – Declarations**





图C.3显示了可以由软件工具操作的声明的类层次结构。附录B中对应于这些子类的句法产生式列于表C.3中。




注意为避免混乱，与适配器类型、实例和连接相关的类未在此图中显示；但是，它们在表C.3中列出以供参考。

**图C.3 声明**

**Table C.3 – Syntactic productions for declarations**

| Class | Syntactic production |
|---|---|
| **AdapterConnectionDeclaration** | adapter_conn |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **AlgorithmDeclaration** | fb_algorithm_declaration |
| **DataConnectionDeclaration** | data_conn |
| **DeviceConfiguration** | device_configuration |
| **EventConnectionDeclaration** | event_conn |
| **EventInputDeclaration** | event_input_declaration |
| **EventOutputDeclaration** | event_output_declaration |
| **FBInstanceDeclaration** | fb_instance_definition |
| **InputVariableDeclaration** | input_var_declaration |
| **InternalVariableDeclaration** | internal_var_declaration |
| **OutputVariableDeclaration** | output_var_declaration |
| **PlugDeclaration** | Part of plug_list |
| **ResourceConfiguration** | resource_instance |
| **SocketDeclaration** | Part of socket_list |

### C.2.4 Function block network declarations

Figure C.4 shows the relationships among the elements of *function block network declarations*. See C.2.2 for definitions of the aggregated classes in this diagram.

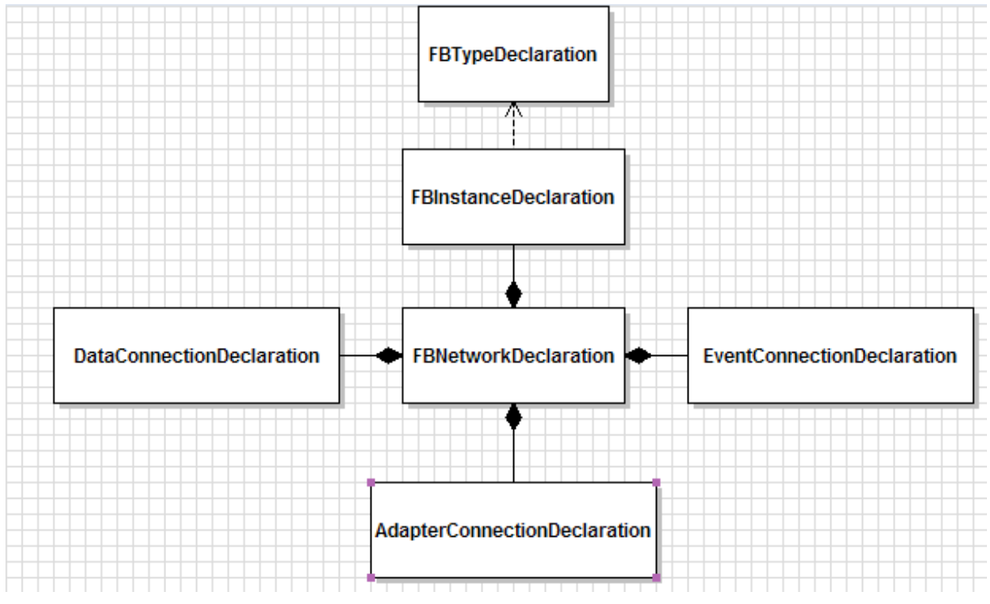



**Figure C.4 – Function block network declarations**



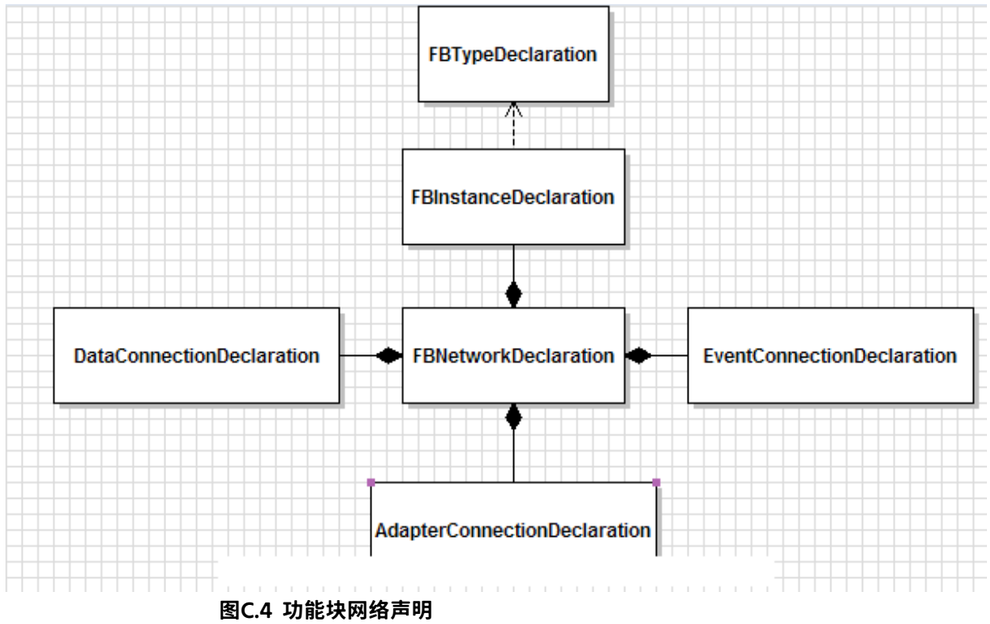


**表C.3 声明的语法产生式**

| Class | 句法产生 |
|---|---|
| **AdapterConnectionDeclaration** | adapter_conn |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **AlgorithmDeclaration** | fb_algorithm_declaration |
| **DataConnectionDeclaration** | data_conn |
| **DeviceConfiguration** | device_configuration |
| **EventConnectionDeclaration** | event_conn |
| **EventInputDeclaration** | event_input_declaration |
| **EventOutputDeclaration** | event_output_declaration |
| **FBInstanceDeclaration** | fb_instance_definition |
| **InputVariableDeclaration** | input_var_declaration |
| **InternalVariableDeclaration** | internal_var_declaration |
| **OutputVariableDeclaration** | ration |
| **PlugDeclaration** | plug_list的一部分 |
| **ResourceConfiguration** | |
| **SocketDeclaration** | socket_list的一部分 |

### 功能块网络声明

图C.4显示了功能块网络声明的元素之间的关系。有关此图中聚合类的定义，请参见C.2.2。




**图C.4 功能块网络声明**

### C.2.5 Function block type declarations

Figure C.5 shows the relationships among the elements of *function block type declarations*. Syntactic productions for the classes **EventInputDeclaration, EventOutputDeclaration, InputVariableDeclaration, OutputVariableDeclaration, InternalVariableDeclaration,** and the component classes of **FBNetworkDeclaration** are given in Table C.3. The syntactic productions `fb_ecc_declaration` and `fb_service_declaration` in Clause B.2 correspond to classes **ECCDeclaration** and **ServiceDeclaration**, respectively.

NOTE 1 Declarations of *subapplications* are represented by instances of the class **CompositeFBTypeDeclaration** which contain no event `WITH` data associations.

NOTE 2 **NamedDeclaration** is the abstract superclass of declarations which have names, e.g., *type names* or *instance names*.

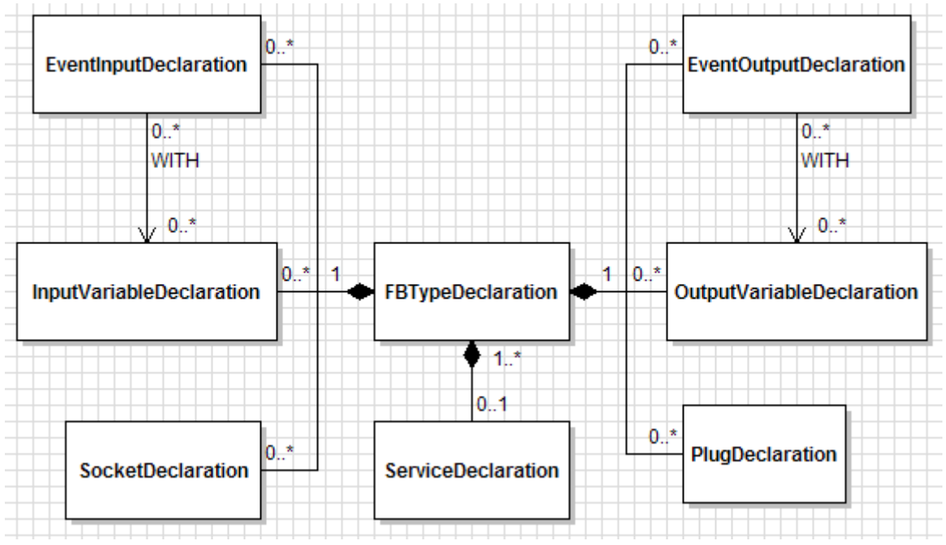



**Figure C.5a – Composition**





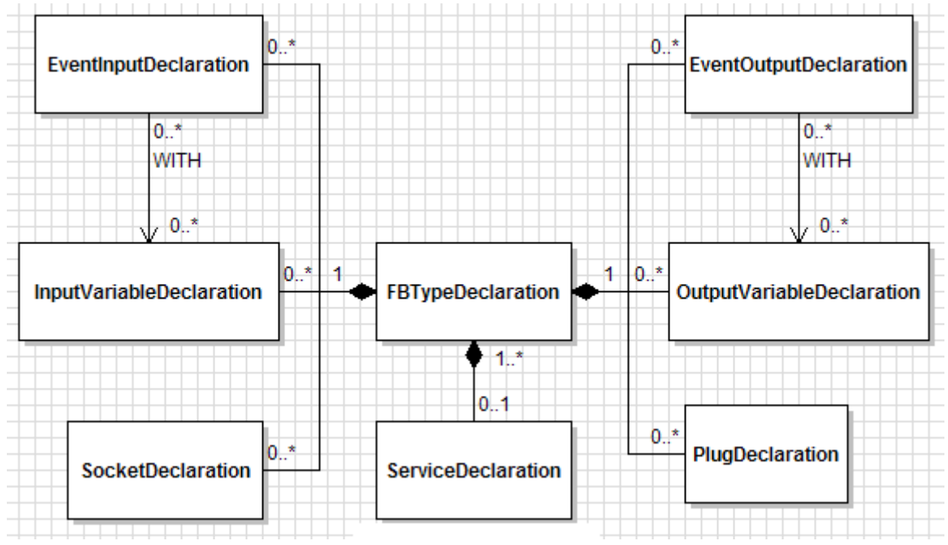
### 功能块类型声明

**图C.5显示了功能块类型声明的元素之间的关系。表C.3给出了EventInputDeclaration、EventOutputDeclaration、InputVariableDeclaration、OutputVariableDeclaration、InternalVariableDeclaration类和FBNetworkDeclaration的组件类的语法产生式。句法产生**

**分别对应ECCDeclaration和ServiceDeclaration类。**

注1声明
**CompositeFBTypeDeclaration不包含任何事件WITH数据关联。**

**注2NamedDeclaration是具有名称的声明的抽象超类，例如，类型名称或实例名称。**




**图C.5a 组成**

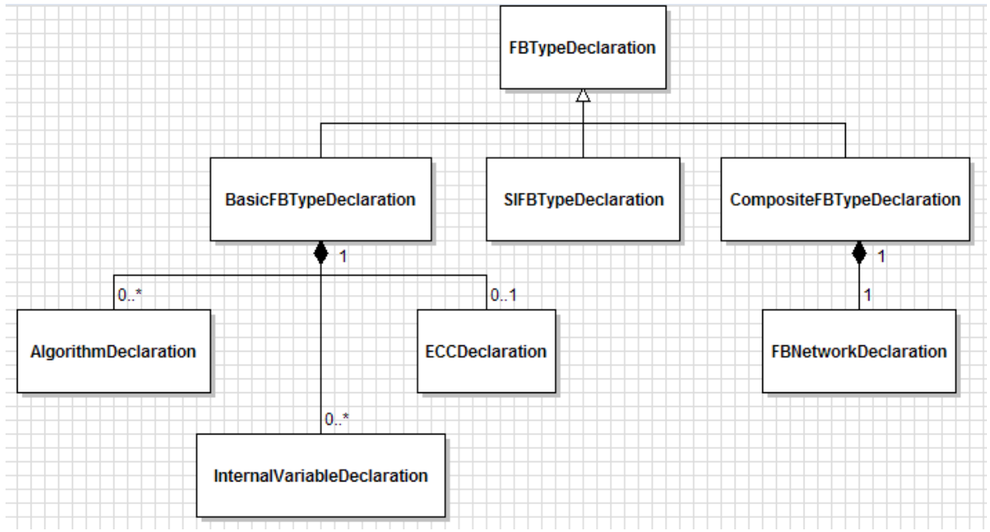



**Figure C.5b – Class hierarchy**

**Figure C.5 – Function block type declarations**

## C.3 IPMCS models

Figure C.6 presents an overview of the major classes in the industrial-process measurement and control system (IPMCS). Descriptions of the classes in Figure C.6 and their corresponding objects in the Engineering Support System (ESS) are given in Table C.4.

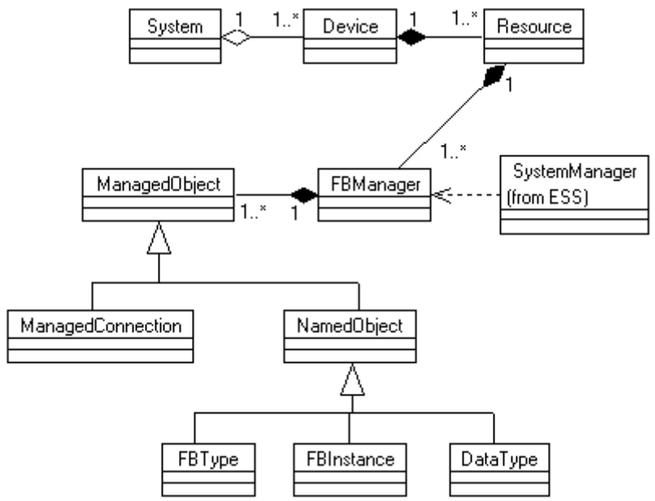
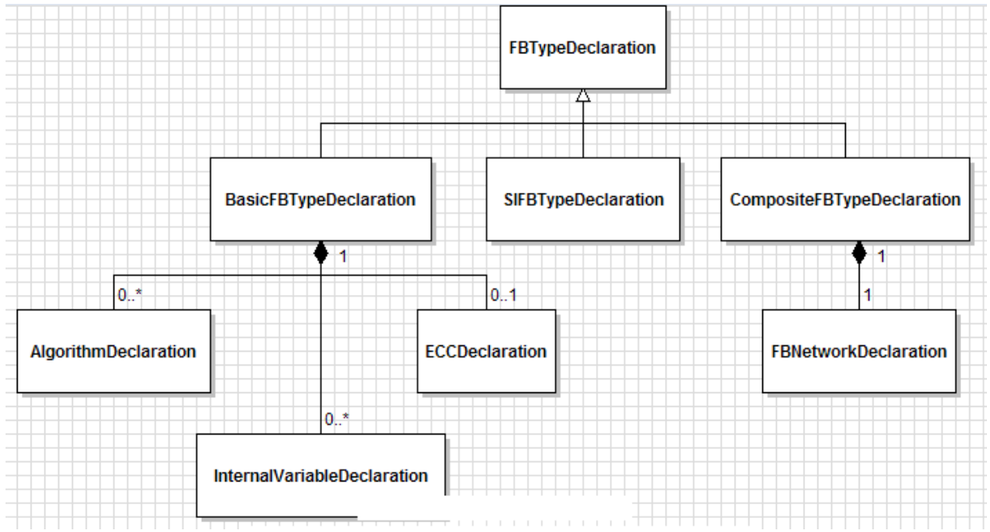



**Figure C.6 – IPMCS overview**



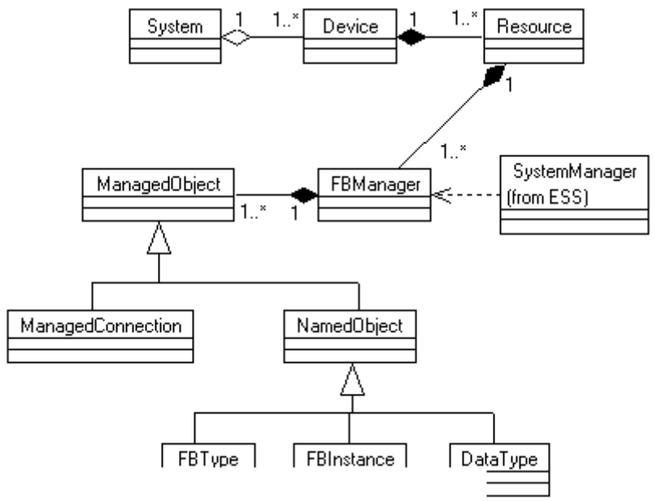





**图C.5b 类层次结构**

**图C.5 功能块类型声明**

图C.6概述了工业过程测量和控制系统(IPMCS)中的主要类别。图C.6中的类及其在工程支持系统（ESS）中的相应对象的描述在表C.4中给出。




**图C.6 IPMCS概述**

**Table C.4 – IPMCS classes**

| IPMCS class | Description | Corresponding ESS class |
|---|---|---|
| **DataType** | An instance of this class is a *data type*. | **DataTypeDeclaration** |
| **Device** | An instance of this class represents a *device*. | **DeviceConfiguration** |
| **FBInstance** | An instance of this class is a *function block instance*. | **FBInstanceDeclaration** |
| **FBManager** | An instance of this class provides the management services defined in Clause 6. | **SystemManager** |
| **FBType** | An instance of this class is a *function block type*. | **FBTypeDeclaration** |
| **ManagedConnection** | Instances of this class can be accessed by an instance of the **FBManager** class using the source and destination combination as a unique key. | **ConnectionDeclaration** |
| **ManagedObject** | This is the abstract superclass of objects which are managed by an instance of the **FBManager** class. Such objects may have supplier (vendor, programmer, etc.) and version (version number, date, etc.) attributes to assist in management. | none |
| **NamedObject** | This is the abstract superclass of objects which can be accessed by name by an instance of the **FBManager** class. | **NamedDeclaration** |
| **Resource** | An instance of this class represents a *resource*. | **ResourceConfiguration** |
| **System** | An instance of this class represents an Industrial-Process Measurement and Control System (IPMCS). | **SystemConfiguration** |

Figure C.7 shows the relationships among the elements of a *function block instance* and its associated *function block type*.





**表C.4 IPMCS类**

| IPMCS class | | 对应的ESS类 |
|---|---|---|
| **DataType** | 此类的一个实例是一种数据类型。 | **DataTypeDeclaration** |
| **Device** | 此类的一个实例代表一个设备。 | **DeviceConfiguration** |
| **FBInstance** | 此类的一个实例是一个功能块实例。 | **FBInstanceDeclaration** |
| **FBManager** | 此类的一个实例提供第6章中定义的管理服务。 | **SystemManager** |
| **FBType** | 此类的一个实例是功能块类型。 | **FBTypeDeclaration** |
| **ManagedConnection** | FBManager类的实例可以使用源和目标组合作为唯一键来访问此类的实例。 | **ConnectionDeclaration** |
| **ManagedObject** | 这是由FBManager类的实例管理的对象的抽象超类。此类对象可能具有供应商（供应商、程序员等）和版本（版本号、日期等）属性以协助管　　理。 | none |
| **NamedObject** | 这是对象的抽象超类，可以通过FBManager类的实例按名称访问。 | **NamedDeclaration** |
| **Resource** | 此类的一个实例代表一个资源。 | **ResourceConfiguration** |
| **System** | 这个类的一个实例代表一个工业过程测量和控制System (IPMCS). | **SystemConfiguration** |

图C.7显示了功能块实例的元素及其相关的功能块类型之间的关系。

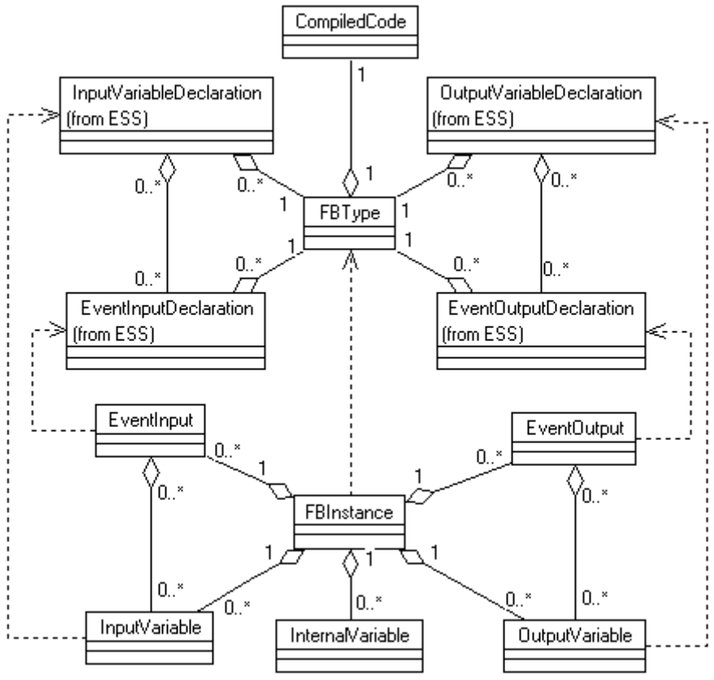
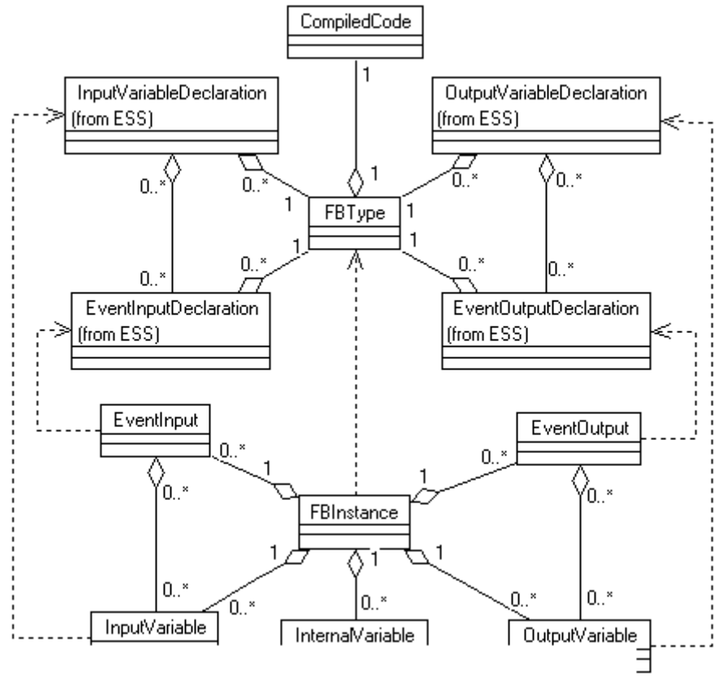



**Figure C.7 – Function block types and instances**








**图C.7 功能块类型和实例**

# Annex D
(informative)

# Relationship to IEC 61131-3

## D.1 General

*Functions* and *function blocks* as defined in IEC 61131-3 can be used for the *declaration* of *algorithms* in *basic function block types* as specified in 5.2.1. Clause D.2 defines rules for the conversion of IEC 61131-3 *functions* and *function block types* into *simple function block types* so that they can be used in the specification of *applications* and *resource types.* Clause D.3 defines event-driven versions of IEC 61131-3 functions and function blocks for the same uses.

## D.2 "Simple" function blocks

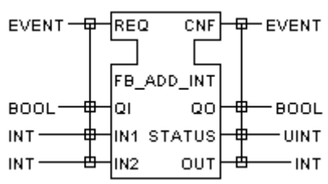
As illustrated in Figure D.1, IEC 61131-3 functions and function blocks can be converted to "simple" function blocks according to the following rules:

a) Simple function blocks are represented as *service interface function blocks* for application-initiated interactions as shown in Figure 21a.

b) The *type name* of the simple function block type is the name of the converted IEC 61131-3 function or function block type with the prefix `FB_` (for instance, `FB_ADD_INT` in Figure D.1). The prefix `F_` instead of `FB_` may optionally be used for simple function block types that are the result of conversions of IEC 61131-3 *functions*.

c) The *input* and *output variables* and their corresponding *data types* are the same as the corresponding input and output variables of the converted IEC 61131-3 function or function block type.

d) The `INIT` event input and `INITO` event output are used with simple function block types that have been converted from IEC 61131-3 *function block types*, and are not used with simple function block types that have been converted from IEC 61131-3 *functions*.




NOTE A complete textual declaration of this function block type is given in Annex F.

**Figure D.1 – Example of a "simple" function block type**

The behavior of *instances* of simple function block *types* is according to the following rules:

e) Initialization is as specified in 2.4.2 of IEC 61131-3:2003 for *variables*, and as specified in 2.6 of IEC 61131-3:2003 for Sequential Function Chart (SFC) elements.

f) The occurrence of an INIT+ service primitive is equivalent to "cold restart" initialization as defined in the above mentioned subclauses of IEC 61131-3:2003, followed by an INITO+ service primitive with a STATUS value of zero (0).

g) The occurrence of an INIT- or REQ- service primitive has no effect except to cause an INITO- or CNF-service primitive, respectively, with a STATUS value of one (1).





# 与IEC61131-3的关系

IEC61131-3中定义的功能和功能块可用于在5.2.1中指定的基本功能块类型中的算法声明。条款D.2定义了将IEC61131-3功能和功能块类型转换为简单功能块类型的规则，以便它们可以在应用程序和资源类型的规范中使用。条款D.3定义了事件驱动版本的IEC61131-3功能和用于相同用途的功能块。

## "简单"功能块

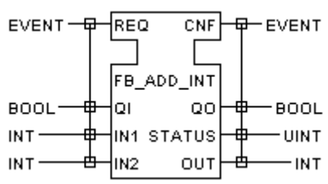
如图D.1所示，IEC61131-3功能和功能块可以根据以下规则转换为"简单"功能块：

a)简单功能块被表示为应用程序启动交互的服务接口功能块，如图21a所示。

b)简单功能块类型的类型名称是转换后的IEC61131-3功能或功能块类型的名称，前缀为FB_（例如，图D.1中的FB_ADD_INT）。前缀F_而不是FB_可以选择用于作为IEC61131-3功能转换结果的简单功能块类型。

c)输入和输出变量及其对应的数据类型与转换后的IEC61131-3功能或功能块类型对应的输入和输出变量相同。

d)INIT事件输入和INITO事件输出用于从IEC61131-3功能块类型转换而来的简单功能块类型，不用于从IEC61131-3功能转换而来的简单功能块类型。




注：附录F中给出了该功能块类型的完整文本声明。

**图D.1 "简单"功能块类型的示例**

简单功能块类型实例的行为遵循以下规则：

e)初始化在IEC61131-3:2003的2.4.2中对变量进行了规定，在IEC61131-3:2003的2.6中对顺序功能图(SFC)元素进行了规定。

f) INIT+服务原语的出现等同于IEC61131-3:2003上述子条款中定义的"冷重启"初始化，然后是状态值为零(0)的INITO+服务原语。

g)INITorREQservice原语的出现除了分别引起一个具有一(1)的STATUS值的INITO或CNF-service原语外没有任何影响。

h) The occurrence of a REQ+ service primitive causes the *execution* of the *algorithm* specified in the function block body, according to the rules given in IEC 61131-3 for the language in which the algorithm is programmed.

i) Successful execution of the algorithm in response to a REQ+ primitive results in a CNF+ primitive with a STATUS value of zero (0).

j) If an error occurs during the execution of the algorithm, the result is a CNF- primitive with a STATUS value determined according to Table D.1.

**Table D.1 – Semantics of STATUS values**

| Value | Semantics |
|---|---|
| **0** | Normal operation |
| **1** | `INIT-` or `REQ-` propagation |
| **2** | Type conversion error |
| **3** | Numerical result exceeds range for data type |
| **4** | Division by zero |
| **5** | Selector (K) out of range for `MUX` function |
| **6** | Invalid character position specified |
| **7** | Result exceeds maximum string length |
| **8** | Simultaneously true, non-prioritized transitions in a selection divergence |
| **9** | Action control contention error |
| **10** | Return from function without value assigned |
| **11** | Iteration fails to terminate |
| **12** | Invalid subscript value |
| **13** | Array size error |

## D.3 Event-driven functions and function blocks

IEC 61131-3 *functions* can be converted into function blocks for efficient use in event-driven systems according to the rules given in Clause D.2 with the following modifications:

a) the *type name* of the event-driven function block type is the same as the name of the converted IEC 61131-3 function with the additional prefix `E_`, e.g., `E_ADD_INT`;

b) a `CNF+` or `CNF-` primitive does not follow execution of the algorithm unless such execution results in a changed value of the function output.

NOTE If "daisy-chaining" of `CNF` outputs to `REQ` inputs is used to implement a sequence of calculations, then the sequence will stop at the first point where an output value does not change.

In general, since IEC 61131-3 *function blocks* have internal state information, such blocks shall be specially converted for use in event-driven systems. For instance, the `E_DELAY` function block shown in Table A.1 can be used for many of the delay functions provided by the timer function blocks in IEC 61131-3. An example of a conversion of the standard IEC 61131-3 `CTU` function block is given as Feature 18 of Table A.1.

## D.4 Compliance with IEC 61131-3

Implementations of this standard shall comply with the requirements of the subclauses 1.5.1, 2.1, 2.2, 2.3 and 2.4 of IEC 61131-3:2003, and the associated elements of Annex B of IEC 61131-3:2003 for the syntax and semantics of textual representation of common elements, with the exceptions and extensions noted in Clause D.5.



h)根据IEC61131-3中给出的算法编程语言规则，REQ+服务原语的出现导致功能块主体中指定的算法的执行。

i) 响应REQ+原语的算法的成功执行导致CNF+原语的STATUS值为零(0)。

j) 如果在算法执行期间发生错误，则结果是一个CNFprimitive，其STATUS值根据表D.1确定。

**表D.1 STATUS值的语义**

| Value | Semantics |
|---|---|
| **0** | 普通手术 |
| **1** | INIT或REQ传播 |
| **2** | 类型转换错误 |
| **3** | 数值结果超出数据类型的范围 |
| **4** | 被零除 |
| **5** | 选择器(K)超出MUX功能范围 |
| **6** | 指定的字符位置无效 |
| **7** | 结果超过最大字符串长度 |
| **8** | 选择分歧中的同时真实的、非优先级的转换 |
| **9** | 动作控制争用错误 |
| **10** | 从没有赋值的函数返回 |
| **11** | 迭代未能终止 |
| **12** | 无效的下标值 |
| **13** | 数组大小错误 |

## 事件驱动功能和功能块

IEC61131-3功能可以根据D.2中给出的规则转换为功能块，以便在事件驱动系统中有效使用，并进行以下修改：

a)事件驱动功能块类型的类型名称与转换后的IEC61131-3功能的名称相同，带有附加前缀E_，例如E_ADD_INT；

b)CNF+或CNFprimitive不跟随算法的执行，除非这种执行导致函数输出的值发生变化。

注意如果CNF输出到REQ输入的"菊花链"用于实现一系列计算，那么该序列将在输出值不改变的第一个点停止。

通常，由于IEC61131-3功能块具有内部状态信息，因此应专门转换此类块以用于事件驱动系统。例如，表A.1中所示的E_DELAY功能块可用于IEC61131-3中定时器功能块提供的许多延迟功能。标准IEC61131-3CTU功能块的转换示例在表A.1的特征18中给出。

## 符合IEC61131-3

本标准的实施应符合IEC61131-3:2003的1.5.1、2.1、2.2、2.3和2.4小节以及IEC61131-3:2003附录B的相关元素对语法和语义的要求通用元素的文本表示，但第D.5条中指出的例外和扩展。

Where syntactic productions are not given for non-terminal symbols in Annex B, the corresponding syntactic productions given in Annex B of IEC 61131-3:2003 shall apply.

## D.5 Exceptions

Implementations of this standard shall **not** utilize the *directly represented variable* notation defined in 2.4.1.1 of IEC 61131-3:2003 and related features in other subclauses. However, a *literal* of `STRING` or `WSTRING` type, containing a string whose syntax and semantics correspond to the directly represented variable notation, may be used as a *parameter* of a *service interface function block* which provides access to the corresponding variable.

## D.6 Interoperation with programmable controllers

### D.6.1 Overview

A programmable controller may act as a *server,* as defined in IEC 61131-5, to a *device* as defined in this standard, acting as a *client* as defined in IEC 61131-5. These services are provided using the means defined in IEC 61131-5, and are accessed from the IEC 61499 device using instances of the *function block types* specified in Annex D. These function block types are modeled as *communication function block types* as defined in this standard.

The IEC 61499 client device may exist on a communication network along with the programmable controller acting as a server, or may be an implementer-specific subsystem within the "main processing unit" of the programmable controller, as illustrated in Figure 4 of IEC 61131-5:2000. In either case, the interaction between the IEC 61499 client device and the main processing unit is modelled as occurring over one or more *communication connections* as defined in IEC 61499-1, utilizing instances of the function block types defined in Annex D.

### D.6.2 Service conventions

Except for the extensions defined in Annex D, the conventions for naming of input and output variables and events, and for describing the *services* (as defined in this standard) provided by instances of the function block types described in Annex D, are as defined in IEC 61499-1 for the descriptions of *service interface function block types* and *communication function block types*.

For the purposes of Annex D, the `PARAMS` input of type `ANY` defined in this standard is replaced by an `ID` input of type `WSTRING`. The contents of this string specify an **implementation-dependent** representation of the path to the *variable* of interest in the server.

EXAMPLE 1 In the case where the IEC 61499 client device is in logical proximity to the IEC 61131 server, it may be sufficient to simply name the IEC 61131-3 *access path* to the desired variable in the `ID` input, for instance "`CELL_1.CHARLIE`" in the example shown in Figure 19a of IEC 61131-3:2003.

EXAMPLE 2 In the case where the IEC 61499 client device is remotely connected to the IEC 61131-3 server via a communication network, it may be possible to use the `ID` input to encapsulate a Universal Resource Identifier (URI) to specify the desired access path, for instance, "`http://192.168.0.1:61131/CELL_1.CHARLIE`".

NOTE Where supported by an implementation, the `ID` input may specify an access path to a status variable, such as the pre-defined access paths `P_PCSTATUS` and `P_PCSTATE` specified in IEC 61131-5.

Where used, the contents of the `TYPE` input of a function block type defined in Annex D specify the name of the *data type* of the data (`SD` or `RD`) being transferred. This may be the name of an elementary data type such as "`BOOL`" or a derived data type such as "`ANALOG_16_INPUT_DATA`".



如果附录B中的非终结符号没有给出句法产生式，则应适用IEC61131-3:2003附录B中给出的相应句法产生式。

本标准的实施不应使用IEC61131-3:2003的2.4.1.1中定义的直接表示的变量符号以及其他子条款中的相关特征。然而，包含语法和语义对应于直接表示的变量表示法的字符串的STRING或WSTRING类型的文字可以用作提供对相应变量的访问的服务接口功能块的参数。

## D.6 与可编程控制器的互操作

可编程控制器可以充当IEC61131-5中定义的服务器，连接到本标准中定义的设备，充当IEC61131-5中定义的客户端。这些服务是使用IEC61131-5中定义的方式提供的，并使用附件D中指定的功能块类型的实例从IEC61499设备访问。这些功能块类型被建模为本标准中定义的通信功能块类型.

IEC61499客户端设备可以与充当服务器的可编程控制器一起存在于通信网络上，或者可以是可编程控制器的"主处理单元"内的特定于实施者的子系统，如IEC61131的图4所示-5:2000。在任一情况下，IEC61499客户端设备和主处理单元之间的交互被建模为发生在IEC61499-1中定义的一个或多个通信连接上，利用附件D中定义的功能块类型的实例。

### 服务约定

除附录D中定义的扩展外，输入和输出变量和事件的命名约定，以及用于描述由附录D中描述的功能块类型的实例提供的服务（如本标准中定义的）的约定，在附录D中定义IEC61499-1用于描述服务接口功能块类型和通信功能块类型。

出于附录D的目的，本标准中定义的ANY类型的PARAMS输入被WSTRING类型的ID输入替换。此字符串的内容指定服务器中感兴趣的变量的路径的实现相关表示。

示例1在IEC61499客户端设备与IEC61131服务器逻辑接近的情况下，只需将IEC61131-3访问路径命名为ID输入中的所需变量就足够了，例如"CELL_1.CHARLIE"在IEC61131-3:2003的图19a所示的示例中。

示例2在IEC61499客户端设备通过通信网络远程连接到IEC61131-3服务器的情况下，可以使用ID输入来封装通用资源标识符(URI)以指定所需的访问路径，例如，"http:192.168.0.1:61131CELL_1.CHARLIE"。

注：在实现支持的情况下，ID输入可以指定状态变量的访问路径，例如IEC61131-5中指定的预定义访问路径P_PCSTATUS和P_PCSTATE。

在使用时，附件D中定义的功能块类型的TYPE输入的内容指定正在传输的数据（SD或RD）的数据类型的名称。这可能是基本数据类型的名称，例如"BOOL"或派生数据类型，例如"ANALOG_16_INPUT_DATA"。

Where used, the contents of the TASK input of a function block type defined in Annex D specify an **implementation-dependent** representation of the path to the *task* of interest in the server.

EXAMPLE 3 In the case where an IEC 61499 client device is in logical proximity to an IEC 61131-3 server configured as shown in Figure 19a of IEC 61131-3:2003, a path to the task named SLOW_1 in resource STATION_1 could be represented as "CELL_1.STATION_1.SLOW_1".

Values of the STATUS output of the function block types defined in D.6.3 are as given in Table 24 of IEC 61131-5:2000.

### D.6.3 Function block types

#### D.6.3.1 READ

An instance of the READ function block type shown graphically in Figure D.2 and textually in Table D.2 can be used by an IEC 61499 client device to read program or status variable values from an IEC 61131-3 server.

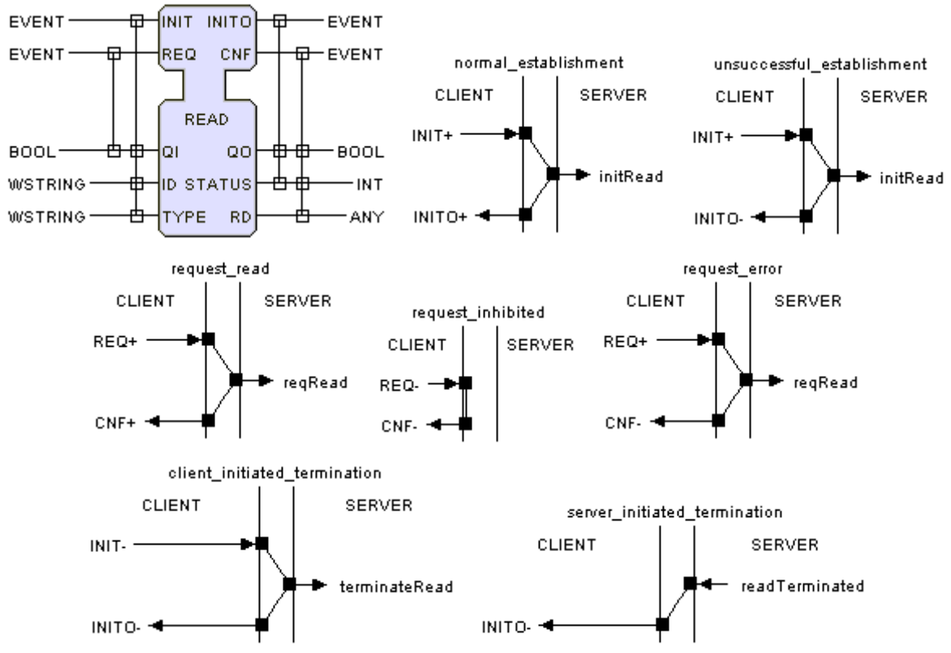



**Figure D.2 – Function block type READ**



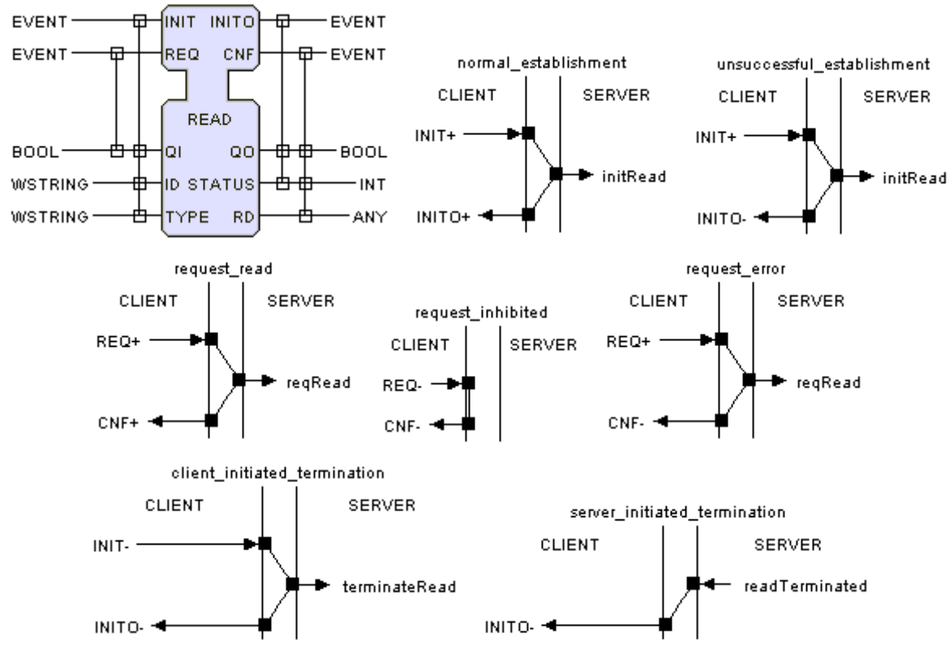
在使用时，附件D中定义的功能块类型的TASK输入的内容指定了服务器中感兴趣的任务路径的依赖于实现的表示。

示例3在IEC61499客户端设备与IEC61131-3服务器逻辑接近的情况下，如IEC61131-3:2003的图19a所示配置，资源STATION_1中名为SLOW_1的任务的路径可以表示为"CELL_1.STATION_1.SLOW_1"。

D.6.3中定义的功能块类型的STATUS输出值在
IEC61131-5:2000的表24。

### D.6.3 功能块类型

IEC61499客户端设备可以使用图D.2中图形显示和表D.2文本显示的READ功能块类型的实例从IEC61131-3服务器读取程序或状态变量值。




**图D.2 功能块类型READ**

**Table D.2 – Source code of function block type READ**

```
FUNCTION_BLOCK READ (* Read server status or program variable *)
EVENT_INPUT
  INIT WITH QI,ID,TYPE; (* Initialize/Terminate Service *)
  REQ WITH QI; (* Service Request *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
  CNF WITH QO,STATUS,RD; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Event Input Qualifier *)
  ID: WSTRING; (* Path to variable to be read *)
  TYPE: WSTRING; (* Data type of RD variable *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
  RD: ANY; (* Variable data from IEC 61131 device *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_read
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF+(RD);
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateRead(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.readTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.2 UREAD

An instance of the UREAD function block type shown graphically in Figure D.3 and textually in Table D.3 can be used by an IEC 61499 client device to request asynchronous notification of a change in value of a program or status variable from an IEC 61131-3 server. Notification is received via the block's IND event output upon completion of the execution of the specified task when a change in the value of the specified variable (with respect to its value upon initiation of task execution) is detected.

An instance of this function block type can also be used to receive notification of the completion of each execution of the specified task by leaving unspecified the ID and TYPE inputs of the block.



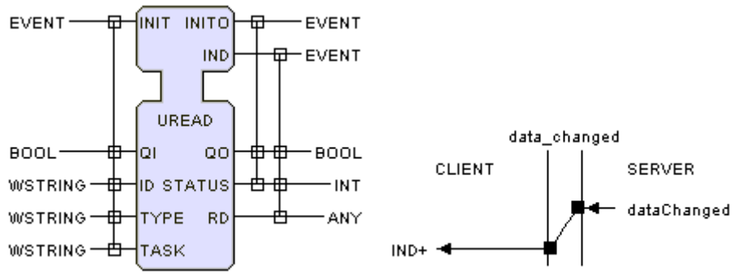


**表D.2 功能块类型READ的源代码**

```
FUNCTION_BLOCKREAD(*读取服务器状态或程序变量*)

用QI ID TYPE初始化；(*初始化终止服务*)
要求QI； (*服务请求*)
END_EVENT


以QO、状态初始化；(*初始化终止确认*)
具有QO、状态、RD的CNF；(*请求服务的确认*)
END_EVENT


问：布尔值；(*事件输入限定符*)
编号：WSTRING；(*要读取的变量路径*)
类型：WSTRING；(*RD变量的数据类型*)
END_VAR


                                                    on *)
  STATUS：INT;
RD：任何；(*来自IEC61131设备的可变数据*)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_read
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF+(RD);
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateRead(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.readTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

UREAD功能块类型的实例如图D.3所示，文本显示在表D.3中，可由IEC61499客户端设备用于请求来自IEC 61131的程序或状态变量值变化的异步通知-3服务器。当检测到指定变量的值（相对于启动任务执行时的值）发生变化时，在完成指定任务的执行时通过块的IND事件输出接收通知。

此功能块类型的实例还可用于接收指定任务的每次执行完成的通知，方法是不指定块的ID和TYPE输入。




NOTE The graphical representation of other service sequences listed in Table D.3 is similar to Figure D.2.

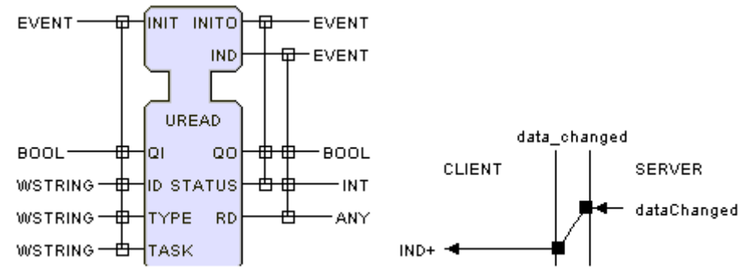
**Figure D.3 – Function block type UREAD**

**Table D.3 – Source code of function block type UREAD**

```
FUNCTION_BLOCK UREAD (* Unsolicited read of IEC 61131 program or status variable *)

EVENT_INPUT
  INIT WITH QI,ID,TASK,TYPE; (* Initialize/Terminate Service *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
  IND WITH QO,STATUS,RD; (* Indication of changed RD value *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Event Input Qualifier *)
  ID: WSTRING; (* Path to variable to be read *)
  TYPE: WSTRING; (* Data type of RD variable *)
  TASK: WSTRING; (* Path to IEC 61131 TASK triggering read on changed value *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
  RD: ANY; (* Input data from resource *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK)
    -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE data_changed
  SERVER.dataChanged() -> CLIENT.IND+(RD);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateURead() -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.UReadTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```



61499-1 © IEC:2012 – 87 –




注：表D.3中列出的其他服务序列的图形表示类似于图D.2。

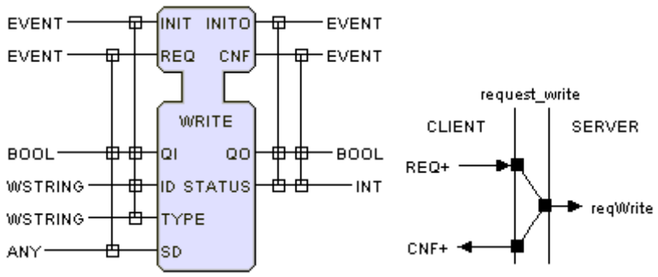
**图D.3 功能块类型UREAD**

**表D.3 功能块类型UREAD的源代码**

```
FUNCTION_BLOCKUREAD(*主动读取IEC61131程序或状态变量*)

EVENT_INPUT
用QI ID TASK TYPE初始化；(*初始化终止服务*)



以QO、状态初始化；(*初始化终止确认*)
IND与QO、状态、RD；(*指示更改的RD值*)



问：布尔值；(*事件输入限定符*)
编号：WSTRING；(*要读取的变量路径*)
类型：WSTRING；(*RD变量的数据类型*)
任务：WSTRING；  (*IEC61131TASK触发读取更改值的路径*)
END_VAR


                                              mal operation *)
  STATUS: INT;
RD: 任何；(*从资源输入数据*)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK)
    -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE data_changed
  SERVER.dataChanged() -> CLIENT.IND+(RD);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateURead() -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.UReadTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.3 WRITE

An instance of the WRITE function block type shown graphically in Figure D.4 and textually in Table D.4 can be used by an IEC 61499 client device to write variable data values to an IEC 61131-3 server.



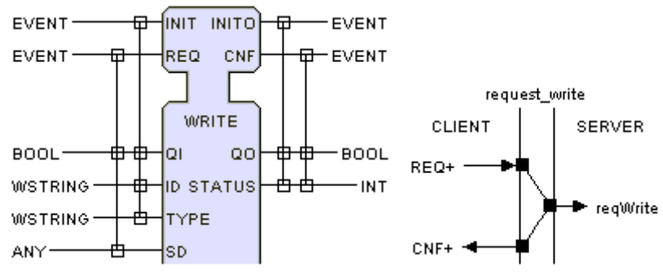

NOTE The graphical representation of other service sequences listed in Table D.4 is similar to Figure D.2.

**Figure D.4 – Function block type WRITE**





IEC61499客户端设备可以使用图D.4中图形显示和表D.4文本显示的WRITE功能块类型的实例将可变数据值写入
IEC 61131-3 server.




注：表D.4中列出的其他服务序列的图形表示类似于图D.2。

**图D.4 功能块类型WRITE**

**Table D.4 – Source code of function block type WRITE**

```
FUNCTION_BLOCK WRITE (* Write a variable value to an IEC 61131 server *)
EVENT_INPUT
  INIT WITH QI,ID,TYPE; (* Initialize/Terminate Service *)
  REQ WITH QI,SD; (* Service Request *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
  CNF WITH QO,STATUS; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Event Input Qualifier *)
  ID: WSTRING; (* Path to variable to be written *)
  TYPE: WSTRING; (* Data type of SD variable *)
  SD: ANY; (* Variable value to write *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_write
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateWrite(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.writeTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.4 TASK

An instance of the TASK function block type shown graphically in Figure D.5 and textually in Table D.5 can be used by an IEC 61499 client device to request the execution of a task on an IEC 61131-3 server.

When an implementation supports this feature, no value is configured for either the SINGLE or INTERVAL input of the corresponding TASK block as defined in Table 50 of IEC 61131-3:2003; rather, execution of the corresponding task is triggered as shown in the request_task service sequence shown in Figure D.5.



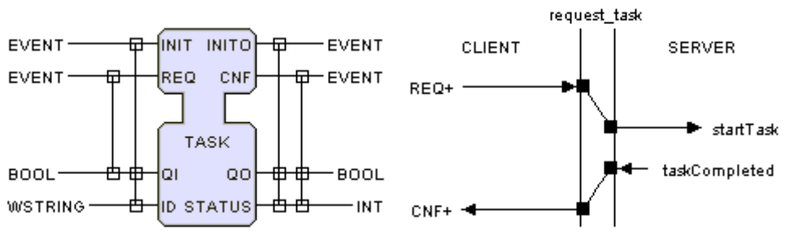


**表D.4 功能块类型WRITE的源代码**

```
FUNCTION_BLOCKWRITE(*将变量值写入IEC61131服务器*)

用QI ID TYPE初始化；(*初始化终止服务*)
请求QI SD；(*服务请求*)



以QO、状态初始化；(*初始化终止确认*)
带QO、状态的CNF；(*请求服务的确认*)



问：布尔值；(*事件输入限定符*)
编号：WSTRING；(*要写入的变量路径*)
类型：WSTRING；(*SD变量的数据类型*)
标清：任何；(*要写入的变量值*)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_write
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateWrite(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.writeTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

IEC61499客户端设备可以使用图D.5中图形显示和表D.5文本显示的TASK功能块类型的实例来请求在

当实现支持此功能时，不会为SINGLE或
IEC61131-3:2003第50条；相反，会触发相应任务的执行，如图D.5所示的request_task服务序列所示。




NOTE The graphical representation of other service sequences listed in Table D.5 is similar to Figure D.2.

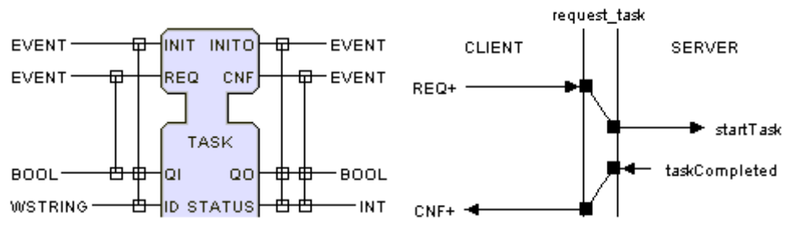
**Figure D.5 – Function block type TASK**

**Table D.5 – Source code of function block type TASK**

```
FUNCTION_BLOCK TASK (* Trigger IEC 61131 task *)
EVENT_INPUT
  INIT WITH QI,ID; (* Initialize/Terminate Service *)
  REQ WITH QI; (* Service Request *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialize/Terminate Confirm *)
  CNF WITH QO,STATUS; (* Confirmation of Requested Service *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Event Input Qualifier *)
  ID: WSTRING; (* Path to task to be triggered *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID) -> SERVER.initTask(ID) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID) -> SERVER.init(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_task
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateTask(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.taskTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```








注：表D.5中列出的其他服务序列的图形表示类似于图D.2。

**图D.5 功能块类型TASK**

**表D.5 功能块类型TASK的源代码**

```
FUNCTION_BLOCKTASK(*触发IEC61131任务*)

用QI ID初始化；(*初始化终止服务*)
要求QI； (*服务请求*)
END_EVENT


以QO、状态初始化；(*初始化终止确认*)
带QO、状态的CNF；(*请求服务的确认*)
END_EVENT


问：布尔值；(*事件输入限定符*)
编号：WSTRING；(*要触发的任务路径*)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Normal operation, 0=Abnormal operation *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID) -> SERVER.initTask(ID) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID) -> SERVER.init(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_task
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateTask(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.taskTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.4 Compliance

The specifications given in Annex D may be referenced in compliance profiles according to the rules given in IEC 61499-4.

When a programmable controller system compliant with IEC 61131-3 supports interoperability with one or more of the IEC 61499 function block types defined in Annex D, it should include in its list of supported features a reference to the supported features taken from Table D.6, and should include specifications of the values for implementation specific features and parameters as defined in 8.1 and 8.2 of IEC 61131-5:2000, respectively.

**Table D.6 – IEC 61499 interoperability features**

| No. | Description |
|---|---|
| 1 | READ function block type |
| 2 | UREAD function block type |
| 3 | WRITE function block type |
| 4 | TASK function block type |



61499-1 © IEC:2012 – 91 –

根据IEC61499-4中给出的规则，附件D中给出的规范可以在合规性配置文件中引用。

当符合IEC61131-3的可编程控制器系统支持与附录D中定义的一种或多种IEC61499功能块类型的互操作性时，它应在其支持的特性列表中包括对表D.6中所支持特性的引用，并且应包括分别在IEC61131-5:2000的8.1和8.2中定义的实现特定功能和参数的值规范。

**表D.6 IEC61499互操作性特性**

| No. | |
|---|---|
| 1 | 读取功能块类型 |
| 2 | UREAD功能块类型 |
| 3 | WRITE功能块类型 |
| 4 | TASK功能块类型 |

# Annex E
(informative)

# Information exchange

## E.1 Use of application layer facilities

Subclause 7.1.3.2 of ISO/IEC 7498-1:1994 identifies a number of facilities provided by *application-entities* (i.e., *entities* in the *application layer*) to enable *application-processes* to exchange information. To provide these facilities, the application-entities use *application-protocols* and *presentation services.* The communication function blocks defined in Clause E.2 may use these facilities, when provided by appropriate application-entities, in the following ways.

a) Communication function blocks utilize the *information transfer* facilities provided by application-entities to provide the *synchronization of cooperating applications* represented by the REQ, CNF, IND, and RSP events and to transfer the data represented by the SD inputs and RD outputs.

b) The following facilities may be used during service initialization as represented by the INIT and INITO events, using elements of the PARAMS data structure as necessary:

- identification of the intended communications partners;
- determination of the acceptable quality of service;
- agreement on responsibility for error recovery;
- agreement on security aspects;
- identification of abstract syntax.

c) Facilities for *selection of mode of dialog* may be used by the specific function block types, e.g., by a SUBSCRIBER to ensure that it is interacting properly with a PUBLISHER.

Many of the facilities listed above may not be provided by application-entities of industrial-process measurement and control systems (IPMCSs). In this case, the communication function blocks shall implement equivalent facilities to provide the required services.

In particular, presentation services are often not provided by IPMCS application-entities. Therefore, in order to facilitate implementation of these services by communication function blocks, transfer syntaxes for both information transfer and application management are defined in Clause E.3.

NOTE 1 See ISO/IEC 7498-1 for definitions of terms used in this annex, but not defined in this standard.

NOTE 2 A *resource* is an "application-process" as defined in ISO/IEC 7498-1.

NOTE 3 The contents of Annex E could be considered normative in that compliance profiles as defined in IEC 61499-4, other standards and specifications can specify a context within which some or all of its provisions are employed.

## E.2 Communication function block types

### E.2.1 General

This subclause defines generic *communication function block types* for *unidirectional* and *bidirectional transactions.* **Implementation-dependent** customizations of these types should adhere to the following rules:

a) the implementation shall specify the data types and semantics of values of the data inputs and data outputs of each such function block type;





# 信息交流

## 应用层设施的使用

ISOIEC7498-1:1994的子条款7.1.3.2确定了应用实体（即应用层中的实体）提供的许多设施，以使应用进程能够交换信息。为了提供这些设施，应用程序实体使用应用程序协议和表示服务。条款E.2中定义的通信功能块可以使用这些设施，当由适当的应用实体提供时，以下列方式。

a)通信功能块利用应用实体提供的信息传输设施来提供由REQ、CNF、IND和RSP事件表示的协作应用程序的同步，并传输由SD输入和RD输出表示的数据。

b)在由INIT和INITO事件表示的服务初始化期间，可以使用以下设施，必要时使用PARAMS数据结构的元素：

- 确定预期的通信合作伙伴；
- 确定可接受的服务质量；
- 错误恢复责任协议；
- 安全方面的协议；

抽象语法的识别。

c)特定功能块类型可以使用选择对话模式的工具，例如，订阅者可以确保它与发布者正确交互。

上面列出的许多设施可能不是由工业过程测量和控制系统(IPMCS)的应用实体提供的。在这种情况下，通信功能块应实现等效设施以提供所需的服务。

特别是，IPMCS应用程序实体通常不提供表示服务。因此，为了便于通过通信功能块实现这些服务，在第E.3节中定义了信息传输和应用程序管理的传输语法。

注1：参见ISOIEC7498-1，了解本附录中使用但未在本标准中定义的术语的定义。

注2资源是ISOIEC7498-1中定义的“应用程序过程”。

注3：附录E的内容可以被认为是规范性的，因为IEC61499-4中定义的合规性配置文件，其他标准和规范可以指定使用其部分或全部条款的上下文。

## E.2 通讯功能块类型

本子条款定义了用于单向和双向事务的通用通信功能块类型。这些类型的依赖于实现的定制应遵守以下规则：

a)实现应规定每个此类功能块类型的数据输入和数据输出的数据类型和值的语义；

b) the implementation shall specify the treatment of abnormal data transfer;

c) the implementation shall specify any differences between the behavior of instances of such function block types and the behaviors specified in Clause E.2.

### E.2.2 Function blocks for unidirectional transactions

Figures E.1 through E.4 provide type declarations and typical service primitive sequences of function blocks which provide *unidirectional transactions* over a *communication connection*. Such a connection consists of one *instance* of `PUBLISH` and one or more instances of `SUBSCRIBE` type.

NOTE 1 Full textual specifications of these function block types are not given in Annex F.

NOTE 2 The data types and semantics of the `PARAMS` input and `STATUS` output are **implementation-dependent**.

NOTE 3 The number (`m`) and types of the received data `RD_1, . . . , RD_m` correspond to the number and types of the transmitted data `SD_1, . . . , SD_m`.

NOTE 4 The means by which communication connections are set up are beyond the scope of this standard.

NOTE 5 Data transfer might be required in order to determine whether `RD_1, . . . , RD_m` meet the constraints expressed in Note 3.

NOTE 6 The transfer syntaxes defined in Clause E.3 can be used to make the determination described in Note 5.

NOTE 7 Treatment of abnormal data transfer is **implementation-dependent.**

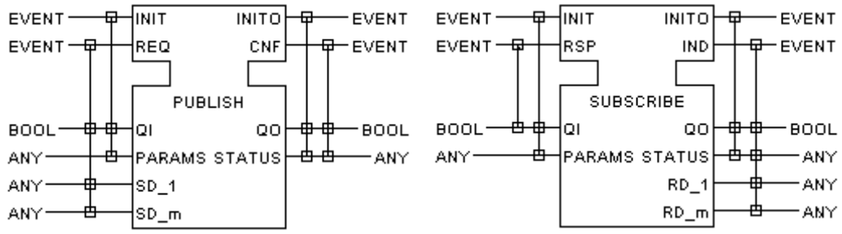



**Figure E.1 – Type specifications for unidirectional transactions**

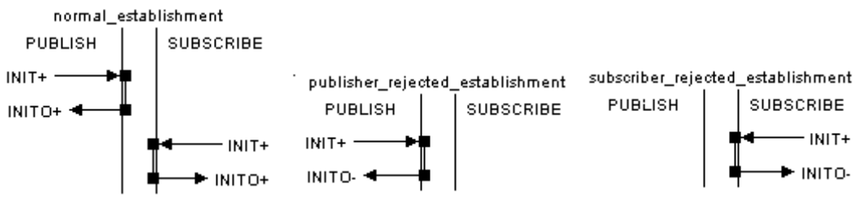



**Figure E.2 – Connection establishment for unidirectional transactions**

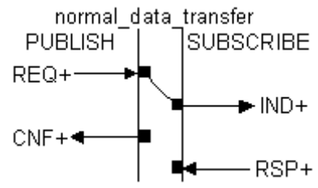



**Figure E.3 – Normal unidirectional data transfer**



b)实施应规定异常数据传输的处理；

c)实现应规定此类功能块类型实例的行为与第E.2条中规定的行为之间的任何差异。

### 单向交易的功能块

图E.1到E.4提供了类型声明和功能块的典型服务原语序列，它们通过通信连接提供单向事务。这种连接由一个PUBLISH实例和一个或多个

订阅类型。

注1这些功能块类型的全文规范未在附录F中给出。

**注2：PARAMS输入和STATUS输出的数据类型和语义是依赖于实现的。**

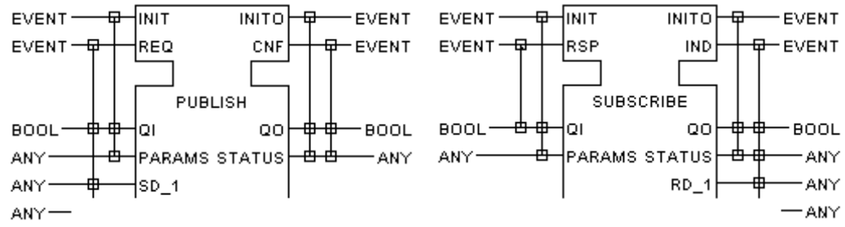
注3接收数据RD_1 … RD_m的数量(m)和类型对应于传输数据SD_1 … SD_m的数量和类型。

注4建立通信连接的方式超出了本标准的范围。

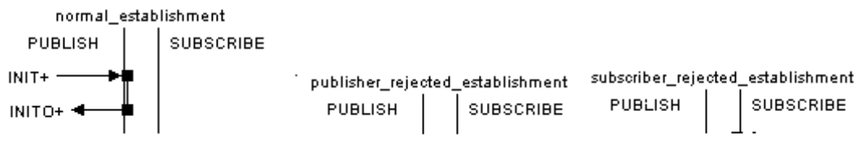
注5可能需要数据传输以确定RD_1 … RD_m是否满足注3中表达的约束。

注6条款E.3中定义的传输语法可用于进行注5中描述的确定。

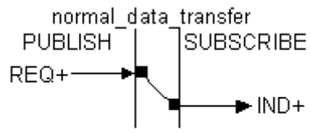
**注7异常数据传输的处理取决于实现。**




**图E.1 单向交易的类型规范**




图E.2 单向事务的连接建立




**图E.3 正常的单向数据传输**

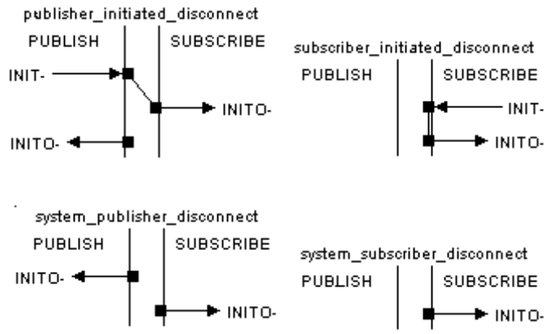



**Figure E.4 – Connection release in unidirectional data transfer**

### E.2.3 Function blocks for bidirectional transactions

Figures E.5 through E.8 provide type declarations and service primitive sequences of function blocks which provide *bidirectional transactions* over a *communication connection*. Such a connection consists of one instance of CLIENT type and one instance of SERVER type.

NOTE 1 Full textual specifications of these function block types are not given in Annex F.

NOTE 2 The data types and semantics of the PARAMS input and STATUS output are **implementation-dependent**.

NOTE 3 The number (m) and types of the received data `RD_1, . . . , RD_m` correspond to the number and types of the transmitted data `SD_1, . . . , SD_m`.

NOTE 4 The number (n) and types of the received data `RD_1, . . . , RD_n` correspond to the number and types of the transmitted data `SD_1, . . . , SD_n`.

NOTE 5 Data transfer may be required in order to determine whether `RD_1, . . . , RD_m` and `RD_1, . . . , RD_n` meet the constraints expressed in Notes 3 and 4.

NOTE 6 The transfer syntaxes defined in Clause E.3 may be used to make the determination described in Note 5.

NOTE 7 Treatment of abnormal data transfer is **implementation-dependent.**

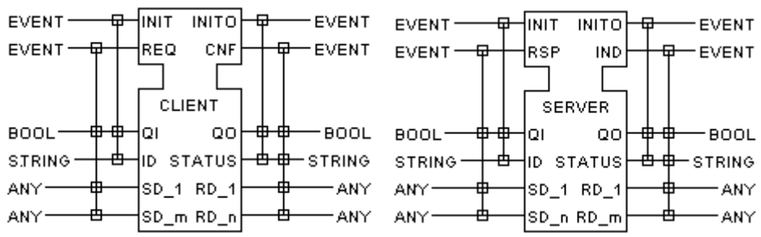
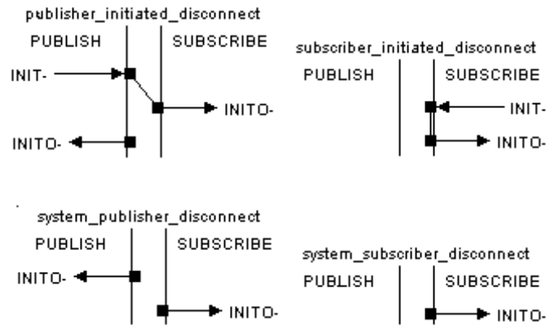



**Figure E.5 – Type specifications for bidirectional transactions**








**图E.4 单向数据传输中的连接释放**

### 双向交易功能块

图E.5到E.8提供了类型声明和功能块的服务原语序列，它们通过通信连接提供双向事务。这样的连接由一个CLIENT类型的实例和一个SERVER类型的实例组成。

注1这些功能块类型的全文规范未在附录F中给出。

**注2：PARAMS输入和STATUS输出的数据类型和语义是依赖于实现的。**

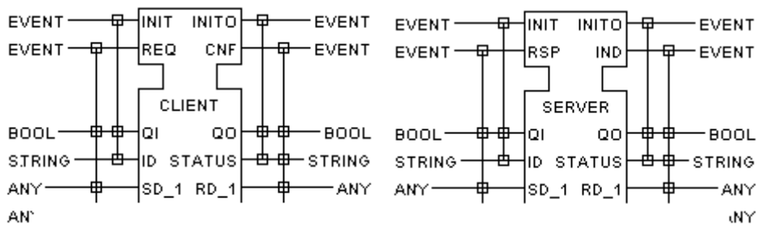
注3接收数据RD_1 … RD_m的数量(m)和类型对应于传输数据SD_1 … SD_m的数量和类型。

注4接收数据RD_1 … RD_n的数量(n)和类型对应于传输数据SD_1 … SD_n的数量和类型。

注5可能需要数据传输以确定RD_1 … RD_m和RD_1 … RD_n是否满足注3和注4中表达的约束。

注6条款E.3中定义的传输语法可用于进行注5中描述的确定。

注7异常数据传输的处理取决于实现。




**图E.5 双向交易的类型规范**

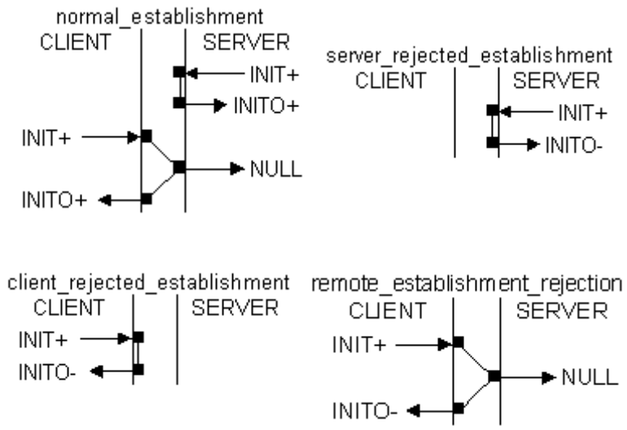



**Figure E.6 – Connection establishment for bidirectional transaction**

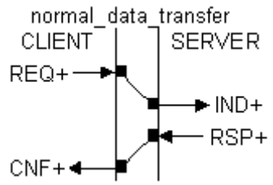



**Figure E.7 – Bidirectional data transfer**

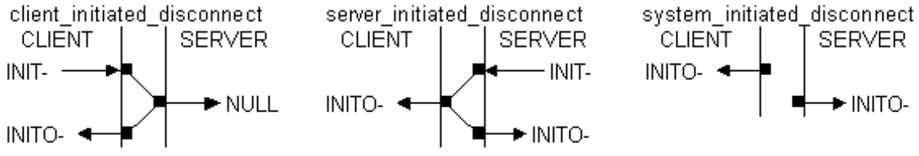



**Figure E.8a – Client initiated** **Figure E.8b – Server initiated** **Figure E.8c – System initiated**

**Figure E.8 – Connection release in bidirectional data transfer**

## E.3 Transfer syntaxes

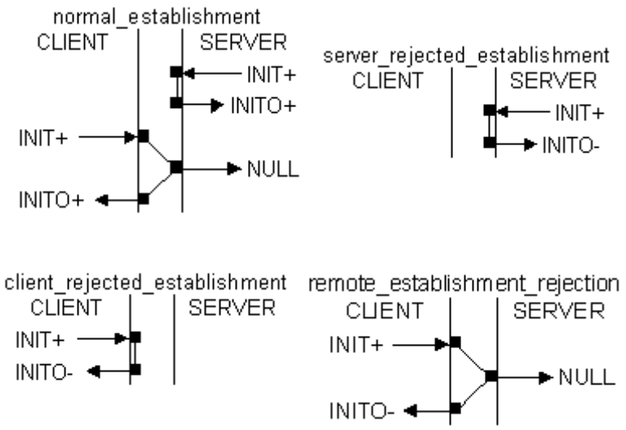
### E.3.1 Background

A transfer syntax is defined in terms of an *abstract syntax* describing the types of data to be transferred, and a set of *encoding rules* for encoded representation of instances of the data types so defined. Subclause E.3.2 utilizes Abstract Syntax Notation One (ASN.1), as defined in ISO/IEC 8824-1, to define the IEC61499-FBDATA syntax for data transfer.

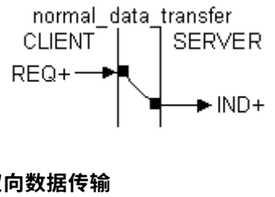
Two sets of encoding rules are given in Annex E:



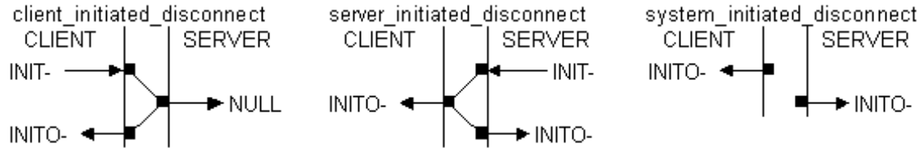
61499-1 © IEC:2012 – 95 –




**图E.6 双向事务的连接建立**




**图E.7 双向数据传输**




**图E.8a 客户端发起** **图E.8b 服务器启动** **图E.8c 系统启动**

**图E.8 双向数据传输中的连接释放**

## E.3 Transfer syntaxes

传输语法是根据描述要传输的数据类型的抽象语法和一组编码规则来定义的，用于对如此定义的数据类型的实例进行编码表示。子条款E.3.2使用ISOIEC8824-1中定义的抽象语法符号一(ASN.1)来定义用于数据传输的IEC61499-FBDATA语法。

附件E给出了两组编码规则：

a) Subclause E.3.3.1 defines BASIC encoding rules, utilizing the rules defined in ISO/IEC 8825-1.

b) Subclause E.3.3.2 utilizes the special characteristics of the data types in the IEC61499-FBDATA syntax to obtain a set of COMPACT encoding rules according to the following principles:

- Where the number of "contents octets" is fixed, "length octets" are not used in the encoding.
- Special encodings are used to minimize the number of octets and encoding/decoding effort required for fixed length types.
- "Identifier octets" are not used for individual elements of STRUCT and ARRAY data types, since the type of each element is fixed in the corresponding *type declaration*.

### E.3.2 IEC61499-FBDATA abstract syntax

The transfer syntax obtained by applying the COMPACT encoding rules in E.3.3.2 to the abstract syntax in E.3.2 is recommended for:

- transferring values from the SD inputs of a *communication function block* to the RD outputs of the communication function block(s) at the opposite end of a *communication connection*;
- determining whether the constraints on corresponding number and type of variables between SD inputs and RD outputs are met as noted in Figures E.1 and E.5.

The use of the abstract syntax defined in E.3.2 for the transfer of data expressed as *literals* and values of *variables* is subject to the following semantic RULES:

a) Where the name of a data type in this module (for example, `BOOL`) corresponds to the name of a data type defined in IEC 61131-3, the type definition given is intended for the transfer of data of the corresponding IEC 61131-3 data type.

b) The values of "VisibleString" for the data types `DATE` and `TIME_OF_DAY` is restricted to the textual syntax for these data types as defined in IEC 61131-3.

c) The notation `[typeID]` implies that the tag of the data consists of the value of the `ASN.1 tag` of the corresponding derived data type, established as specified in Annex A of IEC 61499-2:2005 or by other means beyond the scope of this standard.

d) The value of an `EnumeratedData` item consists of the cardinal position (beginning at zero) of the corresponding identifier in the sequence of identifiers defined for the corresponding enumerated data type, established as specified in IEC 61131-3.

e) The specific type of a `SubrangeData` item is as for its particular *subrange data type*, declared as specified in IEC 61131-3.

f) The type of the elements of an ARRAY data item is established as specified for *array data types* in IEC 61131-3.

g) The types of the elements of a STRUCT data item are established as specified for *structured data types* in IEC 61131-3.

### ASN.1 MODULE

```
IEC61499-FBDATA DEFINITIONS::=

BEGIN

EXPORTS FBDataSequence, FBData, ElementaryData, BOOL, FixedLengthInteger,
     FixedLengthReal, TIME, AnyDate, AnyString, FixedLengthBitString,
     SignedInteger, UnsignedInteger, REAL, LREAL, DATE, TIME_OF_DAY,
     DATE_AND_TIME, STRING, WSTRING, BYTE, WORD, DWORD, LWORD,
     DirectlyDerivedData, EnumeratedData, SubrangeData, ARRAY, STRUCT;

FBDataSequence::= [APPLICATION 23] IMPLICIT SEQUENCE OF FBData

FBData::= CHOICE{ElementaryData, DerivedData}
```



a)使用ISOIEC8825-1中定义的规则，子条款E.3.3.1定义了BASIC编码规则。

b)子条款E.3.3.2利用IEC61499FBDATA语法中数据类型的特殊特性，根据以下原则获得一组COMPACT编码规则：

- 在“内容八位字节”的数量固定的情况下，编码中不使用“长度八位字节”。
- 特殊编码用于最小化固定长度类型所需的八位字节数和编码解码工作量。
- “标识符八位字节”不用于STRUCT和ARRAY数据类型的单个元素，因为每个元素的类型在相应的类型声明中是固定的。

### IEC61499-FBDATA抽象语法

通过将E.3.3.2中的COMPACT编码规则应用于E.3.2中的抽象语法获得的传输语法推荐用于：

- 将值从通信功能块的SD输入传输到通信连接另一端的通信功能块的RD输出；
- 确定是否满足SD输入和RD输出之间变量的相应数量和类型的约束，如图E.1和E.5所示。

使用E.3.2中定义的抽象语法来传输表示为文字和变量值的数据，必须遵守以下语义规则：

a)如果此模块中的数据类型名称（例如BOOL）对应于IEC61131-3中定义的数据类型名称，则给出的类型定义旨在传输相应的IEC61131-3数据类型。

b)数据类型DATE和TIME_OF_DAY的“VisibleString”值受限于IEC61131-3中定义的这些数据类型的文本语法。

c)符号[typeID]表示数据标签由相应派生数据类型的ASN.1标签的值组成，按照IEC61499-2:2005附件A的规定或通过其他方式建立本标准的范围。

d)EnumeratedData项的值由为对应枚举数据类型定义的标识符序列中对应标识符的基数位置（从零开始）组成，按照IEC61131-3的规定建立。

e)SubrangeData项的特定类型与其特定的子范围数据类型一样，声明为IEC61131-3中的规定。

ARRAY数据项的元素类型按照IEC61131-3中对数组数据类型的规定建立。

g)STRUCT数据项的元素类型按照IEC61131-3中结构化数据类型的规定建立。

### ASN.1 MODULE

```
IEC61499-FBDATA DEFINITIONS::=

BEGIN

EXPO                                                              ger,

     有符号整数、无符号整数、实数、实数、日期、TIME_OF_DAY、
     DATE_AND_TIME STRING WSTRING BYTE WORD DWORD LWORD
                                                            TRUCT;
```

FBDataSequence::=[应用23]FBData的隐式序列

```
FBData::= CHOICE{ElementaryData, DerivedData}
```

```
ElementaryData::= CHOICE{
     BOOL,
     FixedLengthInteger,
     FixedLengthReal,
     TIME,
     AnyDate,
     AnyString,
     FixedLengthBitString}

FixedLengthInteger::= CHOICE{SignedInteger, UnsignedInteger}

SignedInteger::= CHOICE{SINT, INT, DINT, LINT}

UnsignedInteger::= CHOICE{USINT, UINT, UDINT, ULINT}

FixedLengthReal::= CHOICE{REAL, LREAL}

AnyDate::= CHOICE{DATE, TIME_OF_DAY, DATE_AND_TIME}

AnyString::= CHOICE{STRING, WSTRING}

FixedLengthBitString::= CHOICE{BYTE, WORD, DWORD, LWORD}

BOOL::= CHOICE{BOOL0, BOOL1}

BOOL0::= [APPLICATION 0] IMPLICIT NULL

BOOL1::= [APPLICATION 1] IMPLICIT NULL

SINT::= [APPLICATION 2] IMPLICIT INTEGER(-128..127)

INT::= [APPLICATION 3] IMPLICIT INTEGER(-32768..32767)

DINT::= [APPLICATION 4] IMPLICIT INTEGER(-2147483648..2147483647)

LINT::= [APPLICATION 5]
     IMPLICIT INTEGER(-9223372036854775808..9223372036854775807)

USINT::= [APPLICATION 6] IMPLICIT INTEGER(0..255)

UINT::= [APPLICATION 7] IMPLICIT INTEGER(0..65535)

UDINT::= [APPLICATION 8] IMPLICIT INTEGER(0..4294967295)

ULINT::= [APPLICATION 9] IMPLICIT INTEGER(0..18446744073709551615)

REAL::= [APPLICATION 10] IMPLICIT OCTET STRING (SIZE(4))

LREAL::= [APPLICATION 11] IMPLICIT OCTET STRING (SIZE(8))

TIME::= [APPLICATION 12] IMPLICIT LINT -- Duration in 1μs units

DATE::= [APPLICATION 13] IMPLICIT ULINT -- See Table E.1.

TIME_OF_DAY::= [APPLICATION 14] IMPLICIT ULINT -- See Table E.1.

DATE_AND_TIME::= [APPLICATION 15] IMPLICIT ULINT -- See Table E.1.

STRING::= [APPLICATION 16] IMPLICIT OCTET STRING -- 1 octet/char

BYTE::= [APPLICATION 17] IMPLICIT BIT STRING (SIZE(8))

WORD::= [APPLICATION 18] IMPLICIT BIT STRING (SIZE(16))

DWORD::= [APPLICATION 19] IMPLICIT BIT STRING (SIZE(32))

LWORD::= [APPLICATION 20] IMPLICIT BIT STRING (SIZE(64))

WSTRING::= [APPLICATION 21] IMPLICIT OCTET STRING -- 2 octets/char

DerivedData::= CHOICE{
     DirectlyDerivedData,
     EnumeratedData,
     SubrangeData,
     ARRAY,
     STRUCT}
```



61499-1 © IEC:2012 – 97 –

```
ElementaryData::= CHOICE{
     BOOL,
     FixedLengthInteger,
     FixedLengthReal,
     TIME,
     AnyDate,
     AnyString,
     FixedLengthBitString}
                                                    ignedInteger}
```

SignedInteger::=CHOICE{SINT INT DINT LINT

```
UnsignedInteger::= CHOICE{USINT, UINT, UDINT, ULINT}

FixedLengthReal::= CHOICE{REAL, LREAL}

AnyDate::= CHOICE{DATE, TIME_OF_DAY, DATE_AND_TIME}

AnyString::= CHOICE{STRING, WSTRING}

FixedLengthBitString::= CHOICE{BYTE, WORD, DWORD, LWORD}
```

BOOL0::=[应用程序0]隐式NULL

BOOL1::=[应用程序1]隐式NULL

```
SINT::= [APPLICATION 2] IMPLICIT INTEGER(-128..127)

INT::= [APPLICATION 3] IMPLICIT INTEGER(-32768..32767)

DINT::= [APPLICATION 4] IMPLICIT INTEGER(-2147483648..2147483647)

LINT::= [APPLICATION 5]
     IMPLICIT INTEGER(-9223372036854775808..9223372036854775807)

USINT::= [APPLICATION 6] IMPLICIT INTEGER(0..255)

UINT::= [APPLICATION 7] IMPLICIT INTEGER(0..65535)

UDINT::= [APPLICATION 8] IMPLICIT INTEGER(0..4294967295)
                                                              09551615)
```

REAL::=[应用程序10]隐式八位字节字符串（大小（4））

LREAL::=[应用程序11]隐式八位字节字符串（大小（8））

TIME::=[应用程序12]隐式LINT-持续时间以1μs为单位

DATE::=[应用程序13]隐式ULINT-见表E.1。

TIME_OF_DAY::=[应用程序14]隐式ULINT-见表E.1。

DATE_AND_TIME::=[应用程序15]隐式ULINT-见表E.1。

STRING::=[应用程序16]隐式八位字节字符串-1八位字节字符

BYTE::=[应用程序17]隐式位串（大小（8））

WORD::=[应用程序18]隐式位串（大小（16））

DWORD::=[应用程序19]隐式位串（大小（32））

LWORD::=[应用程序20]隐式位串（大小（64））

WSTRING::=[应用程序21]隐式八位字节字符串-2八位字节字符

```
DerivedData::= CHOICE{
     DirectlyDerivedData,
     EnumeratedData,
     SubrangeData,
     ARRAY,
     STRUCT}
```

```
DirectlyDerivedData::= [typeID] IMPLICIT ElementaryData
EnumeratedData::= [typeID] IMPLICIT UINT
SubrangeData::= [typeID] IMPLICIT FixedLengthInteger
ARRAY::= CHOICE {ArrayVariable, TypedArray}
ArrayVariable::= [APPLICATION 22] IMPLICIT FBDataSequence -- same type
TypedArray::= [typeID] IMPLICIT FBDataSequence - same type
STRUCT::= [typeID] IMPLICIT SEQUENCE -- different types
END
```

### E.3.3 Encoding rules

#### E.3.3.1 BASIC encoding

This encoding shall be the result of applying the basic encoding rules of ISO/IEC 8825-1 to variables of the types defined in E.3.2.

#### E.3.3.2 COMPACT encoding

This encoding shall be the result of modifying the rules for BASIC encoding given in E.3.3.1 as follows.

a) "Length octets" shall not be included in the encoding of values of the data types shown in Table E.1.

b) The length (in octets) and encoding of the "contents octets" described in ISO/IEC 8825-1 shall be as defined in Table E.1 for values of the data types shown there.

c) Encoding of variables of `TIME`, `DirectlyDerivedData`, `EnumeratedData`, or `SubrangeData` types shall follow the same encoding rules as the base type.

d) "Type octets" shall not be included in the encoding of individual elements of `STRUCT` types, except for the encoding of elements of type `BOOL`, which shall be encoded according to rule (1) of Table E.1.

e) The encoding of values of `STRING` and `WSTRING` types shall be primitive.

f) The encoding of `ARRAY` elements shall be *constructed* in the sense of ISO/IEC 8825-1, with the following provisions for COMPACT encoding:

   1) The "length" subfield of the `ARRAY` element shall be encoded as a value of the `UINT` type without identifier or length octets, i.e., as a 16-bit unsigned integer;

      NOTE 1 This would appear to restrict the maximum number of elements of an `ARRAY` to 65535. However, the actual length may be further restricted by the maximum number of octets that can be transferred by the underlying transport protocol.

      EXAMPLE For UDP messages with a maximum number of is 65508 octets, the maximum transmittable length of an `ARRAY` of `BYTE` elements would be (maximum octets - tag octets - length octets - element type octets)/(element length) = (65508-1-2-1)/1 = 65504 elements.

   2) COMPACT encoding shall be used for the first element of the "values" field;

   3) Subsequent elements, if any, shall be encoded using the COMPACT syntax without an "identifier" subfield, except for elements of type `BOOL`, which shall be encoded according to rule (1) of Table E.1;

   4) If the specified length of the received `ARRAY` is less than the locally allocated space, the remaining elements of the local array are unaffected; if the length of the received `ARRAY` is greater than the locally allocated space, the remaining received elements are ignored.

NOTE 2 Since `ARRAY` is a subclass of `FBData`, a multidimensional `ARRAY` can be encoded recursively as an `ARRAY` whose elements are `ARRAY` elements.






```
DirectlyDerivedData::= [typeID] IMPLICIT ElementaryData
EnumeratedData::= [typeID] IMPLICIT UINT
SubrangeData::= [typeID] IMPLICIT FixedLengthInteger
```


ArrayVariable::=[应用22]隐式FBDataSequence–相同类型

TypedArray::=[typeID]IMPLICITFBDataSequence相同类型

STRUCT::=[typeID]隐式序列–不同的类型

END

### E.3.3 编码规则

#### 基本编码

该编码应是将ISOIEC8825–1的基本编码规则应用于E.3.2中定义的类型的变量的结果。

#### COMPACT编码

该编码应是修改E.3.3.1中给出的BASIC编码规则的结果，如下所示。

a)“长度八位字节”不应包含在表E.1所示数据类型值的编码中。

b)ISOIEC8825–1中描述的“内容八位字节”的长度（以八位字节为单位）和编码应如表E.1中所示的数据类型值的定义。

c)TIME、DirectlyDerivedData、EnumeratedData或SubrangeData类型的变量的编码应遵循与基本类型相同的编码规则。

d)“类型八位字节”不应包含在STRUCT类型的单个元素的编码中，但BOOL类型的元素的编码除外，它应根据表E.1的规则(1)进行编码。

e)STRING和WSTRING类型值的编码应该是原始的。

f) ARRAY元素的编码应按照ISOIEC8825–1的意义构建，对COMPACT编码有以下规定：

1)ARRAY元素的“length”子字段应编码为不带标识符或长度八位字节的UINT类型的值，即16位无符号整数；

注1这似乎将ARRAY的最大元素数限制为65535。但是，实际长度可能进一步受到底层传输协议可以传输的最大八位字节数的限制。

示例对于最大数量为65508个八位字节的UDP消息，BYTE元素的数组的最大可传输长度将是（最大八位字节标记八位字节长度八位字节元素类型八位字节）（元素长度）=（65508–1–2–1）1=65504个元素。

2)COMPACT编码应用于“values”字段的第一个元素；

3)后续元素（如果有）应使用不带“标识符”子字段的COMPACT语法进行编码，但类型为BOOL的元素除外，其应根据表E.1的规则(1)进行编码；

4）如果接收到的ARRAY的指定长度小于本地分配的空间，则本地数组的剩余元素不受影响；如果接收到的ARRAY的长度大于本地分配的空间，则忽略接收到的剩余元素。

注2由于ARRAY是FBData的子类，因此可以将多维ARRAY递归编码为其元素为ARRAY元素的ARRAY。

**Table E.1 – COMPACT encoding of fixed length data types**

| Data type | Contents octets | |
|---|---|---|
| | Length | Encoding rule |
| **BOOL** | 0 | (1) |
| **SINT** | 1 | (2) |
| **INT** | 2 | (2) |
| **DINT** | 4 | (2) |
| **LINT** | 8 | (2) |
| **USINT** | 1 | (3) |
| **UINT** | 2 | (3) |
| **UDINT** | 4 | (3) |
| **ULINT** | 8 | (3) |
| **REAL** | 4 | (4) |
| **LREAL** | 8 | (4) |
| **DATE** | 8 | (5) |
| **TIME** | 8 | (7) |
| **TIME_OF_DAY** | 12 | (5) |
| **DATE_AND_TIME** | 20 | (5) |
| **BYTE** | 1 | (6) |
| **WORD** | 2 | (6) |
| **DWORD** | 4 | (6) |
| **LWORD** | 8 | (6) |

ENCODING RULES FOR TABLE E.1

(1) Values of this data type shall be encoded as a single identifier octet containing the tag encoding for the `BOOL0` or `BOOL1` class, as defined in E.3.2, corresponding to values of `FALSE (0)` or `TRUE (1)`, respectively.

(2) Values of these `SignedInteger` data types shall be encoded in the same manner as an `UnsignedInteger` of the same length as the `SignedInteger` type with a value of $N - N_{min}$, where N is the value of the `SignedInteger` variable to be encoded and $N_{min}$ is the lower end point of the value range of the `SignedInteger` subtype as defined in E.3.2.

(3) Values of these `UnsignedInteger` data types shall be encoded by numbering the bits in the contents octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 of the first octet. Each bit is assigned a value of $2^N$, where N is its position in the above numbering sequence. The value of the unsigned integer is obtained by summing the numerical values assigned to each bit for those bits which are set to one.

(4) Values of these data types shall be encoded as 32-bit single format and 64-bit double format numbers, respectively, as defined in ISO/IEC/IEEE 60559, where the "lsb" defined in ISO/IEC/IEEE 60559 corresponds to "bit zero" as defined in Rule (3).

(5) Values of these types shall be encoded as for type `ULINT`, representing the number of milliseconds since midnight for `TIME_OF_DAY`, the number of milliseconds since `1970-01-01-00:00:00.000` for `DATE_AND_TIME`, or the number of milliseconds from `1970-01-01-00:00:00.000` to `YYYY-MM-DD-00:00:00.000` for `DATE`, where `YYYY-MM_DD` is the current date.

(6) Encoding of values of these `FixedLengthBitString` data types shall be primitive, and shall be obtained by placing the bits in the bitstring, commencing with the first bit and proceeding to the trailing bit, in bits 8 to 1 of the first contents octet, followed in turn by bits 8 to 1 of each of the subsequent octets, where the notation "first bit" and "trailing bit" is specified in ISO/IEC 8824-1.

(7) Encoding of values of this data type shall be the same as for values of type `LINT`, representing a time interval in units of 1 μs.





**表E.1 固定长度数据类型的COMPACT编码**

| 数据类型 | Contents octets | |
|---|---|---|
| | Length | 编码规则 |
| **BOOL** | 0 | (1) |
| **SINT** | 1 | (2) |
| **INT** | 2 | (2) |
| **DINT** | 4 | (2) |
| **LINT** | 8 | (2) |
| **USINT** | 1 | (3) |
| **UINT** | 2 | (3) |
| **UDINT** | 4 | (3) |
| **ULINT** | 8 | (3) |
| **REAL** | 4 | (4) |
| **LREAL** | 8 | (4) |
| **DATE** | 8 | (5) |
| **TIME** | 8 | (7) |
| **TIME_OF_DAY** | 12 | (5) |
| **DATE_AND_TIME** | 20 | (5) |
| **BYTE** | 1 | (6) |
| **WORD** | 2 | (6) |
| **DWORD** | 4 | (6) |
| | 8 | (6) |

表E.1的编码规则

(1)此数据类型的值应编码为单个标识符八位字节，其中包含E.3.2中定义的BOOL0或BOOL1类的标签编码，分别对应于FALSE(0)或TRUE(1)的值。

(2)这些SignedInteger数据类型的值应以与
UnsignedInteger与SignedInteger类型长度相同，值为N Nmin，其中N是要编码的SignedInteger变量的值，Nmin是E.3.2中定义的SignedInteger子类型的值范围的下端点.

(3)这些UnsignedInteger数据类型的值应通过对内容中的位编号进行编码
字节，从最后一个字节的第1位开始作为第0位，并以第一个字节的第8位结束编号。每个位被分配一个值2N，其中N是它在上述编号序列中的位置。无符号整数的值是通过对设置为1的那些位分配给每个位的数值求和来获得的。

(4)这些数据类型的值应编码为32位单格式和64位双格式数字，
分别如ISOIECIEEE60559中定义的那样，其中ISOIECIEEE60559中定义的“lsb”对应于规则(3)中定义的“位零”。

(5)这些类型的值应编码为ULINT类型，表示毫秒数
自午夜以来TIME_OF_DAY，自1970-01-01-00:00:00.000以来的毫秒数DATE_AND_TIME，或从1970-01-01-00:00:00.000到YYYYMM-DD-00:00的毫秒数：00.000表示DATE，其中YYYY-MM_DD是当前日期。

(6)这些FixedLengthBitString数据类型的值的编码应该是原始的，并且应该是
通过将位放入位串中获得，从第一位开始并继续到尾随位，在第一个内容八位字节的位8到1中，然后依次是每个后续八位位组的位8到1，其中符号ISOIEC8824-1规定了“第一位”和“尾随位”。

(7)该数据类型值的编码应与LINT类型值的编码相同，表示时间
间隔以1μs为单位。

# Annex F
(normative)

## Textual specifications

Annex F provides textual specifications, in the syntax defined in Annex B, for all function block and adapter types illustrated in this standard. The contents of Annex F are normative to the extent defined in the description of each such function block type or adapter type in this standard.

NOTE The specifications are listed alphabetically by type name.

```
(*!===========================================================================*)
FUNCTION_BLOCK E_CTU (* Event-Driven Up Counter *)
EVENT_INPUT
  CU WITH PV; (* Count Up *)
  R; (* Reset *)
END_EVENT
EVENT_OUTPUT
  CUO WITH Q,CV; (* Count Up Output Event *)
  RO WITH Q,CV; (* Reset Output Event *)
END_EVENT
VAR_INPUT
  PV: UINT; (* Preset Value *)
END_VAR
VAR_OUTPUT
  Q: BOOL; (* CV>=PV *)
  CV: UINT;
END_VAR
EC_STATES
  START;
  CU: CU -> CUO;
  R: R -> RO;
END_STATES
EC_TRANSITIONS
  START TO CU:= CU [CV<65535];
  CU TO START:= 1;
  START TO R:= R;
  R TO START:= 1;
END_TRANSITIONS
ALGORITHM CU IN ST: (* Count Up *)
CV:= CV + 1;
Q:= (CV >= PV);
END_ALGORITHM
ALGORITHM R IN ST: (* Reset *)
CV:= 0;
Q:= FALSE;
END_ALGORITHM
END_FUNCTION_BLOCK
(*!===========================================================================*)
FUNCTION_BLOCK E_CYCLE (* Periodic (cyclic) Generation of an Event *)
EVENT_INPUT
  START WITH DT;
  STOP;
END_EVENT
EVENT_OUTPUT
 EO;   (* Periodic event at period DT, starting at DT after GO *)
END_EVENT
VAR_INPUT
  DT: TIME; (* Period between events *)
END_VAR
FBS
 DLY: E_DELAY;
END_FBS
EVENT_CONNECTIONS
 START TO DLY.START;
```





## 文字规格

附录F以附录B中定义的语法提供了本标准中说明的所有功能块和适配器类型的文本规范。附件F的内容在本标准中每个此类功能块类型或适配器类型的描述中定义的范围内是规范性的。

注意规格按型号名称的字母顺序列出。

```
(*!===========================================================================*)FUNCTION
_BLOCKE_CTU(*事件驱动的向上计数器*)

铜带光伏; (*计数*)
  R; (* Reset *)


CUOWITHQ CV;(*计数输出事件*)
RO带Q CV;(*复位输出事件*)


PV: 单位; (*预设值*)
END_VAR
VAR_OUTPUT
  Q: BOOL; (* CV>=PV *)
  CV: UINT;
END_VAR
EC_STATES
  START;
  CU: CU -> CUO;
  R: R -> RO;


开始到CU: =CU[CV<65535];
CU开始: =1;
开始R:=R;R开始: =1;


ST中的算法CU: (*向上计数*)
CV:= CV + 1;


ST中的算法R: (*重置*)
CV:= 0;


END_FUNCTION_BLOCK(*!===========================================================
=========FUNCTION_BLOCKE_CYCLE (*周期性（循环）生成事件*)


从DT开始;
  STOP;


EO;(*周期DT的周期性事件，从GO后的DT开始*)


DT: 时间; (*事件之间的时间段*)
END_VAR
FBS
 DLY: E_DELAY;


开始DLY.START;
```

```
 STOP TO DLY.STOP;
 DLY.EO TO DLY.START;
 DLY.EO TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_D_FF (* Event-driven Data(D)Latch *)
EVENT_INPUT
  CLK WITH D; (* Data Clock *)
END_EVENT
EVENT_OUTPUT
  EO WITH Q; (* Output Event when Q output changes *)
END_EVENT
VAR_INPUT
  D: BOOL; (* Data Input *)
END_VAR
VAR_OUTPUT
  Q: BOOL; (* Latched Data *)
END_VAR
EC_STATES
  Q0; (* Q is FALSE initially *)
  RESET: LATCH -> EO; (* Reset Q and issue EO *)
  SET: LATCH -> EO; (* Latch and issue EO *)
END_STATES
EC_TRANSITIONS
  Q0 TO SET:= CLK [D];
  SET TO RESET:= CLK [NOT D];
  RESET TO SET:= CLK [D];
END_TRANSITIONS
ALGORITHM LATCH IN ST:
Q:=D;
END_ALGORITHM
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_DELAY
 (* Delayed propagation of an event - Cancellable *)
EVENT_INPUT
 START WITH DT; (* Begin Delay *)
 STOP;      (* Cancel Delay *)
END_EVENT
EVENT_OUTPUT
 EO;     (* Delayed Event *)
END_EVENT
VAR_INPUT
 DT: TIME; (* Delay Time *)
END_VAR
SERVICE E_DELAY/RESOURCE
SEQUENCE event_delay
 E_DELAY.START(DT) ->E_DELAY.EO();
END_SEQUENCE
SEQUENCE delay_canceled
 E_DELAY.START(DT);
 E_DELAY.STOP();
END_SEQUENCE
SEQUENCE no_multiple_delay
 E_DELAY.START(DT);
 E_DELAY.START(DT);
 ->E_DELAY.EO();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_DEMUX (* Event demultiplexer *)
EVENT_INPUT
 EI WITH K;  (* Event to demultiplex *)
END_EVENT
```





```
停下来DLY.STOP;
 DLY.EO TO DLY.START;
 DLY.EO TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=================================================================*)
                                    driven Data(D)Latch *)

带D的时钟；(*数据时钟*)


EO与Q；(*Q输出改变时的输出事件*)


D：布尔值；  (*数据输入*)


问：布尔；(*锁存数据*)


Q0;(*Q最初为FALSE*)
复位：锁存器->EO；(*重置Q并发出EO*)
设置：锁存器->EO；(*锁定并发布EO*)
END_STATES


设置为复位：=CLK[非D];
重置为设置：=CLK[D];

ST中的算法锁存器：


END_FUNCTION_BLOCK(*=============================================================
==========FUNCTION_BLOCKE_DELAY(*可取消事件的延迟传播*)EVENT_INPUT



从DT开始；(*开始延迟*)
停止;(*取消延迟*)


EO;(*延迟事件*)


DT：时间；  (*延迟时间*)
END_VAR
SERVICE E_DELAY/RESOURCE
SEQUENCE event_delay
 E_DELAY.START(DT) ->E_DELAY.EO();
END_SEQUENCE
SEQUENCE delay_canceled
 E_DELAY.START(DT);
 E_DELAY.STOP();
END_SEQUENCE
SEQUENCE no_multiple_delay
 E_DELAY.START(DT);
 E_DELAY.START(DT);
 ->E_DELAY.EO();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
(*=================================================================*)
                                          lexer *)

EI与K；(*要解复用的事件*)
END_EVENT
```

```
EVENT_OUTPUT
 EO0;
 EO1;
 EO2;
 EO3;     (* Number of outputs is implementation dependent *)
END_EVENT
VAR_INPUT
 K: UINT;  (* Event index, maximum is implementation dependent *)
END_VAR
EC_STATES
 START; (* Initial State *)
 TRIGGERED; (* Intermediate state after EI arrives *)
 EO0: -> EO0;
 EO1: -> EO1;
 EO2: -> EO2;
 EO3: -> EO3;
END_STATES
EC_TRANSITIONS
 START TO TRIGGERED:= EI;
 TRIGGERED TO EO0:= [K=0];
 TRIGGERED TO EO1:= [K=1];
 TRIGGERED TO EO2:= [K=2];
 TRIGGERED TO EO3:= [K=3];
 TRIGGERED TO START:= [K>3];
 EO0 TO START:= 1;
 EO1 TO START:= 1;
 EO2 TO START:= 1;
 EO3 TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_F_TRIG (* Boolean falling edge detection *)
EVENT_INPUT
 EI WITH QI;   (* Event Input *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Event Output *)
END_EVENT
VAR_INPUT
 QI: BOOL;  (* Boolean input for falling edge detection *)
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;
END_FBS
EVENT_CONNECTIONS
  EI TO D.CLK;
  D.EO TO SW.EI;
  SW.EO0 TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
  QI TO D.D;
  D.Q TO SW.G;
END_CONNECTIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_MERGE (* Merge (OR) of multiple events *)
EVENT_INPUT
  EI1; (* First input event *)
  EI2; (* Second input event *)
  END_EVENT
EVENT_OUTPUT EO; (* Output Event *)
END_EVENT
EC_STATES
  START;  (* Initial State *)
  EO:    (* Issue EO Event *)
    ->EO;
END_STATES
EC_TRANSITIONS
  START TO EO:= EI1;
```





```
EVENT_OUTPUT
 EO0;
```

EO3;（*输出数量取决于实现*）

K：单位；(*事件索引，最大值取决于实现*)

开始;(*初始状态*)
触发；(*EI到达后的中间状态*)

```
 EO0: -> EO0;
 EO1: -> EO1;
 EO2: -> EO2;
 EO3: -> EO3;
```

开始触发：=EI;
触发到EO0:=[K=0];触发到EO1:=[K=1];
触发到EO2:=[K=2];触发到EO3:=[K=3];

触发开始：=[K>3];

```
 EO0 TO START:= 1;
 EO1 TO START:= 1;
 EO2 TO START:= 1;
```

END_FUNCTION_BLOCK==========
==========FUNCTION_BLOCKE_F_TRIG（*布尔下降沿检测*）

EI与气；(*事件输入*)

EO;(*事件输出*)

问：布尔值；(*用于下降沿检测的布尔输入*)

```
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;
```

EI转D.CLK;
D.EO到SW.EI;

```
  SW.EO0 TO EO;
             NS
             ONS
```

QITOD.D;

END_FUNCTION_BLOCK==========
==========FUNCTION_BLOCKE_MERGE(*合并(OR)多个事件*)

EI1;(*第一个输入事件*)
EI2;(*第二个输入事件*)

EVENT_OUTPUTEO；(*输出事件*)

开始;(*InitialState*)EO:(*IssueEOEvent*)->E
O;END_STATES

开始到EO:=EI1;

```
  START TO EO:= EI2;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_N_TABLE (* Generation of a finite train of separate events,
table driven *)
EVENT_INPUT
  START WITH DT, N;
  STOP;
END_EVENT
EVENT_OUTPUT
 EO0;   (* N events at periods DT, starting at DT[0] after START *)
 EO1;
 EO2;
 EO3;   (* Extensible *)
END_EVENT
VAR_INPUT
  DT: TIME[3]; (* Periods between events *)
  N: UINT;  (* Number of events to generate (=3 in this example) *)
END_VAR
SERVICE E_N_TABLE/RESOURCE
SEQUENCE typical_operation
  E_N_TABLE.START(DT,N) -> E_N_TABLE.EO0() -> E_N_TABLE.EO1() ->
E_N_TABLE.EO2() -> E_N_TABLE.EO3();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_PERMIT (* Permissive propagation of an event *)
EVENT_INPUT EI WITH PERMIT; (* Event input *)
END_EVENT
EVENT_OUTPUT EO; (* Event output *)
END_EVENT
VAR_INPUT PERMIT: BOOL; END_VAR
EC_STATES
  START;  (* Initial State *)
  EO:    (* Issue EO Event *)
    ->EO;
END_STATES
EC_TRANSITIONS
  START TO EO:= EI [PERMIT];
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_R_TRIG (* Boolean rising edge detection *)
EVENT_INPUT
 EI WITH QI;   (* Event Input *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Event Output *)
END_EVENT
VAR_INPUT
 QI: BOOL;  (* Boolean input for rising edge detection *)
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;
END_FBS
EVENT_CONNECTIONS
  EI TO D.CLK;
  D.EO TO SW.EI;
  SW.EO1 TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
  QI TO D.D;
  D.Q TO SW.G;
END_CONNECTIONS
```





开始到EO:=EI2;
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK(*=================================================================
=FUNCTION_BLOCKE_N_TABLE(*生成有限序列的单独事件，表驱动*)EVENT_INPUT

以DT、N开头；
  STOP;

EO0;（*时间段DT的N个事件，从START之后的DT[0]开始*）
 EO1;
 EO2;
 EO3;   (* Extensible *)

DT：时间[3]；(*事件之间的时间段*)
N：单位；（*要生成的事件数（在本例中=3）*）

SERVICEE_N_TABLERESOURCESEQUEN
CE典型操作
  E_N_TABLE.START(DT,N) -> E_N_TABLE.EO0() -> E_N_TABLE.EO1() ->
E_N_TABLE.EO2() -> E_N_TABLE.EO3();

END_FUNCTION_BLOCK(*=================================================================
==========FUNCTION_BLOCKE_PERMIT(*允许传播事件*)

EVENT_INPUTEIWITHPERMIT；(*事件输入*)
END_EVENTEVENT_OUTPUTEO；(*事件输出*)

END_EVENT
R

开始;(*InitialState*)EO:(*IssueEOEvent*)->E
O;END_STATES

开始到EO:=EI[许可];
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK(*=================================================================
==========FUNCTION_BLOCKE_R_TRIG（*布尔上升沿检测*）

EI与气；(*事件输入*)

EO;(*事件输出*)

问：布尔值；(*上升沿检测的布尔输入*)
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;

EI转D.CLK；
D.EO到SW.EI；
  SW.EO1 TO EO;
NS
ONS
QITOD.D;
  D.Q TO SW.G;
END_CONNECTIONS

```
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_REND (* Rendezvous of two events *)
EVENT_INPUT
  EI1; (* First Event Input *)
  EI2; (* Second Event Input *)
  R;  (* Reset Event *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Rendezvous Output Event *)
END_EVENT
EC_STATES
  START;  (* Initial State *)
  EI1;   (* EI1 has arrived, wait for EI2 or R *)
  EO:    (* Issue rendezvous event *)
    ->EO;
  EI2;   (* EI2 has arrived, wait for EI1 or R *)
END_STATES
EC_TRANSITIONS
  START TO EI1:= EI1;
  EI1 TO START:= R;
  START TO EI2:= EI2;
  EI2 TO START:= R;
  EI1 TO EO:= EI2;
  EI2 TO EO:= EI1;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_RESTART (* Generation of Restart Events *)
EVENT_OUTPUT
 COLD;     (* Cold Restart *)
 WARM;     (* Warm Restart *)
END_EVENT
SERVICE RESOURCE/E_RESTART
SEQUENCE cold_restart ->E_RESTART.COLD(); END_SEQUENCE
SEQUENCE warm_restart ->E_RESTART.WARM(); END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_RS (* Event-driven bistable *)
EVENT_INPUT
  S; (* Set Event *)
  R; (* Reset Event *)
END_EVENT
EVENT_OUTPUT
  EO WITH Q; (* Output Event *)
END_EVENT
VAR_OUTPUT
  Q: BOOL; (* Current Output State *)
END_VAR
EC_STATES
  Q0; (* Q is FALSE initially *)
  RESET: RESET -> EO; (* Reset Q and issue EO *)
  SET: SET -> EO; (* Set Q and issue EO *)
END_STATES
EC_TRANSITIONS
  Q0 TO SET:= S;
  SET TO RESET:= R;
  RESET TO SET:= S;
END_TRANSITIONS
ALGORITHM SET IN ST: (* Set Q *)
Q:=TRUE;
END_ALGORITHM
ALGORITHM RESET IN ST: (* Reset Q *)
Q:=FALSE;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SELECT (* Selection between two events *)
```





```
END_FUNCTION_BLOCK================================================================
==========FUNCTION_BLOCKE_REND(*两个事件的集合*)


EI1;(*第一个事件输入*)
EI2;(*第二个事件输入*)
R;(*重置事件*)


EO;(*集合输出事件*)


开始;(*初始状态*)
EI1;(*EI1已到，等待EI2或R*)
EO: （*发出会合事件*）->EO；EI2;(*EI2已到，等待EI1或R*)




开始到EI1：=EI1；EI1开始：=R
；开始到EI2：=EI2;


EI1到EO:=EI2;EI2到EO:=EI1;
EO开始：=1；END_TRANSI
TIONS

END_FUNCTION_BLOCK================================================================
==========FUNCTION_BLOCKE_RESTART(*重新启动事件的生成*)


寒冷的;(*冷重启*)WARM;(*热重启*)

END_EVENT
SERVICE RESOURCE/E_RESTART
SEQUENCE cold_restart ->E_RESTART.COLD(); END_SEQUENCE
SEQUENCE warm_restart ->E_RESTART.WARM(); END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================================
                        * Event-driven bistable *)

小号；(*设置事件*)
R;(*重置事件*)


EO与Q；(*输出事件*)


问：布尔；(*当前输出状态*)


Q0;(*Q最初为FALSE*)
重置：重置->EO；(*重置Q并发出EO*)
设置：设置->EO；(*设置Q并发出EO*)
END_STATES


设置为复位：=R；重置为设
置：=S;

ST中的算法集：（*集Q*）


ST中的算法重置：（*重置Q*）


END_FUNCTION_BLOCK================================================================
==========FUNCTION_BLOCKE_SELECT（*两个事件之间的选择*）
```

```
EVENT_INPUT
  EI0 WITH G; (* Input event, selected when G=0 *)
  EI1 WITH G; (* Input event, selected when G=1 *)
END_EVENT
EVENT_OUTPUT EO; (* Output Event *)
END_EVENT
VAR_INPUT G: BOOL; (* Select EI0 when G=0, EI1 when G=1 *)
END_VAR
EC_STATES
  START;   (* Initial State *)
  EO: -> EO; (* Issue Output Event *)
END_STATES
EC_TRANSITIONS
  START TO EO:= EI0 [NOT G];
  START TO EO:= EI1 [G];
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SPLIT (* Split an event *)
EVENT_INPUT
  EI; (* Input event *)
END_EVENT
EVENT_OUTPUT
 EO1;  (* First output event *)
 EO2;  (* Second output event, etc. *)
END_EVENT
EC_STATES
  START; (* Initial State *)
  EO: (* Extensible *)
    ->EO1, (* Output first event *)
    ->EO2; (* Output second event, etc. *)
END_STATES
EC_TRANSITIONS
  START TO EO:= EI;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SWITCH (* Switch (demultiplex) an event *)
EVENT_INPUT EI WITH G; (* Event Input *)
END_EVENT
EVENT_OUTPUT
  EO0; (* Output, switched from EI when G=0 *)
  EO1; (* Output, switched from EI when G=1 *)
END_EVENT
VAR_INPUT G: BOOL; (* Switch EI to EI0 when G=0, to EI1 when G=1 *)
END_VAR
EC_STATES
  START; (* Initial State *)
  G0:  (* Issue EO0 when EI arrives with G=0 *)
    ->EO0;
  G1: (* Issue EO1 when EI arrives with G=1 *)
    ->EO1;
END_STATES
EC_TRANSITIONS
  START TO G0:= EI [NOT G];
  G0 TO START:= 1;
  START TO G1:= EI [G];
  G1 TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_TABLE (* Generation of a finite train of events, table
driven *)
EVENT_INPUT
 START WITH DT, N;
 STOP; (* Cancel *)
END_EVENT
```



61499-1 © IEC:2012 – 105 –

EI0与G；(*输入事件，当G=0时选择*)EI1WITHG;(*输入事件，当G=1时选择
*)
END_EVENTEVENT_OUTPUTEO；(*输出事件*)

VAR_INPUTG：布尔；(*G=0时选择EI0，G=1时选择EI1*)

开始;(*初始状态*)
EO：->EO；(*发出输出事件*)

开始到EO:=EI0[不是G];
开始到EO:=EI1[G];
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK==========
==========FUNCTION_BLOCKE_SPLIT（*拆分事件*）

EI；(*输入事件*)

EO1;(*第一个输出事件*)
EO2;(*第二个输出事件等*)

开始;(*初始状态*)
EO:(*可扩展*)->EO1 (*输出第一个事件*)->EO2;(*输出第二个事
件等*)END_STATES

开始到EO:=EI;
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK==========
==========FUNCTION_BLOCKE_SWITCH（*切换（解复用）事件*）

EVENT_INPUTEI与G；(*事件输入*)

EO0;(*输出，G=0时从EI切换*)EO1;(*输出，当G=1时从EI切换*)

VAR_INPUTG：布尔；(*当G=0时将EI切换到EI0，当G=1时切换到EI1*)

开始;(*初始状态*)
G0:(*当EI到达G=0*时发出EO0*)->EO0;G1：（*当EI到达时发出EO1，G
=1*）->EO1；END_STATES

开始到G0:=EI[非G];
G0开始：=1；开始到G1:=EI[G];

END_FUNCTION_BLOCK==========
======FUNCTION_BLOCKE_TABLE(*生成有限的事件序列，表驱动*)EVENT_INPUT

以DT、N开头；
 STOP; (* Cancel *)
END_EVENT

```
EVENT_OUTPUT
 EO WITH CV; (* N events at periods DT, starting at DT[0] after START *)
END_EVENT
VAR_INPUT
  DT: TIME[4]; (* Periods between events *)
  N: UINT; (* Number of events to generate *)
END_VAR
VAR_OUTPUT
  CV: UINT; (* Current event index, 0..N-1 *)
END_VAR
FBS
  CTRL: E_TABLE_CTRL;
  DLY: E_DELAY;
END_FBS
EVENT_CONNECTIONS
  START TO CTRL.INIT;
  CTRL.CLKO TO DLY.START;
  DLY.EO TO EO;
  DLY.EO TO CTRL.CLK;
  STOP TO DLY.STOP;
END_CONNECTIONS
DATA_CONNECTIONS
  DT TO CTRL.DT;
  N TO CTRL.N;
  CTRL.DTO TO DLY.DT;
  CTRL.CV TO CV;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_TABLE_CTRL (* Control for E_TABLE *)
EVENT_INPUT
 INIT WITH DT, N;
 CLK;
END_EVENT
EVENT_OUTPUT
 CLKO WITH DTO, CV;
END_EVENT
VAR_INPUT
 DT: TIME[4]; (* Array length is implementation dependent *)
 N: UINT;   (* Actual number of time steps *)
END_VAR
VAR_OUTPUT
 DTO: TIME;  (* Current delay interval *)
 CV: UINT;   (* Current event index, 0..N-1 *)
END_VAR
EC_STATES
 START;
 INIT0: INIT;
 INIT1: -> CLKO;
 STEP: STEP -> CLKO;
END_STATES
EC_TRANSITIONS
 START TO INIT0:= INIT;
 INIT0 TO INIT1:= [N>0];
 INIT0 TO START:= [N=0]; (* Don't run if N=0 *)
 INIT1 TO START:= 1;
 START TO STEP:= CLK [CV < MIN(3,N-1)];
 STEP TO START:= 1;
END_TRANSITIONS
ALGORITHM STEP IN ST:
  CV:= CV+1;
  DTO:= DT[CV];
END_ALGORITHM
ALGORITHM INIT IN ST:
  CV:= 0;
  DTO:= DT[0];
END_ALGORITHM
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_TRAIN (* Generation of a finite train of events *)
```





```
带有简历的EO；（*时间段DT的N个事件，从START之后的DT[0]开始*）


DT：时间[4]；(*事件之间的时间段*)
N：单位；(*要生成的事件数*)


简历：单位；(*当前事件索引，0..N-1*)
END_VAR
FBS
  CTRL：E_TABLE_CTRL;
  DLY：E_DELAY;


开始CTRL.INIT;
  CTRL.CLKO TO DLY.START;

                        ;
停下来DLY.STOP;
END_CONNECTIONS
DATA_CONNECTIONS
  DT TO CTRL.DT;
  N TO CTRL.N;
  CTRL.DTO TO DLY.DT;


END_FUNCTION_BLOCK(*=================================================================
==========FUNCTION_BLOCKE_TABLE_CTRL(*E_TABLE的控制*)


带DT的初始化，N;
 CLK;
END_EVENT
EVENT_OUTPUT
 CLKO WITH DTO, CV;


DT：时间[4]；(*数组长度取决于实现*)
N：单位；(*实际时间步数*)


DTO：时间；(*当前延迟间隔*)
简历：单位；(*当前事件索引，0..N-1*)
END_VAR
EC_STATES
 START;
 INIT0：INIT;
 INIT1：-> CLKO;
 STEP：STEP -> CLKO;


开始到INIT0:=INIT;

INIT0开始：=[N=0]；(*如果N=0则不运行*)

开始步：=CLK[CV<MIN(3 N-1)];
开始步骤：=1;

ST中的算法步骤：
  CV:= CV+1;


ST中的算法初始化：
  CV:= 0;


END_FUNCTION_BLOCK(*=================================================================
==========FUNCTION_BLOCKE_TRAIN(*生成有限的事件序列*)
```

```
 EVENT_INPUT
  START WITH DT, N;
  STOP;
 END_EVENT
 EVENT_OUTPUT
  EO WITH CV; (* N events at period DT, starting at DT after START *)
 END_EVENT
 VAR_INPUT
   DT: TIME; (* Period between events *)
   N: UINT; (* Number of events to generate *)
 END_VAR
 VAR_OUTPUT
  CV: UINT; (* EO index (0..N-1) *)
 END_VAR
 FBS
  CTR: E_CTU;
  GATE: E_SWITCH;
  DLY: E_DELAY;
 END_FBS
 EVENT_CONNECTIONS
  START TO CTR.R;
  STOP TO DLY.STOP;
  DLY.EO TO EO;
  DLY.EO TO CTR.CU;
  CTR.CUO TO GATE.EI;
  CTR.RO TO GATE.EI;
  GATE.EO0 TO DLY.START;
 END_CONNECTIONS
 DATA_CONNECTIONS
  DT TO DLY.DT;
  N TO CTR.PV;
  CTR.Q TO GATE.G;
  CTR.CV TO CV;
 END_CONNECTIONS
 END_FUNCTION_BLOCK
 ================================================================
 FUNCTION_BLOCK FB_ADD_INT (* INT Addition *)
 EVENT_INPUT
  REQ WITH QI, IN1, IN2;
 END_EVENT
 EVENT_OUTPUT
  CNF WITH QO, STATUS, OUT;
 END_EVENT
 VAR_INPUT
  QI: BOOL;   (* Event Qualifier *)
  IN1: INT;   (* Augend *)
  IN2: INT;   (* Addend *)
 END_VAR
 VAR_OUTPUT
  QO: BOOL;    (* Output Qualifier *)
  STATUS: UINT;  (* Operation Status *)
  OUT: INT;   (* Sum *)
 END_VAR
 VAR
  RESULT: DINT;
 END_VAR
 EC_STATES
  START;
  REQ: REQ -> CNF;
 END_STATES
 EC_TRANSITIONS
  START TO REQ:= REQ;
  REQ TO START:= 1;
 END_TRANSITIONS
 ALGORITHM REQ IN ST:
   QO:= QI;
   IF QI THEN
    STATUS:= 0;
    RESULT:= INT_TO_DINT(IN1) + INT_TO_DINT(IN2);
```



61499-1 © IEC:2012 – 107 –

以DT、N开头；
 STOP;

带有简历的EO；（*时间段DT的N个事件，从START之后的DT开始*）

DT：时间；(*事件之间的时间段*)
N：单位；(*要生成的事件数*)

简历：单位；(*EO指数(0..N-1)*)

```
END_VAR
FBS
 CTR: E_CTU;
 GATE: E_SWITCH;
 DLY: E_DELAY;
```

开始点击率；
停下来DLY.STOP;DLY.EO到E
O；

```
 DLY.EO TO CTR.CU;
 CTR.CUO TO GATE.EI;
 CTR.RO TO GATE.EI;
 GATE.EO0 TO DLY.START;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
 N TO CTR.PV;
 CTR.Q TO GATE.G;
```

END_FUNCTION_BLOCK==========
==========FUNCTION_BLOCKFB_ADD_INT(*INT添加*)

带QI、IN1、IN2的请求；

带QO、状态、输出的CNF；

问：布尔值；(*活动预选赛*)

```
 IN1: INT;   (* Augend *)
 IN2: INT;   (* Addend *)
```

QO：布尔值；(*输出限定符*)
状态：单位；(*运行状态*)

```
 OUT: INT;   (* Sum *)
END_VAR
VAR
 RESULT: DINT;
END_VAR
EC_STATES
 START;
 REQ: REQ -> CNF;
```

开始请求：=请求；
请求开始：=1；

ST中的算法请求：

如果齐那么

```
   STATUS:= 0;
   RESULT:= INT_TO_DINT(IN1) + INT_TO_DINT(IN2);
```

```
   IF (RESULT > 32767) OR (RESULT < -32768) THEN
    QO = FALSE;
    STATUS = 3;
    IF (RESULT > 32767) THEN OUT:= 32767;
    ELSE OUT:= -32768;
    END_IF;
   ELSE OUT:= RESULT;
   END_IF;
  ELSE STATUS = 1;
  END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK INTEGRAL_REAL
EVENT_INPUT
 INIT: INIT_EVENT WITH CYCLE;
 EX WITH HOLD, XIN;
END_EVENT
EVENT_OUTPUT
 INITO: INIT_EVENT WITH XOUT;
 EXO WITH XOUT;
END_EVENT
VAR_INPUT
 HOLD: BOOL;  (* 0 = Run, 1 = Hold *)
 XIN: REAL;  (* Integrand *)
 CYCLE: TIME; (* Sampling period *)
END_VAR
VAR_OUTPUT
 XOUT: REAL;  (* Integrated output *)
END_VAR
VAR DT: REAL; END_VAR
EC_STATES
 START;        (* EC Initial state *)
 INIT:INIT -> INITO; (* EC State with Algorithm and EC Action *)
 MAIN: MAIN -> EXO;
END_STATES
EC_TRANSITIONS
 START TO INIT:= INIT; (* An EC Transition *)
 START TO MAIN:= EX;
 INIT TO START:= 1;
 MAIN TO START:= 1;
END_TRANSITIONS
ALGORITHM INIT IN ST:
 XOUT:= 0.0;
 DT:= TIME_TO_REAL(CYCLE);
END_ALGORITHM
ALGORITHM MAIN IN ST:
 IF NOT HOLD THEN
  XOUT:= XOUT + XIN * DT;
 END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
ADAPTER LD_UNLD (* LOAD/UNLOAD Adapter Interface *)
EVENT_INPUT
 UNLD;   (* UNLOAD Request *)
END_EVENT
EVENT_OUTPUT
 LD WITH WO,WKPC; (* LOAD Request *)
 CNF WITH WO,WKPC; (* UNLD Confirm *)
END_EVENT
VAR_OUTPUT
 WO: BOOL;  (* Workpiece present *)
 WKPC: COLOR; (* Workpiece Color *)
END_VAR
SERVICE PLUG/SOCKET
SEQUENCE normal_operation
 PLUG.LD(WO,WKPC) -> SOCKET.LD(WO,WKPC);
 SOCKET.UNLD() -> PLUG.UNLD();
 PLUG.CNF() -> SOCKET.CNF();
```





```
如果（结果>32767）或（结果<-32768）那么
QO=假；状态=3；

如果（结果>32767）则输出：=32767；
    ELSE OUT:= -32768;
    END_IF;
                    T;

其他状态=1；
  END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK INTEGRAL_REAL

                     TH CYCLE;
EXWITHHOLD XIN;
END_EVENT

                  NT WITH XOUT;
带XOUT的EXO；


保持：布尔；（*0=运行，1=保持*）

周期;(*采样周期*)


XOUT：真实；(*综合输出*)
END_VAR


开始;(*EC初始状态*)
初始化：初始化->初始化；（*带有算法和EC操作的EC状态*）
 MAIN: MAIN -> EXO;


开始初始化：=初始化；(*EC过渡*)
开始主要：=EX；
初始化开始：=1；主要开始
：=1；

ST中的算法初始化：
 XOUT:= 0.0;
                        LE);

ST中的主要算法：
如果不持有则
  XOUT:= XOUT + XIN * DT;


END_FUNCTION_BLOCK==========================================================
==========ADAPTERLD_UNLD(*LOADUNLOADAdapterInterface*)


未定；(*卸载请求*)


LD与WO，WKPC；(*加载请求*)
CNF与WO WKPC;(*UNLD确认*)


WO：布尔值；(*存在工件*)WKPC：颜色；(*工件颜
色*)
END_VAR
SERVICE PLUG/SOCKET
SEQUENCE normal_operation
 PLUG.LD(WO,WKPC) -> SOCKET.LD(WO,WKPC);
 SOCKET.UNLD() -> PLUG.UNLD();
 PLUG.CNF() -> SOCKET.CNF();
```

```
END_SEQUENCE
END_SERVICE
END_ADAPTER
================================================================
FUNCTION_BLOCK MANAGER (* Management Service Interface *)
EVENT_INPUT
  INIT WITH QI, PARAMS; (* Service Initialization *)
  REQ WITH QI, CMD, OBJECT; (* Service Request *)
END_EVENT
EVENT_OUTPUT
  INITO WITH QO, STATUS; (* Initialization Confirm *)
  CNF WITH QO, STATUS, RESULT; (* Service Confirmation *)
END_EVENT
VAR_INPUT
  QI: BOOL; (* Event Input Qualifier *)
  PARAMS: WSTRING; (* Service Parameters *)
  CMD: UINT; (* Enumerated Command *)
  OBJECT: BYTE[512]; (* Command Object *)
END_VAR
VAR_OUTPUT
  QO: BOOL; (* Event Output Qualifier *)
  STATUS: UINT; (* Service Status *)
  RESULT: BYTE[512]; (* Result Object *)
END_VAR
SERVICE MANAGER/resource
SEQUENCE normal_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement() -> MANAGER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement(PARAMS) -> MANAGER.INITO-
(STATUS);
END_SEQUENCE
SEQUENCE normal_command_sequence
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.CNF+(STATUS,RESULT);
END_SEQUENCE
SEQUENCE command_error
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.IND-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
  MANAGER.INIT-() -> resource.terminateService() -> MANAGER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
  resource.serviceTerminated(STATUS) -> MANAGER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK PI_REAL
EVENT_INPUT
  INIT WITH KP, KI, CYCLE;
  EX WITH HOLD, PV, SP, KP, KI, CYCLE;
END_EVENT
EVENT_OUTPUT
  INITO WITH XOUT;
  EXO WITH XOUT;
END_EVENT
VAR_INPUT
  HOLD: BOOL; (* Hold when TRUE *)
  PV: REAL; (* Process variable *)
  SP: REAL; (* Set point *)
  KP: REAL; (* Proportionality constant *)
  KI: REAL; (* Integral constant,1/s *)
  CYCLE: TIME; (* Sampling period *)
END_VAR
VAR_OUTPUT
  XOUT: REAL;
END_VAR
```



61499-1 © IEC:2012 – 109 –

END_SERVICEEND_ADAPTER==========
==========FUNCTION_BLOCKMANAGER(*管理服务接口*)

用QI、PARAMS初始化；(*服务初始化*)
带QI、CMD、OBJECT的请求；（*服务请求*）

使用QO、状态初始化；(*初始化确认*)
带有QO、状态、结果的CNF；(*服务确认*)

问：布尔值；(*事件输入限定符*)
参数：WSTRING；(*服务参数*)
CMD：单位；(*枚举命令*)
对象：字节[512]；(*命令对象*)

QO：布尔值；(*事件输出限定符*)STATUS:UINT;(*服务状态*)

结果：字节[512]；(*结果对象*)

```
END_VAR
SERVICE MANAGER/resource
SEQUENCE normal_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement() -> MANAGER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement(PARAMS) -> MANAGER.INITO-
(STATUS);
END_SEQUENCE
SEQUENCE normal_command_sequence
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.CNF+(STATUS,RESULT);
END_SEQUENCE
SEQUENCE command_error
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.IND-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
  MANAGER.INIT-() -> resource.terminateService() -> MANAGER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
  resource.serviceTerminated(STATUS) -> MANAGER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
```

用KP、KI、CYCLE初始化；
EXWITHHOLD、PV、SP、KP、KI、CYCLE；
END_EVENT

;
带XOUT的EXO；

保持：布尔；(*TRUE时保持*)PV:REAL;(*过程变量*)

SP：真实；（*设定点*）
KP：真实；(*比例常数*)
*)
周期;(*采样周期*)

```
END_VAR
VAR_OUTPUT
  XOUT: REAL;
END_VAR
```

```
FBS
  CALC: PID_CALC;
  INTEGRAL_TERM: INTEGRAL_REAL;
END_FBS
EVENT_CONNECTIONS
  INIT TO CALC.INIT;
  EX TO CALC.PRE;
  CALC.POSTO TO EXO;
  INTEGRAL_TERM.INITO TO INITO;
  CALC.INITO TO INTEGRAL_TERM.INIT;
  CALC.PREO TO INTEGRAL_TERM.EX;
  INTEGRAL_TERM.EXO TO CALC.POST;
END_CONNECTIONS
DATA_CONNECTIONS
  HOLD TO INTEGRAL_TERM.HOLD;
  PV TO CALC.PV;
  SP TO CALC.SP;
  KP TO CALC.KP;
  KI TO CALC.KI;
  CYCLE TO INTEGRAL_TERM.CYCLE;
  CALC.XOUT TO XOUT;
  CALC.ETERM TO INTEGRAL_TERM.XIN;
  INTEGRAL_TERM.XOUT TO CALC.ITERM;
  0 TO CALC.TD;
  0 TO CALC.DTERM;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=========================================================================*)
SUBAPPLICATION PI_REAL_APPL (* A Subapplication *)
EVENT_INPUT
 INIT;
 EX;
END_EVENT
EVENT_OUTPUT
 INITO;
 EXO;
END_EVENT
VAR_INPUT
 HOLD: BOOL; (* Hold when TRUE *)
 PV: REAL; (* Process variable *)
 SP: REAL; (* Set point *)
 KP: REAL; (* Proportional gain *)
 KI: REAL; (* Integral gain = Sample period/Reset time *)
 X0: REAL; (* Initial integrator output *)
END_VAR
VAR_OUTPUT XOUT: REAL; END_VAR
FBS
 ETERM: FB_SUB_REAL;
 INTEGRATOR: ACCUM_REAL;
 CALC: PI_CALC;
END_FBS
EVENT_CONNECTIONS
  INIT TO INTEGRATOR.INIT;
  INTEGRATOR.INITO TO INITO;
  EX TO ETERM.REQ;
  ETERM.CNF TO INTEGRATOR.EX;
  INTEGRATOR.EXO TO CALC.EX;
  CALC.EXO TO EXO;
END_CONNECTIONS
DATA_CONNECTIONS
  X0 TO INTEGRATOR.X0;
  HOLD TO INTEGRATOR.HOLD;
  PV TO ETERM.IN1;
  SP TO ETERM.IN2;
  KP TO CALC.KP;
  KI TO CALC.KI;
  ETERM.OUT TO INTEGRATOR.XIN;
  ETERM.OUT TO CALC.ETERM;
  INTEGRATOR.XOUT TO CALC.ITERM;
  CALC.XOUT TO XOUT;
```



– 110 – 61499-1 © IEC:2012

```
FBS
  CALC: PID_CALC;
                      GRAL_REAL;

EVENT_CONNECTIONSINIT到C
ALC.INIT；EXTOCALC.PRE；

  CALC.POSTO TO EXO;
  INTEGRAL_TERM.INITO TO INITO;
  CALC.INITO TO INTEGRAL_TERM.INIT;
  CALC.PREO TO INTEGRAL_TERM.EX;
                                ST;

DATA_CONNECTIONS保持到INTEGRAL_TER
M.HOLD；
PV到CALC.PV;SP到CALC.
SP;KP到CALC.KP;KITOCA
LC.KI;

循环到INTEGRAL_TERM.CYCLE；
  CALC.XOUT TO XOUT;
  CALC.ETERM TO INTEGRAL_TERM.XIN;
  INTEGRAL_TERM.XOUT TO CALC.ITERM;
  0 TO CALC.TD;
  0 TO CALC.DTERM;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=========================================================================*)
SUBAPPLICATION PI_REAL_APPL (* A Subapplication *)
EVENT_INPUT
 INIT;
 EX;
END_EVENT
EVENT_OUTPUT
 INITO;
 EXO;


保持：布尔；(*TRUE时保持*)PV:REAL;(*过程变量*
)
SP：真实；  (*设定点*)
KP：真实；(*比例增益*)
KI：真实的；(*积分增益=采样周期复位时间*)
XO：真实；(*初始积分器输出*)
END_VAR
VAR_OUTPUT XOUT: REAL; END_VAR
FBS
 ETERM: FB_SUB_REAL;
 INTEGRATOR: ACCUM_REAL;
 CALC: PI_CALC;


初始化到积分器.INIT；
                        TO INITO;
EXTOETERM.REQ；
  ETERM.CNF TO INTEGRATOR.EX;
  INTEGRATOR.EXO TO CALC.EX;
  CALC.EXO TO EXO;
END_CONNECTIONS


持有积分。持有；
                    ;
                    ;
KP到CALC.KP;KITOCALC.
KI;
  ETERM.OUT TO INTEGRATOR.XIN;
  ETERM.OUT TO CALC.ETERM;
  INTEGRATOR.XOUT TO CALC.ITERM;
  CALC.XOUT TO XOUT;
```

```
  1 TO ETERM.QI;
END_CONNECTIONS
END_SUBAPPLICATION
================================================================
FUNCTION_BLOCK REQUESTER
          (* Service Requester Interface *)
EVENT_INPUT
 INIT WITH QI, PARAMS;    (* Service Initialization *)
 REQ WITH QI, SD_1, SD_m;  (* Service Request *)
END_EVENT
EVENT_OUTPUT
 INITO WITH QO, STATUS;      (* Initialization Confirm *)
 CNF WITH QO, STATUS, RD_1, RD_n; (* Service Confirmation *)
END_EVENT
VAR_INPUT
 QI: BOOL;   (* Event Input Qualifier *)
 PARAMS: ANY; (* Service Parameters *)
 SD_1: ANY;  (* Data to transfer, extensible *)
 SD_m: ANY;  (* Last data item to transfer *)
END_VAR
VAR_OUTPUT
 QO: BOOL;   (* Event Output Qualifier *)
 STATUS: ANY; (* Service Status *)
 RD_1: ANY;  (* Received data, extensible *)
 RD_n: ANY;  (* Last received data item *)
END_VAR
SERVICE REQUESTER/RESOURCE
SEQUENCE normal_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE normal_data_transfer
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF+(RD_1,...,RD_n);
END_SEQUENCE
SEQUENCE data_transfer_error
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
 REQUESTER.INIT-() -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
 -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK XBAR_MVCA (* XBAR_MVC + Adapters *)
EVENT_INPUT
 INIT WITH VF,VR,DTL,DT,BKGD,LEN,DIA,DIR; (* Initialize *)
END_EVENT
EVENT_OUTPUT
 INITO;
END_EVENT
VAR_INPUT
 VF: INT:= 20;    (* ADVANCE speed in +%/s *)
 VR: INT:= -40;    (* RETRACT speed in -%/s *)
 DTL: TIME:= t#750ms; (* LOAD Delay *)
 DT: TIME:= t#250ms; (* Simulation Interval *)
 BKGD: COLOR:= COLOR#blue; (* Transfer Bar Color *)
 LEN: UINT:= 5;  (* Bar Length in Diameters *)
 DIA: UINT:= 20; (* Workpiece diameter *)
 DIR: UINT;     (* Orientation: 0=L/R, 1=T/B, 2=R/L, 3=B/T *)
END_VAR
SOCKETS
 LDU_SKT: LD_UNLD;
END_SOCKETS
PLUGS
```



61499-1 © IEC:2012 – 111 –

```
END_SUBAPPLICATION================================================================
=========FUNCTION_BLOCKREQUESTER(*ServiceRequesterInterface*)EVENT_INPUT
```

用QI、PARAMS初始化；(*服务初始化*)
请求QI、SD_1、SD_m；  (*服务请求*)

使用QO、状态初始化；(*初始化确认*)
带有QO、状态、RD_1、RD_n的CNF；(*服务确认*)

问：布尔值；(*事件输入限定符*)
参数：任何；(*服务参数*)
SD_1：任何；(*要传输的数据，可扩展*)
SD_m：任何；(*要传输的最后一个数据项*)

QO：布尔值；(*事件输出限定符*)
状态：任何；(*服务状态*)
RD_1：任何；(*接收到的数据，可扩展*)
RD_n：任何；(*最后收到的数据项*)

```
END_VAR
SERVICE REQUESTER/RESOURCE
SEQUENCE normal_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE normal_data_transfer
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF+(RD_1,...,RD_n);
END_SEQUENCE
SEQUENCE data_transfer_error
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
 REQUESTER.INIT-() -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
 -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
```

用VF VR DTL DT BKGD LEN DIA DIR初始化；(*初始化*)

```
END_EVENT
EVENT_OUTPUT
 INITO;
```

VF：INT：=20；(*ADVANCE速度+%s*)
虚拟现实：智力：=-40；(*RETRACTspeedin-%s*)DTL:TIME:=t#750
ms;(*加载延迟*)
DT:时间:=t#250ms;(*模拟间隔*)BKGD:COLOR:=COLOR#blue;(*转移条颜色*)

LEN：UINT：=5；(*棒材长度，以直径为单位*)

```
 DIA: UINT:= 20; (* Workpiece diameter *)
 DIR: UINT;     (* Orientation: 0=L/R, 1=T/B, 2=R/L, 3=B/T *)
END_VAR
SOCKETS
 LDU_SKT: LD_UNLD;
END_SOCKETS
PLUGS
```

```
 LDU_PLG: LD_UNLD;
END_PLUGS
FBS
 MVC: XBAR_MVC;
END_FBS
EVENT_CONNECTIONS
  INIT TO MVC.INIT;
  MVC.INITO TO INITO;
  MVC.LOADED TO LDU_SKT.UNLD;
  LDU_SKT.LD TO MVC.LOAD;
  MVC.ADVANCED TO LDU_PLG.LD;
  LDU_PLG.UNLD TO MVC.UNLOAD;
  MVC.UNLOADED TO LDU_PLG.CNF;
END_CONNECTIONS
DATA_CONNECTIONS
  LDU_SKT.WO TO MVC.WI;
  LDU_SKT.WKPC TO MVC.LDCOL;
  MVC.WO TO LDU_PLG.WO;
  MVC.WKPC TO LDU_PLG.WKPC;
  VF TO MVC.VF;
  VR TO MVC.VR;
  DTL TO MVC.DTL;
  DT TO MVC.DT;
  BKGD TO MVC.BKGD;
  LEN TO MVC.LEN;
  DIA TO MVC.DIA;
  DIR TO MVC.DIR;
END_CONNECTIONS
END_FUNCTION_BLOCK
==========================================================================
```



– 112 – 61499-1 © IEC:2012

```
 LDU_PLG: LD_UNLD;
END_PLUGS
FBS
 MVC: XBAR_MVC;
```

初始化到MVC.INIT;

```
  MVC.INITO TO INITO;
  MVC.LOADED TO LDU_SKT.UNLD;
  LDU_SKT.LD TO MVC.LOAD;
  MVC.ADVANCED TO LDU_PLG.LD;
  LDU_PLG.UNLD TO MVC.UNLOAD;
  MVC.UNLOADED TO LDU_PLG.CNF;
END_CONNECTIONS
DATA_CONNECTIONS
  LDU_SKT.WO TO MVC.WI;
  LDU_SKT.WKPC TO MVC.LDCOL;
  MVC.WO TO LDU_PLG.WO;
  MVC.WKPC TO LDU_PLG.WKPC;
  VF TO MVC.VF;
  VR TO MVC.VR;
  DTL TO MVC.DTL;
```

;

LEN到MVC.LEN;DIA转MVC.DIA；目录到MVC.DIR;

```
END_CONNECTIONS
END_FUNCTION_BLOCK
==========================================================================
```

# Annex G
(informative)

# Attributes

## G.1 General principles

*Attributes* may be associated with *data types, variables, applications*, and *types* and *instances* of *function blocks, devices, resources*, and their component elements. Attributes have values that may be modified and accessed at various points in the life cycle of the function block type or instance.

In addition to the descriptions of function block *algorithms*, supplementary information is necessary to support the use of a function block during the course of its software life cycle. This information may be provided by attaching *attributes* to the component elements of function block *types* or *instances*.

Attributes can be applied to elements such as *data types, variables*, and *parameters* that are used in the specification of function block types or instances. Graphical language elements may require additional attributes for holding information such as position, color, size, etc.

Attributes can also be applied directly to function block types and instances, for instance to hold the version of a function block type specification.

Certain attributes may be used throughout the life cycle of a function block. For instance, an attribute related to a function block type specification may be accessed when the function block type is selected from a library, when an instance of the function block type is queried, etc.

Other attributes may only exist at certain points in the life cycle. For instance, text defining the purpose of a particular function block instance might be applied only when the function block is instantiated, and might be modified during the life of the function block instance.

Certain function block attributes may be installed in associated *resources* and be accessible during the lifetime of the distributed *application*. Such attributes are typically used to support access to function block parameter values by external devices, e.g., to restrict the values of parameters that may be set using a hand-held configurator to predefined safe limits.

## G.2 Attribute definitions

An attribute definition provides the information specified in Table G.1. Each attribute has a name and a data type of its associated value. An attribute may have a default value that will be used until a value is given at some point in the software life cycle. In the example given in G.1, the `DESCRIPTION` attribute has an initial value of '' (the empty string) that may be overwritten with a more meaningful description when a function block instance is configured or even during its active use.

Attributes themselves may require additional information to that shown in Table G.1. Such information is designated as *sub-attributes*.



61499-1 © IEC:2012 – 113 –

Annex G
(informative)

Attributes

## 一般原则

属性可以与数据类型、变量、应用程序以及功能块、设备、资源及其组件元素的类型和实例相关联。属性具有可以在功能块类型或实例的生命周期中的各个点进行修改和访问的值。

除了功能块算法的描述外，还需要补充信息来支持功能块在其软件生命周期过程中的使用。可以通过将属性附加到功能块类型或实例的组件元素来提供此信息。

属性可应用于功能块类型或实例规范中使用的数据类型、变量和参数等元素。图形语言元素可能需要附加属性来保存位置、颜色、大小等信息。

属性也可以直接应用于功能块类型和实例，例如保存功能块类型规范的版本。

某些属性可以在功能块的整个生命周期中使用。例如，当从库中选择功能块类型、查询功能块类型的实例等时，可以访问与功能块类型规范相关的属性。

其他属性可能只存在于生命周期中的某些点。例如，定义特定功能块实例用途的文本可能仅在功能块被实例化时应用，并且可能在功能块实例的生命周期内被修改。

某些功能块属性可以安装在相关资源中，并且可以在分布式应用程序的生命周期内访问。此类属性通常用于支持外部设备对功能块参数值的访问，例如，将可以使用手持配置器设置的参数值限制为预定义的安全限制。

## 属性定义

属性定义提供表G.1中规定的信息。每个属性都有其关联值的名称和数据类型。一个属性可能有一个默认值，直到在软件生命周期的某个时间点给出一个值为止。在G.1中给出的示例中，DESCRIPTION属性具有初始值''（空字符串），当配置功能块实例时，甚至在其活动使用期间，可能会被更有意义的描述覆盖。

属性本身可能需要表G.1所示的附加信息。此类信息被指定为子属性。

**Table G.1 – Elements of attribute definitions**

| Element | Example | |
|---|---|---|
| **Name** | DESCRIPTION | |
| **Data type** | WSTRING(30) | |
| **Default value** | "" | |
| **Associated element** | **Function block types** | **Function block instances** |
| **Usage** | Configuration | Run-time |

## G.3 Examples

NOTE The following examples are for the purpose of illustrating the use of attributes and are not to be considered as normative definitions of standard attributes.

An example of a *data type attribute* is:

- `Max_System_Value` - This attribute defines the maximum supported value of a numeric data type. It is applied to the generic data type `ANY_NUM`, so that all numeric types such as `INT` and `REAL` will inherit this attribute. Note that each specific data type will have its own value for this attribute, and that standard values for this attribute for some data types are given in Table E.1.

Examples of attributes that apply to *variables* are:

- `Diagnostic_Access` – This determines whether the value of a variable is accessible by a run-time diagnostic system.
- `Write_Access` – This defines the access level required to change the value of a variable, e.g., `'Operator', 'System', 'Diagnostics'`.
- `Units` – The dimensional units that apply to a variable, e.g., `'l', 'm/s', 'cm'`.
- `Usage` – A multi-line textual description of the usage of the associated variable.

Examples of function block type attributes are:

- `Usage_Class` – This describes the general usage of the function block, e.g., `'Input', 'Output', 'Control'`.
- `Version` – This describes the version number of the function block type definition, e.g., `'1.2'`.
- `Help` – A multi-line textual description that may be accessed at various points in the life cycle.

Attributes which are relevant to the scheduling of *algorithms* for *execution* include:

- `ExecutionTime` – This attribute, of type `TIME`, specifies the worst-case time for execution of a particular *algorithm* of a specified *function block type* in a particular *resource type*.
- `Priority` – This attribute is associated with a particular *event connection* within a *resource*, and may be inherited from the *resource type*. This attribute may be used by a resource which supports pre-emptive *multitasking* to determine the priority of *execution* of an *algorithm* invoked by an *EC action* associated with an *EC state* which is activated by an event with the specified priority.





**表G.1 属性定义的元素**

| Element | Example | |
|---|---|---|
| | DESCRIPTION | |
| **数据类型** | WSTRING(30) | |
| **默认值** | " | |
| **关联元素** | **功能块类型** | **功能块实例** |
| **Usage** | Configuration | Run-time |

注：以下示例用于说明属性的使用，不应视为标准属性的规范性定义。

数据类型属性的一个示例是：

- Max_System_Value此属性定义数字数据类型的最大支持值。它应用于通用数据类型ANY_NUM，因此所有数字类型（例如INT和REAL）都将继承此属性。请注意，每个特定的数据类型都有自己的该属性值，并且某些数据类型的该属性的标准值在表E.1中给出。

适用于变量的属性示例如下：

- Diagnostic_Access——这决定了一个变量的值是否可以被运行时诊断系统访问。
- Write_Access 这定义了更改变量值所需的访问级别，例如“操作员”、“系统”、“诊断”。
- 单位–适用于变量的维度单位，例如，'l'、'ms'、'cm'。
  用法–关联变量用法的多行文本描述。

功能块类型属性的示例如下：

- Usage_Class 这描述了功能块的一般用法，例如，“输入”、“输出”、“控制”。
- 版本——这描述了功能块类型定义的版本号，例如，“1.2”。
- 帮助–可以在生命周期的不同点访问的多行文本描述。

与执行算法调度相关的属性包括：

- ExecutionTime 此属性，类型为TIME，指定在特定资源类型中执行特定功能块类型的特定算法的最坏情况时间。
- 优先级——该属性与资源中的特定事件连接相关联，并且可以从资源类型继承。支持抢先式多任务的资源可以使用该属性来确定由与EC状态相关联的EC动作调用的算法的执行优先级，该EC状态由具有指定优先级的事件激活。

## G.4 Attribute sources

Attributes may come from the following main sources:

- **Implicit** attributes such as function block *type names*, *instance names*, *variable names* and their *data types*, are defined as part of the normal *declaration* process for the function block.
- **Standard** attributes are those which are required as part of a standard, such function block type versions, maximum range of parameters, parameter descriptions, etc.
- **Product-specific** attributes are those which a system vendor has provided, such as function block type product codes, hardware addresses of function block instances, etc.
- **Application-specific** attributes are those which a system developer specifies to support the use of a particular data type or function block in an application, such as an additional function block instance identifier to fit a customer's desired style, a fail-safe default value for output parameters, an alternative parameter description in a national language, etc.

## G.5 Attribute inheritance

Function block elements will inherit attributes from more primitive elements. For instance, a *variable* within a *function block type declaration* will inherit attributes of its associated *data type*, and a function block *instance* will inherit attributes of the associated function block *type*.

*Data types* will inherit attributes down the generic type hierarchy defined in IEC 61131-3. For example, attributes applied to ANY_REAL will also apply to LREAL and REAL.

## G.6 Declaration syntax

The assignment of an attribute value to a declared element is similar to assigning a value to an *instance* of an *attribute type* in which the instance has the same name as the type.

The declaration of an attribute *type* uses the same syntax as the declaration of a *data type* as defined in IEC 61131-3, with the exception that the delimiting keywords are `ATTRIBUTE...END_ATTRIBUTE` instead of `TYPE...END_TYPE`. For instance, the declaration of the attribute type `DESCRIPTION` in Table G.1 would be:

```
ATTRIBUTE DESCRIPTION: WSTRING(30); END_ATTRIBUTE
```

The assignment of a value to an attribute *instance* uses the same syntax as that for assigning an initial value to a *variable* as described in IEC 61131-3, with the following extensions:

a) the name of the attribute instance is the same as the name of the corresponding attribute type;

b) no data type is specified for the attribute instance;

c) the value assignment is enclosed in the **pragma** construct defined in IEC 61131-3;

d) multiple attribute value assignments, separated by semicolons, may be included in the pragma construct;

e) the pragma construct shall be located in such a manner that the declaration to which it applies can be determined unambiguously.

An example of the application of these rules is:





## 属性来源

属性可能来自以下主要来源：

- 隐式属性，例如功能块类型名称、实例名称、变量名称及其数据类型，被定义为功能块正常声明过程的一部分。
- 标准属性是作为标准的一部分所必需的属性，例如功能块类型版本、参数的最大范围、参数描述等。
- 产品特定属性是系统供应商提供的属性，例如功能块类型产品代码、功能块实例的硬件地址等。
- 特定于应用程序的属性是系统开发人员为支持在应用程序中使用特定数据类型或功能块而指定的属性，例如满足客户所需样式的附加功能块实例标识符、输出的故障安全默认值参数、以国家语言编写的替代参数描述等。

## 属性继承

功能块元素将继承更多原始元素的属性。例如，功能块类型声明中的变量将继承其关联数据类型的属性，而功能块实例将继承关联功能块类型的属性。

数据类型将继承IEC61131-3中定义的通用类型层次结构的属性。例如，应用于ANY_REAL的属性也将应用于LREAL和REAL。

## 声明语法

将属性值分配给已声明的元素类似于将值分配给属性类型的实例，其中该实例与该类型具有相同的名称。

属性类型的声明使用与定义的数据类型声明相同的语法
关键字是ATTRIBUTE...END_ATTRIBUTE而不是TYPE...END_TYPE。例如，表G.1中的属性类型DESCRIPTION的声明将是：

将值分配给属性实例使用的语法与IEC61131-3中描述的为变量分配初始值的语法相同，但具有以下扩展：

a)属性实例的名称与对应的属性类型的名称相同；

b)没有为属性实例指定数据类型；

c)赋值包含在IEC61131-3中定义的pragma结构中；

d)多个属性值分配，用分号分隔，可以包含在pragma构造中；

e)pragma构造的位置应使其适用的声明可以明确确定。

应用这些规则的一个例子是：

```
FUNCTION_BLOCK PID
     {DESCRIPTION:= "Proportional + Integral + Derivative Control;
      AUTHOR:= "JHC"; VERSION:= "19990103/JHC"}
     INPUT_EVENT
      INIT WITH QI, PARAMS; {DESCRIPTION:= "Initialization Request"}
     ...etc.
```



– 116 – 61499-1 © IEC:2012

```
FUNC
     {描述：="比例+积分+微分控制；作者：="JHC"；版本：="19990103JHC"

     用QI、PARAMS初始化；{DESCRIPTION:="初始化请求" …等。
```

## Bibliography

IEC 60050-351:2006, *International Electrotechnical Vocabulary – Part 351: Control technology*

IEC 61131-5:2000, *Programmable controllers – Part 5: Communications*

IEC 61499 (all parts), *Function blocks*

IEC 61499-2:2012, *Function blocks – Part 2: Software tools requirements*

IEC 61499-4, *Function blocks – Part 4: Rules for compliance profiles*

ISO/IEC 7498-4, *Information processing systems – Open systems interconnection – Basic reference model – Part 4: Management framework*

ISO/IEC 8825-1:2008, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

ISO/IEC 10040:1998, *Information technology – Open Systems Interconnection – Systems management overview*

ISO/IEC/IEEE 60559, *Information technology – Microprocessor systems – Floating-point arithmetic*

ISO 2382 (all parts), *Information technology – Vocabulary*



61499-1 © IEC:2012 – 117 –

## Bibliography

IEC 60050-351:2006, *International Electrotechnical Vocabulary – Part 351: Control*

IEC61131-5:2000，可编程控制器 第5部分：通信

IEC61499（所有部分），功能块

IEC61499-2:2012，功能块-第2部分：软件工具要求

IEC61499-4，功能块 第4部分：合规性配置文件规则

ISOIEC7498-4，信息处理系统-开放系统互连-基本参考模型-第4部分：管理框架

ISOIEC8825-1:2008，信息技术 ASN.1编码规则：基本编码规则(BER)、规范编码规则(CER)和可区分编码规则(DER)规范

ISOIEC10040:1998，信息技术 开放系统互连 系统管理概述

ISOIECIEEE60559，信息技术 微处理器系统 浮点运算

ISO2382（所有部分），信息技术 词汇

# SOMMAIRE





– 118 – 61499-1 © CEI:2012

# SOMMAIRE

61499-1 © CEI:2012 – 119 –

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

# BLOCS FONCTIONNELS –

# Partie 1: Architecture

## AVANT-PROPOS

1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Norme internationale CEI 61499-1 a été établie par le sous-comité 65B: Equipements de mesure et de contrôle-commande, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition parue en 2005. Elle constitue une révision technique.

Cette édition inclut les modifications techniques majeures suivantes par rapport à l'édition précédente:

- Le terme *contrôle d'exécution* dans les blocs fonctionnels de base (5.2) a été clarifié et étendu:
  - Les parties dynamiques et statiques de la condition de transition EC sont clairement délimitées par l'utilisation de la syntaxe `ec_transition_event[guard_condition]` du langage de modélisation unifié (UML) (5.2.1.3, B.2.1).



– 122 – 61499-1 © CEI:2012

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

# B –

# 第1部分：架构

1）国际电工委员会（IEC）是一个由所有国家电工委员会（IECNationalCommittees）组成的世界性标准化组织。IEC的目的是促进电力和电子领域所有标准化问题的国际合作。为此，IEC在其他活动中发布了国际标准、技术规范、技术报告、公开可用规范(PAS)和指南（以下简称"IEC出版物"）。他们的阐述委托给研究委员会，任何对所处理的主题感兴趣的国家委员会都可以参加这些委员会的工作。与IEC联络的国际组织、政府和非政府组织也参与了这项工作。IEC与国际标准化组织(ISO)根据两个组织之间协议确定的条件密切合作。

2)由于IEC国家委员会在每个技术委员会中都有代表，因此正式的IEC决定或关于技术问题的协议尽可能代表所考虑主题的国际协议。

3)IEC出版物采用国际推荐的形式，并被IEC国家委员会接受。已尽一切合理努力确保IEC确保其出版物技术内容的准确性；IEC不对任何最终用户的任何误用或解释负责。

4)为了促进国际统一，IEC国家委员会承诺尽可能在其国家和地区出版物中透明地应用IEC出版物。任何IEC出版物与任何相应的国家或地区出版物之间的任何差异都应在后者中明确指出。

5)IEC本身不提供任何符合性证明。独立的认证机构提供合格评定服务，并在某些领域获得IEC合格标志。IEC不对独立认证机构提供的任何服务负责。

6)所有用户应确保他们拥有本出版物的最新版本。

7)对于因人身伤害、财产损失或任何形式的其他直接或间接损害，或承担因出版或使用IEC的本出版物或任何其他IEC出版物或给予它的信用而产生的成本（包括法律费用）和开支。

8)请注意本出版物中引用的规范性参考文献。为了正确应用本出版物，必须使用参考出版物。

9)请注意，本IEC出版物的某些元素可能是专利权的主题。IEC对未能识别此类专利权和未披露其存在不承担任何责任。

国际标准IEC61499-1由IEC技术委员会65：测量和控制设备分委员会65B：工业过程中的测量、控制和自动化制定。

本次第二版取消并代替了2005年出版的第一版。它构成技术修订。

此版本相对于上一版本包括以下主要技术更改：

- 基本功能块(5.2)中的术语执行控制已得到澄清和扩展：
  - EC转换条件的动态和静态部分使用统一建模语言(UML)语法ec_transition_event[guard_condition](5.2.1.3 B.2.1)清楚地描述。

- La terminologie "franchissement d'une transition EC" est utilisée de préférence à "libération" (3.10) afin d'éviter toute erreur d'interprétation qui laisserait supposer que l'ensemble de la condition de transition correspond à une variable booléenne pouvant être "libérée."
- L'exploitation du diagramme d'états ECC au 5.2.2.2 a été clarifiée et rendue plus rigoureuse.
- Les sorties d'événements et les sorties de données des instances d'adaptateur (prises mâles et prises femelles) peuvent être utilisées dans les conditions de transition EC, et les entrées d'événements des instances d'adaptateur peuvent être utilisées en tant que sorties d'actions EC.

- Les *variables temporaires* (3.97) peuvent être déclarées (B.2.1) et utilisées dans les algorithmes des blocs fonctionnels de base.
- Les *séquences de service* (6.1.3) peuvent à présent être définies pour les types bloc fonctionnel de base et composé et les types adaptateur, ainsi que les types interface de service.
- La syntaxe pour le *mapping* des instances FB entre les applications et les ressources a été simplifiée (Article B.3).
- La syntaxe relative à la définition des *types de segment* (7.2.3) pour les segments de réseau des configurations de système a été ajoutée (Article B.3).
- Les types de blocs fonctionnels pour l'interfonctionnement avec des dispositifs de commande programmables sont définis (Article D.6).
- Les commandes de gestion READ/WRITE (Tableau 8) s'appliquent désormais uniquement aux *paramètres*.

Le texte de cette partie de la CEI 61499 est issu des documents suivants:

| FDIS | Rapport de vote |
|---|---|
| 65B/845/FDIS | 65B/855/RVD |

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme (à la fin du vote).

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61499, présentées sous le titre général *Blocs fonctionnels*, peut être consultée sur le site web de la CEI.

Les termes utilisés dans la présente Norme internationale, définis à l'Article 3, sont présentés en *italiques*.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "http://webstore.iec.ch" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

**IMPORTANT – Le logo *"colour inside"* qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.**





- 术语"跨越EC转换"优先于"释放"（3.10）使用，以避免任何误解，这表明整个转换条件对应于可以"释放"的布尔变量。
- 5.2.2.2中ECC状态图的使用已经明确，更加严谨。
- 适配器实例（引脚插头和插座）的事件输出和数据输出可用于EC转换条件，适配器实例的事件输入可用作输出EC份额。

- 临时变量(3.97)可以声明(B.2.1)并用于基本功能块算法。
- 现在可以为基本和复合功能块类型和适配器类型以及服务接口类型定义服务序列(6.1.3)。
- 在应用程序和资源之间映射FB实例的语法已得到简化（第B.3条）。
- 增加了为系统配置的网段定义段类型（7.2.3）的语法（条款B.3）。
- 定义了与可编程控制设备互通的功能块类型（条款D.6）。
- READWRITE处理命令（表8）现在仅适用于参数。

IEC61499本部分的文本基于以下文件：

| FDIS | 投票报告 |
|---|---|
| | |

上表所示的投票报告提供了导致批准本标准的所有投票信息（在投票结束时）。

本出版物是根据ISOIEC指令第2部分起草的。

IEC61499系列所有部分的列表，在通用标题下显示功能块，可以在IEC网站上找到。

本国际标准中使用的术语，在第3章中定义，以斜体显示。

委员会决定本出版物的内容将保持不变，直到IEC网站"http:webstore.iec.ch"下有关特定出版物的数据中指明的稳定日期。在这一天，该出版物将

- 
- 
- 被修订版取代，或
- amendée.

**重要提示 本出版物封面上的"内部颜色"标志表明它包含被认为有助于正确理解其内容的颜色。因此，用户应使用彩色打印机打印本出版物。**

# INTRODUCTION

La CEI 61499 comprend les parties suivantes, sous le titre général *Blocs fonctionnels*:

- La Partie 1 (le présent document) contient:
  - des exigences générales, y compris le domaine d'application, les références normatives, des définitions et des modèles de référence;
  - des règles pour la déclaration des *types de blocs fonctionnels*, et des règles pour le comportement d'*instances* des types ainsi déclarés;
  - des règles pour l'utilisation des blocs fonctionnels dans la *configuration* de *systèmes* de mesure et commande dans les processus industriels (les IPMCS);
  - des règles pour l'utilisation des blocs fonctionnels pour satisfaire aux exigences de communication des systèmes IPMCS distribués;
  - des règles pour l'utilisation de blocs fonctionnels dans la gestion *d'applications, de ressources* et d'*équipements* dans les IPMCS distribués.
- La Partie 2 définit les exigences relatives aux *outils logiciels* pour prendre en charge les tâches d'études des systèmes suivantes:
  - la spécification des *types de blocs fonctionnels*;
  - la spécification fonctionnelle des *types de ressource* et des *types d'équipement*;
  - la spécification, l'analyse et la validation des IPMCS distribués;
  - la *configuration*, la *mise en œuvre*, l'exploitation et la maintenance de systèmes IPMCS distribués;
  - l'échange d'*informations* entre des *outils logiciels*.
- La Partie 3 (informations tutorielles) a été retirée en raison des nombreux supports tutoriels et éducationnels disponibles concernant la CEI 61499. Cependant, il n'est pas exclu qu'une deuxième édition mise à jour de la Partie 3 puisse être développée dans le futur.
- La Partie 4 définit des règles pour le développement des *profils de conformité* qui spécifient les caractéristiques de la CEI 61499-1 et de la CEI 61499-2 devant être mises en œuvre afin de promouvoir les qualités suivantes des systèmes, équipements et outils logiciels basés sur la CEI 61499:
  - interopérabilité des équipements issus de différents fournisseurs;
  - portabilité des logiciels entre les outils logiciels de plusieurs fournisseurs; et
  - aptitude à configurer des équipements provenant de plusieurs vendeurs par des outils logiciels de différents fournisseurs.





IEC61499由以下部分组成，总标题为功能块：

- 第1部分（本文档）包含：
  - 一般要求，包括范围、规范性参考文献、定义和参考模型；
  - 功能块类型的声明规则，以及如此声明的类型实例的行为规则；
  - 工业过程测量和控制系统配置中功能块的使用规则（IPMCS）；
  - 使用功能块的规则来满足分布式IPMCS系统的通信要求；

  分布式IPMCS中的应用程序、资源和设备管理中功能块的使用规则。
- 第2部分定义了支持以下系统工程任务的软件工具的要求：
  - 功能块类型规范；
  - 资源类型和设备类型的功能规范；
  - 分布式IPMCS的规范、分析和验证；
  - 分布式IPMCS系统的配置、实施、运行和维护；

  软件工具之间的信息交换。
- 由于提供了大量有关IEC61499的教程和教育材料，第3部分（教程信息）已被撤回。但是，不排除将来可能会开发更新的第3部分的第二版。
- 第4部分定义了一致性配置文件的开发规则，这些规则指定了要实施的IEC61499-1和IEC61499-2的特性，以提升基于IEC61499的系统、设备和软件工具的以下质量：
  - 来自不同供应商的设备的互操作性；
  - 来自多个供应商的软件工具之间的软件可移植性；和
  - 能够使用来自多个供应商的软件工具配置来自多个供应商的设备。

# BLOCS FONCTIONNELS –

# Partie 1: Architecture

## 1 Domaine d'application

La présente partie de la CEI 61499 définit une architecture générique et présente des lignes directrices pour l'utilisation de *blocs fonctionnels* dans des systèmes de mesure et de commande dans les processus industriels distribués (IPMCS). Cette architecture est présentée en termes de *modèles* de référence pouvant être mis en œuvre, de syntaxe textuelle et de représentations graphiques. Ces modèles, représentations et syntaxe **peuvent être utilisés pour**:

- la spécification et la normalisation des *types de blocs fonctionnels*;
- la spécification fonctionnelle et la normalisation d'éléments de système;
- la spécification indépendante de toute mise en œuvre, l'analyse et la validation des systèmes IPMCS distribués;
- la *configuration*, la *mise en œuvre*, l'exploitation et la maintenance de systèmes IPMCS distribués;
- l'échange d'*informations* parmi des *outils logiciels* pour l'accomplissement des *fonctions* ci-dessus.

La présente partie de la CEI 61499 ne limite ni ne spécifie les capacités fonctionnelles des IPMCS ou de leurs éléments de système, à l'exception de la manière dont ces capacités sont représentées en utilisant les éléments définis dans la présente norme. La CEI 61499-4 traite de la mesure dans laquelle les éléments définis dans la présente norme sont parfois limités par les capacités fonctionnelles des systèmes, sous-systèmes et équipements conformes.

Un des buts de la présente norme est de fournir des modèles de référence pour l'utilisation de blocs fonctionnels dans d'autres normes traitant de la prise en charge du cycle de vie du système, y compris la planification, la conception, la mise en œuvre, la validation, l'exploitation et la maintenance du système. Les modèles donnés dans la présente norme sont censés être génériques, indépendants vis-à-vis de tout domaine et extensibles à la définition et à l'utilisation de blocs fonctionnels dans d'autres normes ou pour des applications particulières ou des domaines d'application particuliers. L'intention est de faire en sorte que les spécifications écrites selon les règles données dans la présente norme soient concises, réalisables, complètes, non ambiguës et cohérentes.

NOTE 1 Les dispositions de la présente norme seule ne suffisent pas à assurer l'interopérabilité entre les équipements de différents vendeurs. Des normes conformes à la présente partie de la CEI 61499 sont susceptibles de spécifier des dispositions supplémentaires pour assurer ladite interopérabilité.

NOTE 2 Des normes conformes à la présente partie de la CEI 61499 sont susceptibles de spécifier des dispositions supplémentaires pour activer l'accomplissement des *fonctions* de gestion de *système*, d'*équipement*, de *ressource* et d'*application*.

## 2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 61131-1, *Automates programmables – Partie 1: Informations générales*



61499-1 © CEI:2012 – 125 –

# B –
# 第1部分：架构

IEC61499的这一部分定义了一个通用架构，并提供了在分布式工业过程测量和控制系统(IPMCS)中使用功能块的指南。该架构以可实现的参考模型、文本语法和图形表示形式呈现。这些模型、表示和语法可用于：

- 功能块类型的规范和标准化；
- 系统要素的功能规范和标准化；
- 实现分布式IPMCS系统的独立规范、分析和验证；
- 分布式IPMCS系统的配置、实施、运行和维护；
- 实现上述功能的软件工具之间的信息交换。

IEC61499的这一部分不限制或指定IPMCS或其系统元素的功能能力，除非这些能力如何使用本标准中定义的元素来表示。IEC61499-4解决了本标准中定义的元素有时受兼容系统、子系统和设备的功能能力限制的程度。

本标准的目的之一是为其他标准中处理系统生命周期支持的构建块的使用提供参考模型，包括系统的规划、设计、实施实施、验证、操作和维护。本标准中给出的模型是通用的、独立于领域的，并且可扩展至其他标准或特定应用程序或应用程序领域中功能块的定义和使用。其目的是使根据本标准给出的规则编写的规范简洁、可操作、完整、明确和一致。

注1：仅此标准的规定不足以确保来自不同供应商的设备之间的互操作性。符合IEC61499这一部分的标准可以指定额外的规定来确保这种互操作性。

注2符合IEC61499的这一部分的标准可以指定附加条款，以实现系统、设备、资源和应用程序管理功能的性能。

下列文件在本文件中全部或部分被规范引用，对其应用是必不可少的。对于注明日期的参考文献，仅引用的版本适用。对于未注明日期的引用文件，引用文件的最新版本适用（包括任何修改）。

IEC61131-1，可编程逻辑控制器-第1部分：一般信息

CEI 61131-3:2003, *Programmable controllers – Part 3: Programming languages* (disponible en anglais seulement)

ISO/CEI 7498-1:1994, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: le modèle de base*

ISO/CEI 8824-1:2008, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base*

ISO/CEI 10646:2003, *Technologies de l'information – Jeu universel de caractères codés (JUC)*

## 3 Termes and définitions

Pour les besoins du présent document, les termes et définitions suivants s'appliquent.

NOTE Les termes définis dans l'Article 3 sont en *italiques* lorsqu'ils apparaissent dans les définitions et Notes à l'article d'autres termes ainsi que dans l'entier du document.

**3.1**
**utilisateur**
*instance de bloc fonctionnel* qui fournit un *support adaptateur* d'un *type d'adaptateur d'interface* défini

**3.2**
**connexion d'adaptateur**
*connexion* d'une *fiche d'adaptation* à un *support adaptateur* du même *type d'adaptateur d'interface*, qui transporte les flots de *données* et d'*événements* définis par le *type d'adaptateur d'interface*

**3.3**
**type adaptateur d'interface**
*type* qui consiste en la définition d'un jeu d'*événements d'entrées,* d'*événements de sorties,* de *données d'entrées* et de *données* de *sorties* dont des *instances* sont des *fiches d'adaptation* et des *supports adaptateurs*

**3.4**
**algorithme**
jeu fini de règles bien définies pour la solution d'un problème en un nombre fini d'*opérations*

**3.5**
**application**
*unité fonctionnelle logicielle* qui est spécifique à la solution d'un problème pour la mesure et la commande dans les processus industriels

Note 1 à l'article: Une application peut être distribuée parmi des *ressources* et est apte à communiquer avec d'autres applications.

**3.6**
**attribut**
propriété ou caractéristique d'une *entité*, par exemple, l'identificateur de version d'une spécification du *type de bloc fonctionnel*

**3.7**
**type de bloc fonctionnel de base**
*type de bloc fonctionnel* qui ne peut pas être décomposé en d'autres blocs fonctionnels et qui utilise un *graphe de contrôle d'exécution (ECC)* pour contrôler l'*exécution* de ses *algorithmes*



IEC61131-3:2003，可编程控制器 第3部分：编程语言

ISOIEC7498-1:1994，信息技术 开放系统互连(OSI) 基本参考模型：基本模型

ISOIEC8824-1:2008，信息技术 抽象语法符号一(ASN.1)：基本符号规范

ISOIEC10646:2003，信息技术 通用编码字符集(UCS)

就本文件而言，以下术语和定义适用。

注：第3章中定义的术语在出现在定义和其他术语条目注释以及整个文件中时用斜体表示。

3.1提供已定义接口适配器类型的适配器支持的用户功能块实例

3.2适配器连接适配器插头与相同接口适配器类型的适配器介质的连接，承载接口适配器类型定义的数据和事件流

3.3接口适配器类型由一组输入事件、输出事件、输入数据和输出数据的定义组成的类型，实例有适配器插头和适配器支架

3.4algorithm有限集定义明确的规则，用于在有限次数的操作中解决问题

3.5应用软件功能单元，专门用于解决工业过程中的测量和控制问题

注1：应用程序可以分布在资源之间，并且能够与其他应用程序通信。

3.6attribute实体的属性或特性，例如功能块类型规范的版本标识符

3.7基本功能块类型不能分解成其他功能块并使用执行控制图（ECC）来控制其算法执行的功能块类型

**3.8**
**transaction bidirectionnelle**
*transaction* dans laquelle une demande et éventuellement des *données* sont acheminées d'un *demandeur* vers un *répondeur* et dans laquelle une réponse et éventuellement des données sont acheminées du répondeur vers le demandeur

3.8双向事务请求和可能的数据从请求者传送到响应者的事务，其中响应和可能的数据从响应者传送到请求者

**3.9**
**caractère**
membre d'un jeu d'éléments qui est utilisé pour la représentation, l'organisation ou le contrôle de *données*

3.9用于数据表示、组织或控制的项集的成员字符

**3.10**
**franchissement**
libération
<d'une transition EC> *opération* par laquelle le contrôle passe d'un *état EC* précédent d'une *transition EC* à son état EC suivant

3.10crossrelease<ofanECtransition>控制从一个EC转换的前一个EC状态转移到其下一个EC状态的操作

Note 1 à l'article: Cette opération consiste à désactiver l'état EC précédent, puis à activer l'état EC suivant.

注1：此操作包括停用前一个EC状态，然后激活下一个EC状态。

**3.11**
**connexion de communication**
*connexion* qui utilise la fonction de mapping de la communication d'une ou plusieurs *ressources* pour l'acheminement des *informations*

3.11通信连接使用一种或多种资源的通信映射功能来传递信息的连接。

**3.12**
**bloc fonctionnel de communication**
*bloc fonctionnel interface de service* qui représente l'*interface* entre une *application* et la fonction de mapping d'une *ressource*

3.12通信功能块服务接口功能块表示应用程序和资源映射功能之间的接口

**3.13**
**type de bloc fonctionnel de communication**
*type de bloc fonctionnel* dont les *instances* sont des *blocs fonctionnels de communication*

3.13通信功能块类型实例为通信功能块的功能块类型

**3.14**
**bloc fonctionnel composant**
*instance de bloc fonctionnel* qui est utilisée dans la spécification d'un *algorithme* d'un *type de bloc fonctionnel composé*

3.14componentfunctionblock复合功能块类型算法规范中使用的功能块实例

Note 1 à l'article: Un bloc fonctionnel composant peut être du *type basic* (de base), *composite* (composé) ou *service interface* (interface service).

注1：组件功能块可以是基本的、复合的或服务接口。

**3.15**
**sous-application constitutive**
*instance de sous-application* qui est utilisée dans la spécification d'un *type de sous-application*

3.15构成子应用程序一个子应用程序的实例，在一种子应用程序的规范中使用

**3.16**
**type de bloc fonctionnel composé**
*type de bloc fonctionnel* dont les *algorithmes* et le contrôle de leur *exécution* sont entièrement exprimés en termes de *blocs fonctionnels composants, événements,* et *variables* interconnectés

3.16复合功能块类型一种功能块，其算法和对其执行的控制完全用互连的组件、事件和变量功能块来表示。

**3.17**
**concomitant**
relatif à des *algorithmes* qui sont *exécutés* pendant une période de temps commune au cours de laquelle ils peuvent devoir partager en alternance des *ressources* communes

3.17并发与在一个公共时间段内执行的算法有关，在此期间它们可能不得不交替共享公共资源

**3.18**
**configuration (d'un *système* ou d'un *équipement*)**
action consistant à sélectionner des *unités fonctionnelles*, affecter leurs emplacements et définir leurs interconnexions

3.18（系统或设备的）配置选择功能单元、分配它们的位置和定义它们的互连的行为

**3.19**
**paramètre de configuration**
*paramètre* relatif à la *configuration* d'un *système,* d'un *équipement* ou d'une *ressource*

3.19configurationparameter与系统、设备或资源的配置有关的参数

**3.20**
**primitive confirm**
*primitive de service* qui représente une interaction dans laquelle une *ressource* indique l'achèvement d'un *algorithme invoqué* précédemment par une interaction via une *primitive "request"*

3.20确认原语表示交互的服务原语，其中资源指示先前由交互通过"请求"原语调用的算法的完成

**3.21**
**connexion**
association établie entre des *unités fonctionnelles* pour acheminer de l'*information*

3.21连接关联建立功能单元之间传递信息

**3.22**
**région critique**
*opération* ou séquence d'opérations qui est *exécutée* sous le contrôle exclusif d'un objet verrou qui est associé aux *données* sur lesquelles les opérations sont accomplies

3.22临界区操作或操作序列，在与执行操作的数据相关联的锁对象的排他控制下执行

**3.23**
**donnée**
représentation réinterprétable d'*information* d'une manière formalisée adaptée à la communication, à l'interprétation ou au traitement

3.23datareinterpretable以适合交流、解释或处理的形式化方式表示的信息

**3.24**
**connexion de données**
association entre deux *blocs fonctionnels* pour l'acheminement de *données*

3.24数据连接两个功能块之间用于传送数据的关联

**3.25**
**entrée de données**
*interface* d'un *bloc fonctionnel* qui reçoit des *données* d'une *connexion de données*

3.25数据输入接口从数据连接接收数据的功能块

**3.26**
**sortie de données**
*interface* d'un *bloc fonctionnel* qui fournit des *données* à une *connexion de données*

3.26数据输出接口向数据连接提供数据的功能块

**3.27**
**type de données**
jeu de valeurs accompagné d'un jeu d'*opérations* autorisées

3.27数据类型setofvalues伴随着一组允许的操作

**3.28**
**déclaration**
mécanisme pour établir la définition d'une *entité*

3.28建立实体定义的声明机制

Note 1 à l'article: Une déclaration est susceptible d'impliquer le rattachement d'un *identificateur* à l'entité et de lui affecter des *attributs* tels que *types de données* et *algorithmes*.

注1：语句可能涉及将标识符附加到实体并为其分配属性，例如数据类型和算法。

**3.29**
**équipement**
*entité* physique indépendante capable d'accomplir une ou plusieurs *fonctions* spécifiées dans un contexte particulier et délimitée par ses *interfaces*

3.29设备独立物理实体，能够在特定上下文中执行一项或多项指定功能并受其接口限制

Note 1 à l'article: Un *système d'équipement de commande programmable* comme défini dans la CEI 61131-1 est un *équipement*.

注1：IEC61131-1中定义的可编程控制设备系统是设备。

**3.30**
**application de gestion d'équipement**
*application* dont la fonction principale est la gestion de plusieurs *ressources* au sein d'un *équipement*

3.30设备管理应用程序主要功能是管理一个设备内的若干资源的应用程序

**3.31**
**entité**
objet particulier, telle qu'une personne, un lieu, un *processus*, un objet, un concept, une association ou un *événement*

3.31特定对象实体，例如人、地点、过程、对象、概念、关联或事件

**3.32**
**événement**
occurrence instantanée qui est significative pour la programmation de l'*exécution* d'un *algorithme*

3.32对算法的编程执行具有重要意义的瞬时发生事件

Note 1 à l'article: L'exécution d'un algorithme peut utiliser des *variables* associées à un événement.

注1：算法的执行可以使用与事件相关的变量。

**3.33**
**connexion d'événements**
association parmi des *blocs fonctionnels* pour l'acheminement d'*événements*

3.33事件路由功能块之间的事件连接关联

**3.34**
**entrée d'événements**
*interface* d'un *bloc fonctionnel* qui peut recevoir des *événements* d'une *connexion d'événements*

3.34事件输入接口可以从事件连接接收事件的功能块

**3.35**
**sortie d'événements**
*interface* d'un *bloc fonctionnel* qui peut émettre des *événements* vers une *connexion d'événements*

3.35事件输出接口可以向事件连接发送事件的功能块

**3.36**
**exception**
*événement* qui entraîne la suspension d'une *exécution* normale

3.36导致正常执行暂停的异常事件

**3.37**
**exécution**
processus consistant à accomplir une séquence d'*opérations* spécifiée par un *algorithme*

3.37executionprocess执行算法指定的一系列操作的过程

Note 1 à l'article: La séquence d'opérations à exécuter peut varier d'une *invocation* d'une *instance de bloc fonctionnel* à l'autre, selon les règles spécifiées par l'*algorithme* du bloc fonctionnel et des valeurs courantes de *variables* dans la structure de données du bloc fonctionnel.

注1：根据功能块算法规定的规则和功能块数据结构中变量的当前值，从一个功能块实例调用到另一个，要执行的操作顺序可能不同。

**3.38**
**action de contrôle d'exécution**
**action EC**
élément associé à un *état de contrôle d'exécution*, qui identifie un *algorithme* devant être *exécuté* et/ou un *événement* devant être émis

3.38执行控制动作与执行控制状态相关联的EC动作元素，它标识要执行的算法和/或要发出的事件

Note 1 à l'article: La temporisation de l'exécution d'algorithme et l'émission d'un événement sont traitées en 5.2.2.

注1：算法执行时间和事件发射在5.2.2中介绍。

**3.39**
**graphe de contrôle d'exécution**
**graphe ECC**
représentation graphique ou textuelle des relations causales entre des *événements* aux *entrées d'événements* et aux *sorties d'événements* d'un *bloc fonctionnel* et l'*exécution* des *algorithmes* du bloc fonctionnel, en utilisant des *états de contrôle d'exécution,* des *transitions de contrôle d'exécution* et des *actions de contrôle d'exécution*

3.39执行控制图ECC图使用执行控制、执行控制转换和执行控制动作的状态，功能块的事件输入和事件输出与功能块算法的执行之间的因果关系的图形或文本表示。

**3.40**
**état initial de contrôle d'exécution**
**état initial EC**
*état de contrôle d'exécution* qui est actif à la suite de l'initialisation d'un *graphe de contrôle d'exécution*

3.40初始执行控制状态initialstateEC执行控制状态，在执行控制图初始化后处于活动状态

**3.41**
**état de contrôle d'exécution**
**état EC**
situation dans laquelle le comportement d'un *bloc fonctionnel de base* par rapport à ses *variables* est déterminé par les *algorithmes* associés à un jeu spécial d'*actions de contrôle d'exécution*

3.41执行控制状态ECstate基本功能块相对于其变量的行为由与一组特殊的执行控制动作相关的算法确定的状态。

**3.42**
**transition de contrôle d'exécution**
**transition EC**
moyen par lequel le contrôle passe d'un *état de contrôle d'exécution* précédent à un *état de contrôle d'exécution* suivant

3.42执行控制转换EC转换是指控制从先前的执行控制状态改变到随后的执行控制状态。

**3.43**
**défaut**
condition anormale pouvant être à l'origine d'une diminution ou d'une perte de capacité d'une *unité fonctionnelle* pour accomplir une *fonction* requise

3.43缺陷可能导致功能单元执行所需功能的能力下降或丧失的异常状态

**3.44**
**fonction**
but spécifique d'une *entité* ou son action caractéristique

3.44function实体的特定目的或其特征动作

**3.45**
**bloc fonctionnel**
**instance de bloc fonctionnel**
*unité fonctionnelle logicielle* consistant en une copie individuelle nommée d'une structure de données sur laquelle des *opérations* associées peuvent être accomplies comme spécifié par un *type de bloc fonctionnel* correspondant

3.45功能块functionblockinstance软件功能单元，由数据结构的一个命名的单独副本组成，可以按照相应的功能块类型在其上执行相关操作

Note 1 à l'article: Les opérations typiques d'un bloc fonctionnel comprennent la modification des valeurs des données dans sa structure de données associée.

注1：功能块的典型操作包括改变其关联数据结构中的数据值。

Note 2 à l'article: L'*instance de bloc fonctionnel* et son *type de bloc fonctionnel* correspondant défini dans la CEI 61131-3 sont des éléments de langage de programmation avec un jeu différent de caractéristiques.

注2：功能块实例及其对应的功能块类型定义在IEC61131-3是具有一组不同功能的编程语言元素。

**3.46**
**réseau de blocs fonctionnels**
*réseau* dont les nœuds sont des *blocs fonctionnels* ou des *sous-applications* et leurs *paramètres* et dont les branches sont des *connexions de données* et des *connexions d'événements*

3.46networkoffunctionblocksnetwork，其节点为功能块或子应用及其参数，其分支为数据连接和事件连接

Note 1 à l'article: Il s'agit d'une généralisation du *diagramme de blocs fonctionnels* défini dans la CEI 61131-3.

注1：这是IEC61131-3中定义的功能框图的概括。

**3.47**
**type de bloc fonctionnel**
*type* dont les *instances* sont des *blocs fonctionnels*

3.47typeoffunctionblocktype其实例为功能块

Note 1 à l'article: Les types de bloc fonctionnel comprennent des types de bloc fonctionnel de base, des types de bloc fonctionnel composé et des types de bloc fonctionnel interface de service

注1：功能块类型包括基本功能块类型、复合功能块类型和服务接口功能块类型

**3.48**
**unité fonctionnelle**
*entité* d'équipement *matériel* et/ou *logiciel* capable d'accomplir une action spécifiée

3.48功能单元能够执行特定动作的硬件和/或软件设备的实体

**3.49**
**matériel**
équipement physique, par opposition aux programmes, procédures, règles et documentation associée

3.49硬件物理设备，相对于程序、程序、政策和相关文件

**3.50**
**identificateur**
un ou plusieurs *caractères* utilisés pour nommer une *entité*

3.50identifier用来命名一个实体的一个或多个字符

**3.51**
**mise en œuvre**
phase de développement dans laquelle le *matériel* et le *logiciel* d'un *système* deviennent opérationnels

3.51实施阶段系统的硬件和软件开始运行的开发阶段

**3.52**
**primitive indication**
*primitive de service* qui représente une interaction dans laquelle une *ressource* soit
a) indique qu'elle a, de sa propre initiative, *invoqué* un certain *algorithme*;
b) indique qu'un *algorithme* a été invoqué par une *application* homologue

3.52指示原语表示交互的服务原语，其中资源a)表明它主动调用了某种算法；b)表示一个算法已被对等应用程序调用

**3.53**
**information**
signification qui est attribuée actuellement à une *donnée* au moyen de conventions appliquées à la donnée en question

3.53信息指当前通过适用于相关数据的约定分配给数据的含义

**3.54**
**variable d'entrée**
*variable* dont la valeur est fournie par une *entrée de données* et qui peut être utilisée dans une ou plusieurs *opérations* d'un *bloc fonctionnel*

3.54输入变量其值由数据输入提供并且可用于功能块的一项或多项操作的变量

Note 1 à l'article: Un *paramètre d'entrée* d'un *bloc fonctionnel*, tel que défini dans la CEI 61131-3, est une *variable d'entrée.*

注1：功能块的输入参数，如IEC61131-3中定义的，是输入变量。

**3.55**
**instance**
*unité fonctionnelle* consistant en une *entité* nommée individuelle avec les *attributs* d'un *type* défini

3.55功能单元实例，由具有定义类型属性的单个命名实体组成

**3.56**
**nom d'instance**
*identificateur* associé à une *instance* et la désignant

3.56实例名称与实例相关联并指定实例的标识符

**3.57**
**instanciation**
création d'une *instance* d'un *type* spécifié

3.57实例化创建指定类型的实例

**3.58**
**interface**
frontière partagée entre deux *unités fonctionnelles*, définies par les caractéristiques fonctionnelles, caractéristiques de signal ou d'autres caractéristiques appropriées selon le cas

3.58两个功能单元之间的共享边界接口，由功能特性、信号特性或其他适当的特性来定义。

**3.59**
**opération interne**
<d'un *bloc fonctionnel*> *opération* associée à un *algorithme* d'un bloc fonctionnel, avec son contrôle d'*exécution* ou avec les capacités fonctionnelles de la *ressource* associée

3.59internaloperation<ofafunctionblock>与功能块的算法、执行控制或相关资源的功能能力相关的操作

**3.60**
**variable interne**
*variable* dont la valeur est utilisée ou modifiée par une ou plusieurs *opérations* d'un *bloc fonctionnel*, mais n'est pas fournie par une *entrée de données* ou à une *sortie de données*

**3.61**
**invocation**
processus de lancement de l'*exécution* de la séquence d'*opérations* spécifiées dans un *algorithme*

**3.62**
**liaison**
élément de conception décrivant la *connexion* entre un *équipement* et un *segment* du *réseau*

**3.63**
**libellé**
unité lexicale qui représente directement une valeur

**3.64**
**bloc fonctionnel de gestion**
*bloc fonctionnel* dont la *fonction* principale est la gestion d'*applications* dans les limites d'une *ressource*

**3.65**
**ressource de gestion**
*ressource* dont la *fonction* principale est la gestion d'autres *ressources*

**3.66**
**mapping**
ensemble de caractéristiques ou d'*attributs* ayant une correspondance définie avec les membres d'un autre ensemble

**3.67**
**message**
série ordonnée de *caractères* destinés à acheminer de l'*information*

**3.68**
**puits de messages**
partie intégrante d'un *système* de communication dans laquelle des *messages* sont considérés comme reçus

**3.69**
**source de messages**
partie intégrante d'un *système* de communication de laquelle des *messages* sont considérés provenir

**3.70**
**modèle**
représentation mathématique ou physique d'un système ou d'un processus

**3.71**
**multitâche**
mode de fonctionnement qui prévoit l'*exécution concomitante* de deux *algorithmes* ou plus

**3.72**
**réseau**
agencement de nœuds et de branches d'interconnexion



3.60internalvariable其值被功能块的一个或多个操作使用或改变，但不由数据输入或数据输出提供的变量

3.61invocationprocess开始执行算法中指定的操作序列的过程

3.62链路设计元素，描述设备与网段之间的连接

3.63label直接表示一个值的词法单元

3.64管理功能块功能块，其主要功能是管理资源边界内的应用程序

3.65资源管理资源，其主要功能是管理其他资源

3.66映射一组与另一组成员有明确对应关系的特征或属性

3.67message用于传达信息的有序字符系列

3.68messagesink消息接收器一个通信系统的组成部分，在该系统中认为消息被接收。

3.69messagesource消息源一个通信系统的组成部分，消息被认为是源自该部分。

3.70模型系统或过程的数学或物理表示

3.71multitaskingmodeofoperation，提供两个或多个算法的并发执行

3.72互连节点和分支的网络布置

**3.73**
**opération**
action bien définie qui, lorsqu'elle est appliquée à n'importe quelle combinaison admissible d'*entités* connues, produit une nouvelle *entité*

3.73operation定义明确的操作，当应用于已知实体的任何允许组合时，产生一个新实体

**3.74**
**variable de sortie**
*variable* dont la valeur est établie par une ou plusieurs *opérations* d'un *bloc fonctionnel* et est fournie à une *donnée de sortie*

3.74输出变量其值由功能块的一个或多个操作确定并提供给输出数据的变量

Note 1 à l'article: Un *paramètre de sortie* d'un *bloc fonctionnel*, comme défini dans la CEI 61131-3, est une *variable de sortie.*

注1：如IEC61131-3中定义的功能块输出参数是一个输出变量。

**3.75**
**paramètre**
*variable* à laquelle est donnée une valeur constante pour une *application* spécifiée et qui peut représenter l'application

3.75variableparameter给定应用的常数值，可以代表应用

**3.76**
**prise mâle**
**fiche d'adaptation**
*instance* d'un *type d'adaptateur d'interface* qui fournit un point de départ d'une *connexion d'adaptation* à partir d'un bloc fonctionnel *fournisseur*

3.76plugadapterplug接口适配器类型的实例，为来自供应商功能块的适配器连接提供起点

**3.77**
**fournisseur**
*instance de bloc fonctionnel* qui fournit une *fiche d'adaptation* d'un *type d'adaptateur d'interface* défini

3.77provider功能块实例，提供定义接口适配器类型的适配器插头

**3.78**
**primitive request**
*primitive de service* qui représente une interaction dans laquelle une *application* invoque un certain *algorithme* fourni par un *service*

3.78请求原语表示应用程序调用服务提供的某种算法的交互的服务原语

**3.79**
**demandeur**
*unité fonctionnelle* qui lance une *transaction* par le biais d'une *primitive request*

3.79requester通过请求原语发起事务的功能单元

**3.80**
**ressource**
*unité fonctionnelle* qui a un contrôle indépendant de son fonctionnement et qui fournit divers *services* à des *applications,* y compris la programmation et l'*exécution* d'*algorithmes*

3.80功能单元独立控制其操作并向应用程序提供各种服务的资源，包括编程和执行算法

Note 1 à l'article: Le terme RESOURCE défini dans la CEI 61131-3:2003, 1.3.66 est un élément de langage de programmation correspondant à la *ressource* définie ci-dessus.

注1：IEC61131-3:2003 1.3.66中定义的术语RESOURCE是对应于上述资源的编程语言元素。

Note 2 à l'article: Un *équipement* contient une ou plusieurs ressources.

注2：一件设备包含一个或多个资源。

**3.81**
**application de gestion de ressource**
*application* dont la fonction principale est la gestion d'une seule *ressource*

3.81资源管理应用程序主要功能是管理单个资源的应用程序

**3.82**
**répondeur**
*unité fonctionnelle* qui conclut une *transaction* par le biais d'une *primitive response*

3.82响应者功能单元，通过响应原语结束事务

**3.83**
**primitive response**
*primitive de service* qui représente une interaction dans laquelle une *application* indique qu'elle a exécuté un *algorithme* précédemment *invoqué* par une interaction représentée par une *primitive indication*

**3.84**
**échantillonner**, verb
capter et retenir la valeur instantanée d'une *variable* pour une utilisation ultérieure

**3.85**
**fonction de programmation**
*fonction* qui sélectionne des *algorithmes* ou des *opérations* pour l'*exécution* et lance et arrête cette exécution

**3.86**
**segment**
partition physique d'un *réseau de communication*

**3.87**
**service**
capacité fonctionnelle d'une *ressource* qui peut être modélisée par une séquence de *primitives de service*

**3.88**
**bloc fonctionnel interface de service**
*bloc fonctionnel* qui fournit un ou plusieurs *services* à une *application*, en se basant sur un *mapping* des *primitives de service* avec les *événements d'entrée,* les *événements de sortie,* les *données d'entrée* et les *données de sortie* du bloc fonctionnel

**3.89**
**primitive de service**
représentation abstraite et indépendante vis-à-vis de toute mise en œuvre, d'une interaction entre une *application* et une *ressource*

**3.90**
**diagramme de séquence d'un service**
diagramme représentant une séquence de *primitives de service*

**3.91**
**prise femelle**
**support adaptateur**
*instance* d'un *type adaptateur d'interface* qui fournit un point d'extrémité pour une *connexion d'adaptation* à un bloc fonctionnel d'*utilisateur*

**3.92**
**logiciel**
création intellectuelle comprenant les programmes, procédures, règles, *configurations* et toute documentation associée relatifs à l'exploitation d'un *système*

**3.93**
**outil logiciel**
*logiciel* qui est utilisé pour la production, l'inspection ou l'analyse d'un autre logiciel

**3.94**
**instance de sous-application**
*instance* d'un *type sous-application* à l'intérieur d'une *application* ou à l'intérieur d'un type sous-application



3.83响应原语表示交互的服务原语，在该交互中，应用程序表明它已经执行了先前由指示原语表示的交互调用的算法

3.84tosample，动词用来捕获和保留变量的瞬时值以备后用

3.85编程功能选择算法或操作执行并开始和停止执行的功能

3.86段通信网络的物理分区

3.87服务功能能力可以由一系列服务原语建模的资源

3.88服务接口functionalblock向应用程序提供一项或多项服务的功能块，基于服务原语到功能块的输入事件、输出事件、输入数据和输出数据的映射。

3.89服务原语应用程序和资源之间交互的抽象、独立于实现的表示

3.90服务序列图表示服务原语序列的图

3.91adaptersocketsocket接口适配器类型的实例，为适配器连接到用户功能块提供端点

3.92智力创造软件，包括与系统运行有关的程序、程序、规则、配置和任何相关文件

3.93软件用于生产、检查或分析其他软件的软件工具

3.94sub-applicationinstance一个应用程序内部或一个子应用程序类型内部的一个子应用程序类型的实例

Note 1 à l'article: Une instance de sous-application peut être distribuée parmi des *ressources*, c'est-à-dire que ses blocs fonctionnels composants ou le contenu de ses sous-applications constitutives peu(ven)t être affecté(s) à des ressources différentes.

**3.95**
**type de sous-application**
*unité fonctionnelle* dont le corps est constitué de *blocs fonctionnels composants* ou de *sous-applications constitutives*

Note 1 à l'article: Un type sous-application permet la création de sous-structures d'*applications* sous la forme d'une hiérarchie autosemblable.

**3.96**
**système**
ensemble d'éléments reliés entre eux, considérés dans un contexte défini comme un tout et séparés de leur environnement

Note 1 à l'article: De tels éléments peuvent être tant des objets matériels que des concepts et les résultats de ceux-ci (par exemple: formes d'organisation, méthodes mathématiques, langages de programmation)

Note 2 à l'article: Le système est considéré comme séparé de l'environnement et des autres systèmes extérieurs par une surface imaginaire à travers laquelle peuvent passer les liaisons entre le système considéré et l'environnement.

**3.97**
**variable temporaire**
*variable* dont la valeur est initialisée, utilisée et éventuellement modifiée durant l'*exécution* d'un *algorithme*; qui n'est pas visible à l'extérieur du corps de l'algorithme, et dont la valeur ne persiste pas d'une exécution de l'algorithme à l'autre

**3.98**
**transaction**
unité de service dans laquelle une demande et éventuellement des *données* sont acheminées d'un *demandeur* vers un *répondeur* et dans laquelle une réponse et éventuellement des données peuvent également être acheminées en retour du répondeur vers le demandeur

**3.99**
**type**
élément *logiciel* qui spécifie les *attributs* communs partagés par toutes les *instances* du type

**3.100**
**nom de type**
*identificateur* associé à un *type* et le désignant

**3.101**
**transaction unidirectionnelle**
*transaction* dans laquelle une demande et éventuellement des *données* sont acheminées d'un *demandeur* vers un *répondeur* et dans laquelle une réponse n'est pas acheminée en retour du répondeur vers le demandeur

**3.102**
**variable**
*entité logicielle* qui peut prendre différentes valeurs, une à la fois

Note 1 à l'article: Les valeurs d'une variable sont habituellement limitées à un certain *type de données*.

Note 2 à l'article: Les variables peuvent être classées en *variables d'entrée, variables de sortie*, *variables internes* et *variables temporaires*.



注1：子应用程序实例可以分布在资源之间，即它的组件功能块或其组成子应用程序的内容可能会受到影响）到不同的资源。

3.95sub-applicationtype功能单元，其主体由组件功能块或组成子应用程序组成

注1：子应用程序类型允许将应用程序的子结构创建为自相似的层次结构。

3.96系统一组相互关联的元素，在定义的上下文中作为一个整体考虑并与其环境分离

注1：这些元素既可以是物质对象，也可以是概念及其结果（例如组织形式、数学方法、编程语言）

注2：系统被认为是通过一个想象的表面与环境和其他外部系统分开，所考虑的系统和环境之间的链接可以通过该表面。

3.97临时变量其值在算法执行期间被初始化、使用和可能修改的变量；它在算法主体之外不可见，并且其值不会从算法的一次执行到另一次执行

3.98服务单元事务，其中请求和可能的数据从请求者传送到响应者，其中响应和可能的数据也可能从响应者传送回请求者

3.99软件元素类型，指定由该类型的所有实例共享的公共属性

3.100类型名称与类型关联并指定类型的标识符

3.101单向事务一个请求和可能的数据从请求者传送到响应者并且响应不从响应者传送回请求者的事务

3.102软件实体变量，可以取不同的值，一次取一个

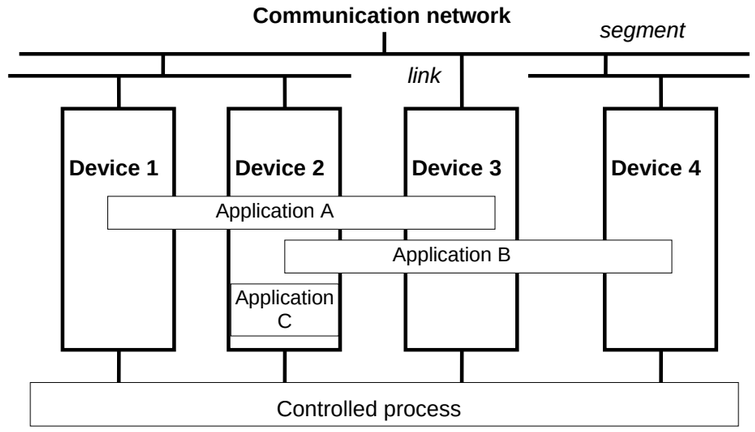
注1：一个变量的值通常被限制为某种数据类型。

注2：变量可分为输入变量、输出变量、内部变量和临时变量。

# 4 Modèles de référence

## 4.1 Modèle pour un système

Pour les besoins de la CEI 61499, un *système* de mesure et de commande dans les processus industriels (IPMCS) est modélisé (voir Figure 1) comme étant un ensemble d'*équipements* connectés entre eux et communiquant les uns avec les autres au moyen d'un *réseau* de communication constitué de *segments* et de *liaisons*. Les équipements sont connectés à des segments de réseau par des *liaisons*.




NOTE Le processus commandé n'est pas partie intégrante du système de mesure et de commande.

**Légende**

| Anglais | Français |
|---|---|
| Communication network | Réseau de communication |
| segment | segment |
| link | liaison |
| Device | Équipement |
| Application A | Application A |
| Application B | Application B |
| Appl. C | Application C |
| Controlled process | Processus commandé |

**Figure 1 – Modèle de système**

Une *fonction* accomplie par l'IPMCS est modélisée comme une *application* qui peut résider dans un seul équipement, telle que l'application C à la Figure 1, ou peut être distribuée parmi plusieurs équipements, telle que les applications A et B de la Figure 1. Par exemple, une application peut être constituée d'une ou plusieurs boucles de commande dans lesquelles l'échantillonnage des entrées est accompli dans un équipement, le traitement de commande est accompli dans un autre équipement et la conversion des sorties dans un troisième.

## 4.2 Modèle pour un équipement

Comme illustré à la Figure 2, un *équipement* doit contenir au moins une *interface,* à savoir, une interface de processus ou une interface de communication et peut contenir zéro *ressource* ou plus.

NOTE 1 Un équipement est considéré être une *instance* d'un *type* d'équipements correspondant comme spécifié en 7.2.2.

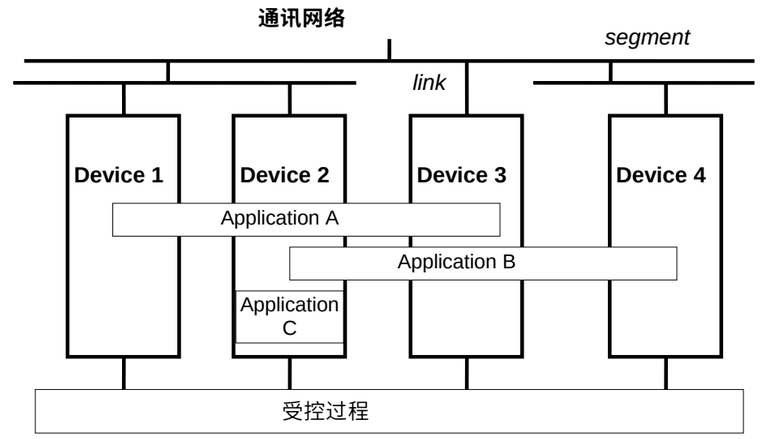




# 4 M

## 系统模型

就IEC61499而言，工业过程测量和控制系统(IPMCS)被建模（参见图1）为一组连接在一起并通过由段和链路组成的通信网络相互通信的设备。设备通过链路连接到网段。




注：受控过程不是测量和控制系统的组成部分。

**Légende**

| lais | is |
|---|---|
| 通讯网络 | 通讯网络 |
| segment | segment |
| link | liaison |
| Device | Équipement |
| Application A | Application A |
| Application B | Application B |
| | Application C |
| 受控过程 | dé |

**图1 系统模型**

IPMCS执行的功能被建模为可以驻留在单个设备中的应用程序，例如图1中的应用程序C，或者可以分布在多个设备中，例如图1中的应用程序A和B例如，一个应用程序可能包含在一个或多个控制回路中，输入采样在一个设备中完成，控制处理在另一个设备中完成，输出转换在第三个设备中完成。

## 设备型号

如图2所示，一个设备必须至少包含一个接口，即进程接口或通信接口，并且可以包含零个或多个资源。

注1设备被认为是7.2.2中规定的相应设备类型的实例。

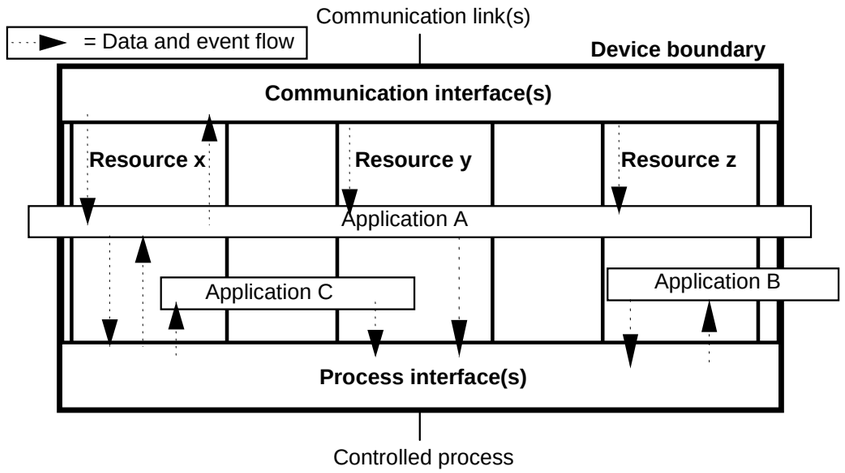
NOTE 2 Un équipement qui ne contient aucune ressource est considéré être fonctionnellement l'équivalent d'une *ressource* telle que définie en 4.3.

Une "interface de processus" assure un *mapping* entre le processus physique (mesures analogiques, E/S discrètes, etc.) et les ressources. Les informations échangées avec le processus physique sont présentées à la ressource sous forme de *données* et/ou d'*événements*.

Les *interfaces* de communication assurent un mapping entre des ressources et les informations échangées par l'intermédiaire d'un *réseau*. Les services fournis par les interfaces de communication peuvent comprendre:

- la présentation d'informations de communication à la ressource sous forme de *données* et/ou d'*événements*;
- des services complémentaires pour prendre en charge la programmation, la *configuration*, le diagnostic, etc.

Les *liaisons* de communication peuvent être associées soit directement à un *équipement*, soit à une instance de type *ressource* spécifique (ressource de communication) à laquelle une partie de l'application distribuée peut ou peut ne pas être mappée, en fonction du type de ressource.




NOTE Cette figure illustre une structure interne possible de l'Équipement 2 de la Figure 1.

**Légende**

| Anglais | Français |
|---|---|
| Communication link(s) | Liaison(s) de communication |
| Resource x | Ressource x |
| Controlled process | Processus commandé |
| Resource z Resource y | Ressource z Ressource y |
| Application B Application C | Application B Application C |
| Application A | Application A |
| Device boundary | Limite de l'équipement |
| Communication interface(s) | Interface(s) de communication |
| Process interface(s) | Interface(s) du processus |
| = Data & event flow | Flot de données et d'événements |

**Figure 2 – Modèle d'un équipement**



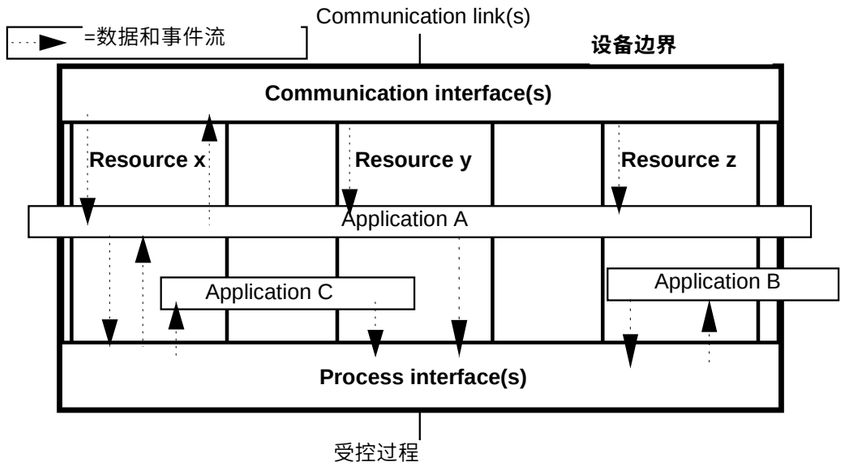
注2：不包含资源的设备被认为在功能上等同于4.3中定义的资源。

“过程接口”确保物理过程（模拟测量、离散IS等）与资源之间的映射。与物理过程交换的信息以数据和/或事件的形式呈现给资源。

通信接口提供资源和通过网络交换的信息之间的映射。通信接口提供的服务可能包括：

- 以数据和/或事件的形式向资源提供通信信息；
- 支持编程、配置、诊断等的附加服务。

通信链路可以直接与设备相关联，也可以与特定资源类型实例（通信资源）相关联，分布式应用程序的一部分可能映射到也可能不映射到，这取决于资源的类型。




注：该图说明了图1中设备2的可能内部结构。

**Légende**

| Anglais | |
|---|---|
| Communication link(s) | 通讯链接 |
| | Ressource x |
| 受控过程 | Processus commandé |
| Resource z Resource y | Ressource z Ressource y |
| Application B Application C | Application B Application C |
| | Application A |
| 设备边界 | |
| Communication interface(s) | 通讯接口 |
| | |
| =数据和事件流 | 数据流和事件流 |

**Figure 2 – Modèle d'un équipement**

### 4.3 Modèle pour une ressource

Pour les besoins de la CEI 61499, une *ressource* est considérée être une *unité fonctionnelle* dotée d'une commande indépendante pour son fonctionnement et qui est contenue dans un *équipement*. Elle peut être créée, configurée, paramétrée, démarrée, supprimée, etc., sans avoir d'influence sur d'autres ressources.

NOTE 1 Une ressource est considérée être une *instance* du *type* de ressources correspondant comme spécifié en 7.2.1.

NOTE 2 Bien qu'une ressource ait une commande indépendante pour son fonctionnement, ses états opérationnels pourraient nécessiter d'être coordonnés avec ceux d'autres ressources à des fins d'installation, d'essai, etc.

Les *fonctions* d'une ressource sont d'accepter des *données* et/ou des *événements* provenant des *interfaces* de processus et/ou de communication, de traiter les données et/ou événements et de retourner les données et/ou événements aux interfaces du processus et/ou de communication, comme spécifié par les *applications* qui utilisent la ressource.

NOTE 3 Outre la prise en charge des fonctions énumérées ci-dessus, des types spécifiques de ressources pourraient représenter une capacité de mise en œuvre des fonctions d'interface telles que les interfaces de processus ou des services de communications de couche inférieure sur des liaisons de communication. En fonction du type de ressource en question, ces services sont susceptibles ou non d'être les seuls qu'elles soient capables de fournir.

NOTE 4 La prise en considération d'autres aspects possibles de ressources ne relève pas du domaine d'application de la présente norme.

Comme illustré à la Figure 3, une ressource est modélisée par ce qui suit:

- Une ou plusieurs "applications locales" (ou parties locales d'applications distribuées). Les *variables* et *événements* manipulés dans la présente partie sont des *variables d'entrée* et de *sortie* et des événements aux *entrées d'événements* et *sorties d'événements* des *blocs fonctionnels* qui accomplissent les *opérations* nécessaires à l'application.
- Une partie "mapping du processus" qui consiste à définir un *mapping* des *données* et des *événements* entre les *applications* et une ou plusieurs *interfaces* du processus. Comme montré à la Figure 3, ce mapping peut être modélisé par des *blocs fonctionnels interface de service* spécialisés dans ce but.
- Une partie "mapping de la communication" qui consiste à définir un *mapping* des *données* et des *événements* entre des *applications* et des *interfaces de communication*. Comme montré à la Figure 3, ce mapping peut être modélisée par des *blocs fonctionnels interface de service* spécialisés dans ce but.
- Une *fonction* de programmation qui effectue l'exécution des blocs fonctionnels dans les applications et le transfert de données entre ceux-ci, conformément aux exigences relatives à la temporisation et à la séquence déterminées par:

  a) l'occurrence des événements;

  b) les interconnexions des blocs fonctionnels; et

  c) les informations de programmation telles que les périodes et les priorités.





**资源模板**

就IEC61499而言，资源被认为是对其操作具有独立控制权并包含在设备中的功能单元。可以创建、配置、配置、启动、删除等，不影响其他资源。

注1资源被认为是7.2.1中规定的相应资源类型的一个实例。

注2尽管资源对其操作具有独立控制，但其操作状态可能需要与其他资源的安装、测试等相协调。

根据使用资源。

注3除了支持上面列出的功能外，特定类型的资源可以表示实现接口功能的能力，例如过程接口或数据链路上的低层通信服务。根据所讨论资源的类型，这些服务可能是也可能不是他们能够提供的唯一服务。

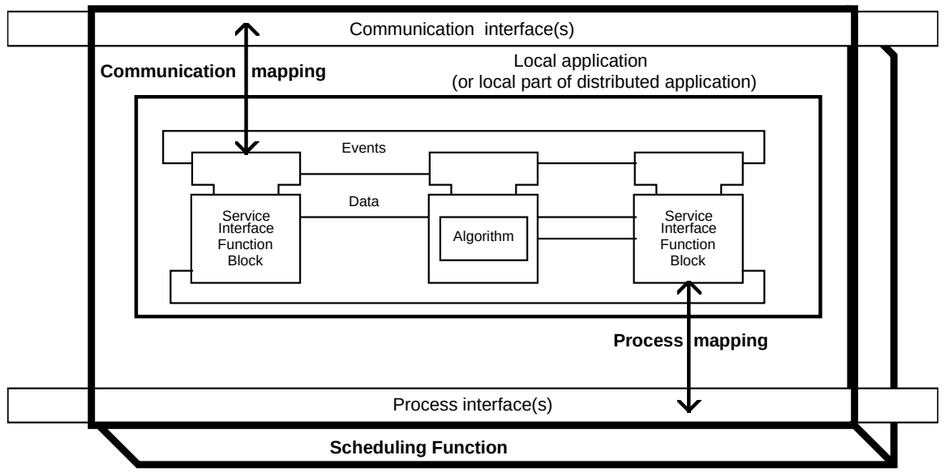
注4对其他可能的资源方面的考虑超出了本标准的范围。

如图3所示，资源通过以下方式建模：

- 一个或多个“本地应用程序”（或分布式应用程序的本地部分）。在这部分中操作的变量和事件是执行应用程序所需操作的功能块的事件输入和事件输出处的 输入和输出变量和事件。
- “流程映射”部分，包括定义应用程序与一个或多个流程接口之间的数据和事件的映射。如图3所示，这种映射可以由专门为此目的的服务接口功能块建模。
- “通信映射”部分，包括定义应用程序和通信接口之间的数据和事件的映射。如图3所示，这种映射可以由专门为此目的的服务接口功能块建模。
- 一种编程功能，根据由以下因素确定的时序和顺序要求，执行应用程序中的功能块并在它们之间传输数据：

  b)功能块互连；和

  c)调度信息，例如时间和优先级。




NOTE 1 Cette figure est uniquement illustrative. Ni la représentation graphique ni l'emplacement des blocs fonctionnels ne sont normatifs.

NOTE 2 Des interfaces de communication et de processus peuvent 'être partagées entre les ressources.

**Légende**

| Anglais | Français |
|---|---|
| Communication interface(s) | Interface(s) de communication |
| Communication mapping | Mapping de la communication |
| Local application (or local part of distributed application) | Application locale (ou partie locale d'une application distribuée) |
| Events | Événements |
| Data | Données |
| Service Interface Function Block | Bloc fonctionnel interface de service |
| Algorithm | Algorithme |
| Process mapping | Mapping du processus |
| Process interface(s) | Interface(s) du processus |
| Scheduling Function | Fonction de programmation |

**Figure 3 – Modèle d'une ressource**

### 4.4 Modèle pour une application

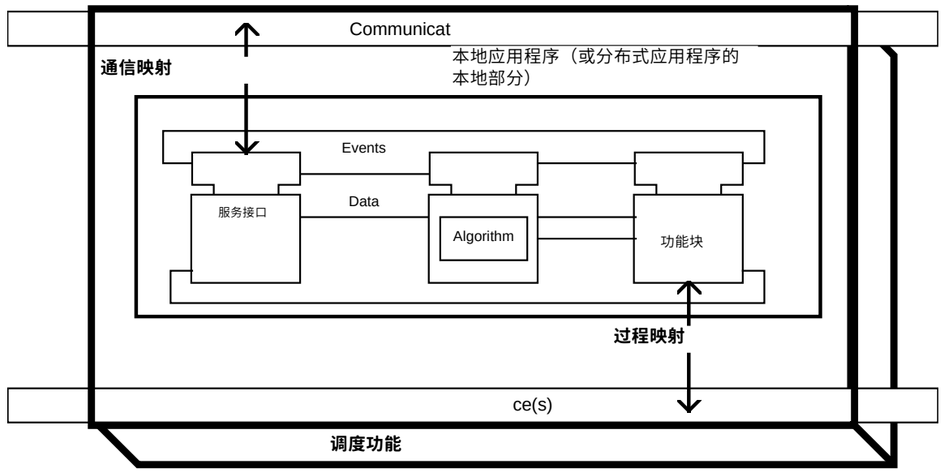
Pour les besoins du présent document, une *application* consiste en un *réseau de blocs fonctionnels,* dont les nœuds sont des *blocs fonctionnels* ou des *sous-applications* et leurs *paramètres* et dont les branches sont des *connexions de données* et des *connexions d'événements*.

Les sous-applications sont des *instances* de *type sous-application*, qui, comme les applications, sont constituées de *réseaux de blocs fonctionnels*. Les noms d'application et les *noms d'instances* de sous-application et de blocs fonctionnels peuvent donc être utilisés pour créer une hiérarchie d'*identificateurs* qui peuvent identifier de façon univoque chaque *instance* de *bloc fonctionnel* dans un *système*.

Une application peut être distribuée parmi plusieurs *ressources* dans des *équipements* identiques ou différents. Une *ressource* utilise les relations causales spécifiées par l'application pour déterminer les réponses appropriées à des *événements* qui peuvent se








注1该图仅用于说明。图形表示和功能块的位置都不是规范的。

注2：通信和过程接口可以在资源之间共享。

**Légende**

| Anglais | |
|---|---|
| s) | 通讯接口 |
| 通信映射 | 通信映射 |
| 本地应用程序（或分布式应用程序的本地部分） | 本地应用程序（或分布式应用程序的本地部分） |
| Events | Événements |
| | |
| 服务接口功能块 | 服务接口功能块 |
| | |
| 过程映射 | 过程映射 |
| | Interface(s) du processus |
| 调度功能 | Fonction de programmation |

**– Modèle d'une ressource**

**应用模型**

就本文档而言，应用程序由功能块网络组成，其节点是功能块或子应用程序及其参数，其分支是数据连接和事件连接。

子应用是类子应用的实例，与应用一样，由功能块网络组成。因此，应用程序名称以及子应用程序和功能块实例的名称可用于创建标识符层次结构，该标识符可以唯一地标识系统中的每个功能块实例。

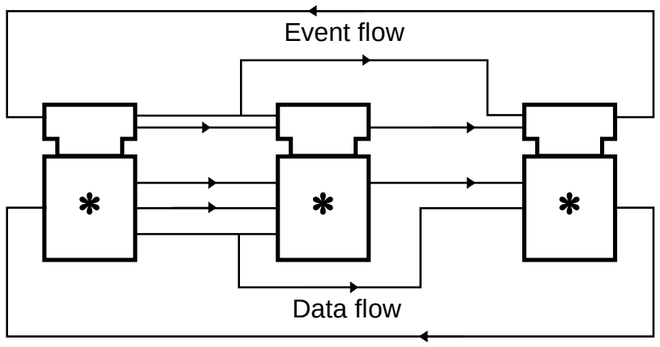
一个应用程序可以分布在相同或不同设备中的多个资源中。资源使用应用程序指定的因果关系来确定对可能发生的事件的适当响应。

produire à partir des interfaces de communication et du processus ou à partir d'autres fonctions de la ressource. Ces réponses peuvent comprendre:

- la programmation et l'*exécution* d'*algorithmes;*
- la modification de *variables;*
- la génération d'événements supplémentaires;
- les interactions avec des interfaces de communication et du processus.

Dans le contexte du présent document, les applications sont définies par des *réseaux de blocs fonctionnels* spécifiant un flot d'événements et de données parmi les *instances* de *blocs fonctionnels* ou de *sous-applications*, comme illustré à la Figure 4. Le flot d'événements détermine la programmation et l'*exécution*, par la ressource associée, des *opérations* spécifiées par le ou les *algorithmes* de chaque bloc fonctionnel, selon les règles données en 5.2.2.

Les normes, composants et systèmes conformes à la présente norme peuvent utiliser des moyens de substitution pour programmer l'exécution. Ces moyens de substitution doivent être spécifiés exactement à l'aide des éléments définis dans la présente norme.




NOTE 1 "*" représente des instances de bloc fonctionnel ou de sous-application.

NOTE 2 Cette figure est uniquement illustrative. La représentation graphique n'est pas normative.

**Légende**

| Anglais | Français |
|---|---|
| Event flow | Flot d'événements |
| Data flow | Flot de données |

**Figure 4 – Modèle d'application**

### 4.5 Modèle de bloc fonctionnel

#### 4.5.1 Caractéristiques des instances de bloc fonctionnel

Un *bloc fonctionnel (une instance de bloc fonctionnel)* est une *unité fonctionnelle* de *logiciel* comprenant une copie nommée qui lui est propre de la structure de données spécifiée par un *type de bloc fonctionnel*, qui demeure d'une *invocation* du bloc fonctionnel à la suivante. Les caractéristiques des instances de bloc fonctionnel sont décrites en 4.5.1, tandis que les spécifications des types de bloc fonctionnel sont décrites en 4.5.2.

Une *instance de bloc fonctionnel* présente les éléments caractéristiques suivants, tels qu'illustrés à la Figure 5:

- son *nom de type* et son *nom* d'*instance*;

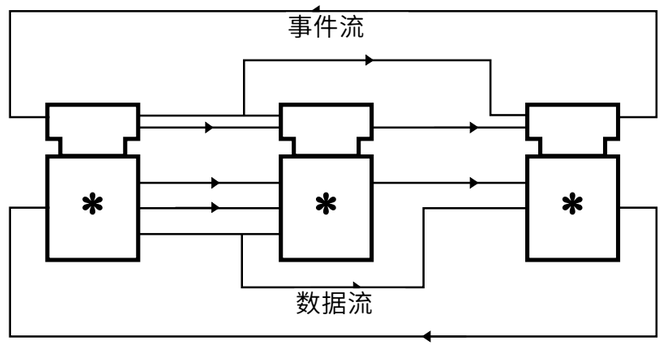


从通信和过程接口或资源的其他功能产生。这些回应可能包括：

- 编程和运行算法；
- 改变变量；
- 与通信和过程接口的交互。

在本文档的上下文中，应用程序由功能块网络定义，在功能块或子应用程序的实例之间指定事件和数据流，如图4所示。事件流决定了编程和执行，由相关资源，由每个功能块的算法指定的操作，根据5.2.2中给出的规则。

符合本标准的标准、组件和系统可以使用替代方法来安排执行。必须使用本标准中定义的元素准确指定这些替代方法。




注1"*"代表功能块或子应用实例。

注2该图仅用于说明。图形表示不是规定性的。

**Légende**

| Anglais | Français |
|---|---|
| 事件流 | Flot d'événements |
| 数据流 | Flot de données |

**e 4 – Modèle d'application**

### 4.5 功能块型号

#### 功能块实例的特征

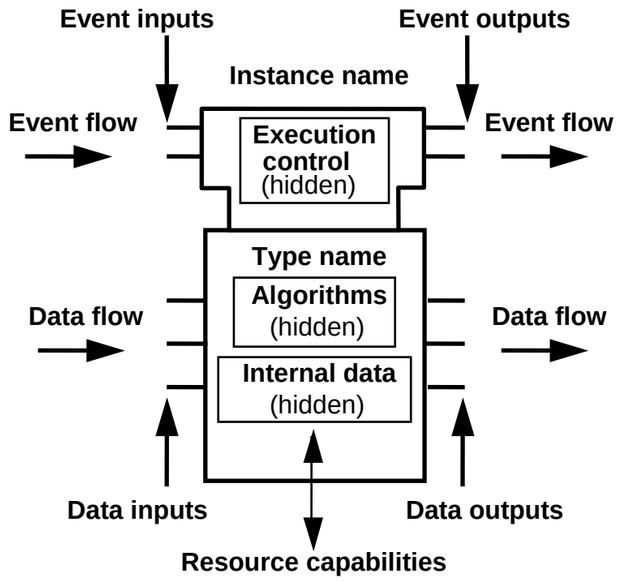
功能块（功能块实例）是软件的功能单元，包括由功能块类型指定的数据结构的唯一命名副本，该副本从功能块的一次调用持续到下一次调用。功能块实例的特征在4.5.1中描述，而功能块类型的规范在4.5.2中描述。

功能块实例具有以下特征元素，如图5所示：

- 它的类型名称和实例名称；

- un ensemble d'*entrées d'événements,* chacune de celles-ci pouvant recevoir des *événements* provenant d'une *connexion d'événements* qui peuvent altérer l'exécution d'un ou plusieurs *algorithmes*;
- un ensemble de *sorties d'événements,* chacune de celles-ci pouvant émettre des *événements* vers une *connexion d'événements* en fonction de l'exécution d'*algorithmes* ou d'autre capacité fonctionnelle de la *ressource* dans laquelle le bloc fonctionnel est situé;
- un ensemble d'*entrées de données,* qui peuvent être *mises en correspondance* avec les *variables d'entrée* correspondantes;
- un ensemble *de sorties de données,* qui peuvent être *mises en correspondance* avec les *variables de sortie* correspondantes;
- des *données* internes, qui peuvent être *mises en correspondance* avec un ensemble de *variables internes*;
- des caractéristiques fonctionnelles qui sont déterminées en combinant des données internes et/ou des informations d'état internes à un ensemble d'*algorithmes* et/ou de caractéristiques fonctionnelles de la *ressource* associée. Ces caractéristiques fonctionnelles sont définies dans la spécification du *type* de bloc fonctionnel.

NOTE Les informations d'état internes peuvent être représentées par des *variables internes* ou par une représentation interne d'un diagramme d'états de contrôle d'exécution.




(Scheduling, communication mapping, process mapping)

NOTE Cette figure est uniquement illustrative. La représentation graphique n'est pas normative.

**Légende**

| Anglais | Français |
|---|---|
| Event inputs | Entrées d'événements |
| Event outputs | Sorties d'événements |
| Instance name | Nom d'instance |
| Event flow | Flot d'événements |
| Execution Control (hidden) | Contrôle d'exécution (caché) |
| Data flow | Flot de données |

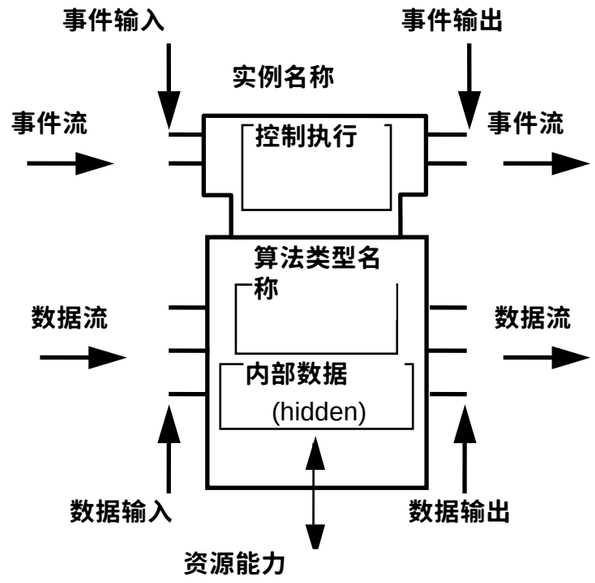




- 一组事件输入，每个输入都可以接收来自事件连接的事件，这些事件可以改变一个或多个算法的执行；
- 一组事件输出，每个事件输出可以根据算法的执行或功能块所在资源的其他功能能力向事件连接发出事件；
- 一组数据输入，可以映射到相应的输入变量；
- 一组数据输出，可以映射到相应的输出变量；
- 内部数据，可以映射到一组内部变量；
- 通过将内部数据和/或内部状态信息与一组算法和/或特性相结合来确定的功能特性

  这些功能特性在功能块类型的规范中定义。

注：内部状态信息可以用内部变量或执行控制状态图的内部表示来表示。




（调度、通信映射、流程映射）

注意此图仅用于说明。图形表示不是规定性的。

**Légende**

| Anglais | Français |
|---|---|
| 事件输入 | Entrées d'événements |
| 事件输出 | Sorties d'événements |
| 实例名称 | Nom d'instance |
| 事件流 | Flot d'événements |
| 执行控制（隐藏） | Contrôle d'exécution (caché) |
| 数据流 | Flot de données |

| Anglais | Français |
|---|---|
| Type name | Nom du type |
| Algorithms (hidden) | Algorithmes (cachés) |
| Internal data (hidden) | Données internes (cachées) |
| Data inputs | Entrées de données |
| Data outputs | Sorties de données |
| Resource capabilities | Capacités des ressources |
| (Scheduling, communication mapping, process mapping) | (Programmation, mapping de la communication, mapping du processus) |

**Figure 5 – Caractéristiques des blocs fonctionnels**

Les algorithmes contenus dans un bloc fonctionnel sont, en principe, invisibles de l'extérieur du bloc fonctionnel, avec les exceptions décrites, de façon formelle ou informelle, par le fournisseur du bloc fonctionnel. En outre, le bloc fonctionnel peut contenir des *variables* internes et/ou des informations d'état internes, qui persistent entre les invocations des algorithmes du bloc fonctionnel, mais qui ne sont pas accessibles avec les connexions des flots de données depuis l'extérieur du bloc fonctionnel.

L'accès aux variables et informations d'état internes des instances du bloc fonctionnel peut être fourni par des capacités fonctionnelles supplémentaires de la ressource associée.

Les moyens pour spécifier les relations causales parmi les entrées d'événements, les sorties d'événements et l'exécution des algorithmes sont définis aux Articles 5 et 6.

### 4.5.2 Spécifications des types de bloc fonctionnel

Un *type de bloc fonctionnel* est l'élément *logiciel* qui spécifie les caractéristiques de toutes les *instances* du type, y compris:

- son *nom de type*.
- le nombre, les noms, les noms de type et l'ordre des *entrées d'événements* et des *sorties d'événements.*
- le nombre, les noms, le *type de données* et l'ordre des *variables* d'entrée, de sortie et internes.

Les mécanismes pour la *déclaration* de ces caractéristiques sont définis en 5.2.1.

En outre, la spécification des types de bloc fonctionnel définit la fonctionnalité des *instances* du type. Cette fonctionnalité peut être exprimée comme suit:

- Pour les *types de bloc fonctionnel de base*, des mécanismes de déclaration sont donnés en 5.2.1.3 pour la spécification des *algorithmes*, qui opèrent sur les valeurs de *variables d'entrée,* de *variables de sortie* et des *variables internes* pour produire de nouvelles valeurs des *variables* de *sortie* et des *variables internes*. Les associations entre l'*invocation* des algorithmes et l'occurrence d'*événements* aux entrées et sorties d'événements sont exprimées au moyen d'un *graphe de contrôle d'exécution* (ECC), en utilisant les mécanismes de déclaration définis en 5.2.1.4.
- La fonctionnalité d'une *instance* d'un *type de bloc fonctionnel composé* ou d'un *type de sous-application* est déclarée, en utilisant respectivement les mécanismes définis en 5.3.1 et en 5.4.1, en termes de *connexions de données* et de *connexions d'événements* parmi ses *blocs fonctionnels composants* ou sous-applications et les entrées et sorties d'événements et de données du bloc fonctionnel composé ou de la sous-application.
- La fonctionnalité d'une instance d'un *type de bloc fonctionnel interface de service* est décrite par un *mapping* de *primitives de service* avec des *entrées d'événements, sorties*





| Anglais | Français |
|---|---|
| 类型名称 | 类型名称 |
| | Algorithmes (cachés) |
| 内部数据（隐藏） | Données internes (cachées) |
| 数据输入 | |
| 数据输出 | 数据输出 |
| 资源能力 | 资源能力 |
| （调度、通信映射、流程映射） | （编程、通信映射、过程映射） |

**图5 功能块的特性**

功能块中包含的算法原则上从功能块外部是不可见的，但功能块供应商正式或非正式地描述的例外情况除外。此外，功能块可能包含内部变量和/或内部状态信息，它们在功能块算法的调用之间持续存在，但不能通过来自块外部的数据流连接访问。

可以通过相关资源的附加功能能力提供对功能块实例的内部变量和状态信息的访问。

用于指定事件输入、事件输出和算法执行之间的因果关系的方法在第5章和第6章中定义。

**功能块类型规格**

功能块类型是指定该类型所有实例的特征的软件元素，包括：

- 它的类型名称。
- 事件输入和事件输出的数量、名称、类型名称和顺序。
- 输入、输出和内部变量的数量、名称、数据类型和顺序。

声明这些特性的机制在5.2.1中定义。

此外，功能块类型的规范定义了该类型实例的功能。这个功能可以表达如下：

- 对于基本功能块类型，5.2.1.3给出了算法规范的声明机制，对输入变量、输出变量和内部变量的值进行操作，产生输出变量和内部变量的新值。算法调用与事件输入和输出处事件发生之间的关联通过执行控制图(ECC)表示，使用5.2.1.4中定义的声明机制。
- 复合功能块类型或子应用程序类型实例的功能分别使用5.3.1和5.4.1中定义的机制，根据其组件功能块或子应用程序之间的数据连接和事件连接来声明应用程序以及复合功能块或子应用程序的事件和数据输入和输出。
- 服务接口功能块类型实例的功能通过服务原语与事件输入、输出的映射来描述

*d'événements, entrées de données* et *sortie de données*, en utilisant les mécanismes de déclaration définis en 6.1.

- D'autres moyens tels qu'un texte en langage naturel peuvent être utilisés pour décrire la fonctionnalité du type de bloc fonctionnel; cependant, la spécification d'un tel moyen ne relève pas du domaine d'application de la présente norme.

### 4.5.3 Modèle d'exécution pour les blocs fonctionnels de base

Comme montré à la Figure 6, l'*exécution* des *algorithmes* pour les *blocs fonctionnels de base* est invoquée par la partie **contrôle d'exécution** d'une *instance de bloc fonctionnel* en réponse à des événements au niveau des *entrées d'événements*. Cette *invocation* prend la forme d'une demande faite à la **fonction de programmation** de la *ressource* associée de programmer l'exécution des *opérations* de l'algorithme. À la fin de l'exécution d'un algorithme, le contrôle d'exécution génère zéro événement ou plus au niveau des *sorties d'événements* selon le cas.

Les *événements aux entrées d'événements* sont fournis par une connexion aux *sorties d'événements* d'autres instances de blocs fonctionnels ou de la même instance de bloc fonctionnel. Les événements sur ces sorties d'événements peuvent être générés par le contrôle d'exécution tel que décrit ci-dessus ou par le "mapping de la communication", le "mapping du processus", la "programmation" ou autre capacité fonctionnelle de la *ressource.*

NOTE 1 Le contrôle d'exécution dans des blocs fonctionnels composés est réalisé par le biais d'un flot d'événements au sein du corps du bloc fonctionnel.

La Figure 6 montre l'ordre des événements et l'exécution de l'algorithme dans le cas de l'association d'une seule entrée d'événements, d'un seul algorithme et d'une seule sortie d'événements. Les temps pertinents dans ce diagramme sont définis comme suit:

$t_1$: les valeurs des variables d'entrée pertinentes (c'est-à-dire: celles qui sont associées à l'entrée d'événements par le qualificateur WITH défini en 5.2.1.2) sont rendues disponibles;

$t_2$: l'événement sur l'entrée d'événements se produit;

$t_3$: la fonction de contrôle d'exécution notifie à la fonction de programmation de ressource de programmer un algorithme pour l'exécution;

$t_4$: l'exécution de l'algorithme commence;

$t_5$: l'algorithme parachève l'établissement des valeurs pour les variables de sortie associées à la sortie d'événements par le qualificateur WITH défini en 5.2.1.2;

$t_6$: la fonction de programmation de ressource reçoit notification que l'exécution d'algorithme est terminée;

$t_7$: la fonction de programmation invoque la fonction de contrôle d'exécution;

$t_8$: la fonction de contrôle d'exécution signale un événement sur la sortie d'événements.

Comme montré à la Figure 7, les retards significatifs de temporisation qui présentent dans ce cas un intérêt dans la conception de l'application sont:

$T_{setup}$ = $t_2 - t_1$

$T_{start}$ = $t_4 - t_2$ (temps depuis l'apparition de l'événement sur l'entrée d'événements jusqu'au début de l'exécution d'algorithme)





事件、数据输入和数据输出，使用6.1中定义的声明机制。

- 可以使用自然语言文本等其他方式来描述功能块类型的功能；但是，这种方法的规范超出了本标准的范围。

#### 基本功能块的执行模型

如图6所示，基本功能块的算法执行由功能块实例的执行控制部分调用，以响应事件输入处的事件。该调用采用对相关资源的编程功能提出请求的形式，以对算法操作的执行进行编程。在算法执行结束时，执行控件在事件输出处适当地生成零个或多个事件。

事件输入处的事件由与其他功能块实例或同一功能块实例的事件输出的连接提供。这些事件输出上的事件可以通过如上所述的执行控制或通过“通信映射”、“进程映射”、“编程”或资源的其他功能能力来生成。

注1：复合功能块中的执行控制是通过功能块主体内的事件流来实现的。

图6显示了在单个事件输入、单个算法和单个事件输出关联的情况下，事件的顺序和算法的执行情况。该图中的相关时间定义如下：

$t_1$: 相关输入变量的值（即：与由5.2.1.2中定义的WITH限定符输入的事件相关联的值）可用；

$t_2$: 事件输入上的事件发生；

$t_3$: 执行控制功能通知资源调度功能调度算法执行；

$t_4$:

$t_5$: 算法完成5.2.1.2中定义的WITH限定符对与事件输出相关的输出变量值的建立；

$t_6$: 资源调度函数接收算法执行完成的通知；

$t_7$: 调度函数调用执行控制函数；

执行控制功能在事件输出上报告一个事件。

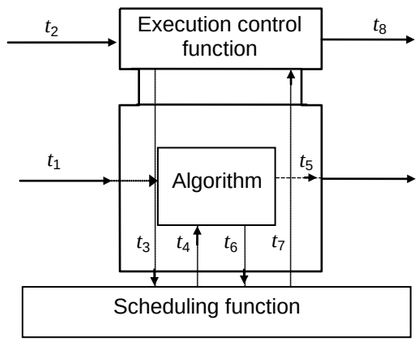
如图7所示，在这种情况下，应用设计中关注的重要时序延迟是：

$T_{setup}$

$T_{start}$ =t4 t2（从事件输入发生事件到算法开始执行的时间）

$T_{alg}$ = $t_6 - t_4$ (temps d'exécution d'algorithme)

$T_{finish}$ = $t_8 - t_6$ (temps depuis la fin de l'exécution d'algorithme jusqu'à l'apparition de l'événement sur la sortie d'événements)

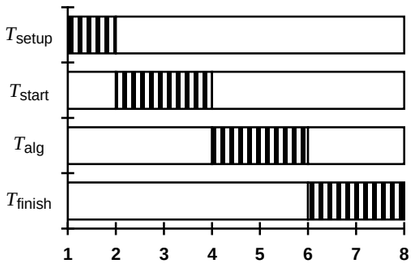



**Légende**

| Anglais | Français |
|---|---|
| Execution control function | Fonction de contrôle d'exécution |
| Algorithm | Algorithme |
| Scheduling function | Fonction de programmation |

NOTE Cette figure est uniquement illustrative. La représentation graphique n'est pas normative.

**Figure 6 – Modèle d'exécution**




NOTE les étiquettes d'axe 1, 2,... dans la figure ci-dessus correspondent aux temps $t_1$, $t_2$,... de la Figure 6.

**Figure 7 – Temporisation de l'exécution**

Les exigences spécifiques pour la présentation graphique des types de de bloc fonctionnel sont définies en 5.2.1.1.

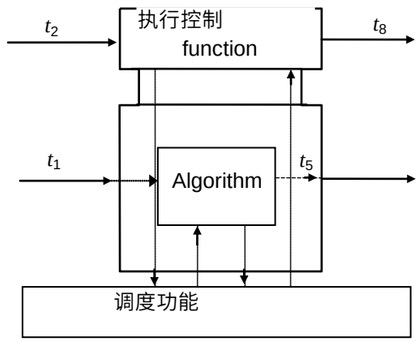
NOTE 2 En fonction du problème à résoudre, des exigences diverses pourraient exister pour la synchronisation des valeurs des *variables d'entrée* avec l'*exécution* des *algorithmes afin d'assurer la prédictibilité des résultats de l'exécution de l'algorithme*. De telles exigences pourraient comprendre, par exemple:

- l'assurance que les valeurs des variables utilisées par un algorithme restent stables pendant l'exécution de l'algorithme;





Tfinish=t8 t6（从算法执行结束到事件输出发生事件的时间）

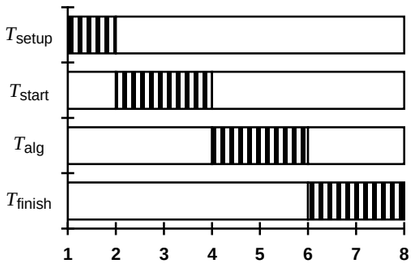



**Légende**

| | Français |
|---|---|
| 执行控制功能 | Fonction de contrôle d'exécution |
| | Algorithme |
| 调度功能 | |

注意此图仅用于说明。图形表示不是规定性的。

**Figure 6 – Modèle d'exécution**




注意上图中的轴标签1、2、…对应于图6的时间t1、t2、…。

**图7 执行时序**

5.2.1.1中定义了功能块类型图形表示的具体要求。

注2：根据要解决的问题，输入变量的值与算法执行的同步可能存在各种要求，以确保算法执行结果的可预测性。此类要求可能包括，例如：

- 确保算法使用的变量值在算法执行过程中保持稳定；

- l'assurance que les valeurs des variables utilisées par un algorithme correspondent aux données présentes lors de la survenue de l'événement sur l'entrée d'événements qui a entrainé la programmation de l'algorithme en vue d'une exécution;
- l'assurance que les valeurs des variables utilisées par tous les algorithmes programmés en vue de l'exécution dans un bloc fonctionnel correspondent aux données présentes lors de survenue de l'événement sur l'entrée d'événements qui a entrainé la programmation du premier de ces algorithmes en vue d'une exécution.

NOTE 3 Des *ressources* pouraient avoir besoin de programmer l'*exécution* d'*algorithmes* de manière *multitâche*. La spécification des attributs pour faciliter une telle programmation est décrite dans l'Annexe G.

### 4.6 Modèle de distribution

Comme illustré à la Figure 8a), une *application* ou une *sous-application* peut être distribuée en allouant ses *instances de bloc fonctionnel* à différentes *ressources* dans un ou plusieurs *équipement*. Les détails internes d'un bloc fonctionnel étant cachés à tout application ou sous-application utilisant celui-ci, un bloc fonctionnel doit former une unité atomique de distribution. Autrement dit, tous les éléments contenus dans une instance de bloc fonctionnel donnée doivent être contenus dans la même ressource.

Les relations fonctionnelles entre les blocs fonctionnels d'une application ou d'une sous-application ne doivent pas être altérées par sa distribution. Cependant, contrairement à une application ou sous-application confinée à une seule ressource, la temporisation et la fiabilité des fonctions de communications auront une incidence sur la temporisation et la fiabilité d'une application ou sous-application distribuée.

Les articles suivants s'appliquent lorsque des applications ou sous-applications sont distribuées entre plusieurs ressources:

- l'Article 6 définit les exigences pour les services de communication pour prendre en charge la distribution d'applications ou de sous-applications entre plusieurs équipements;
- l'Article 7 définit les exigences pour le cas dans lequel plusieurs applications ou sous-applications sont distribuées entre plusieurs ressources et équipements.

### 4.7 Modèle de gestion

Les Figures 8b et 8c donnent une représentation schématique de la gestion des *ressources* et des *équipements*. La Figure 8b illustre un cas dans lequel une *ressource de gestion* fournit des moyens partagés pour la gestion d'autres *ressources* au sein d'unéquipement, tandis que la Figure 8c illustre la distribution de services de gestion parmi des ressources au sein d'un équipement. Des *applications* de gestion peuvent être modélisées à l'aide des *blocs fonctionnels interface de service* et des *blocs fonctionnels de communication* **dépendants de la mise en œuvre**.

NOTE 1 Le 6.3 définit des *types de blocs fonctionnels interface de service* pour la gestion d'*applications*, et la CEI 61499-2 donne des exemples de leur utilisation.

NOTE 2 Les *applications de gestion* pourraient contenir des *instances de blocs fonctionnels interface de service* représentant des instances d'*équipement* ou de *ressource* pour les besoins d'interrogation ou de modification des *paramètres* de l'équipement ou de la ressource.





- 确保算法使用的变量的值与事件输入上发生事件时存在的数据相对应，从而导致算法被编程执行；
- 确保在功能块中为执行而编程的所有算法所使用的变量的值与事件输入上发生事件时存在的数据相对应，这导致对这些算法中的第一个进行编程以执行。

注3：资源可能需要以多任务方式调度算法的执行。
促进此类编程的属性规范在附件G中进行了描述。

**分销模式**

如图8a)所示，可以通过将其功能块实例分配给一个或多个设备中的不同资源来分发应用程序或子应用程序。由于功能块的内部细节对任何使用它的应用程序或子应用程序都是隐藏的，因此功能块必须形成一个原子分布单元。换句话说，给定功能块实例中包含的所有元素必须包含在同一资源中。

应用程序或子应用程序的功能块之间的功能关系不得因其分布而改变。然而，与受限的单一资源应用程序或子应用程序不同，通信功能的时序和可靠性将影响分布式应用程序或子应用程序的时序和可靠性。

当应用程序或子应用程序分布在多个资源中时，以下文章适用：

- 第6条定义了通信服务的要求，以支持在多个设备之间分配应用程序或子应用程序；
- 第7章定义了多个应用程序或子应用程序分布在多个资源和设备之间的情况的要求。

图8b和8c给出了资源和设备管理的示意图。图8b说明了管理资源提供用于管理设备内的其他资源的共享方式的情况，而图8c说明了管理服务在设备内的资源之间的分布。可以使用服务接口功能块和依赖于实现的通信功能块对管理应用程序进行建模。

注16.3定义了用于应用管理的服务接口功能块的类型，以及
IEC61499-2给出了它们的使用示例。

注2管理应用程序可以包含代表设备或资源实例的服务接口功能块实例，用于查询或修改设备或资源参数。

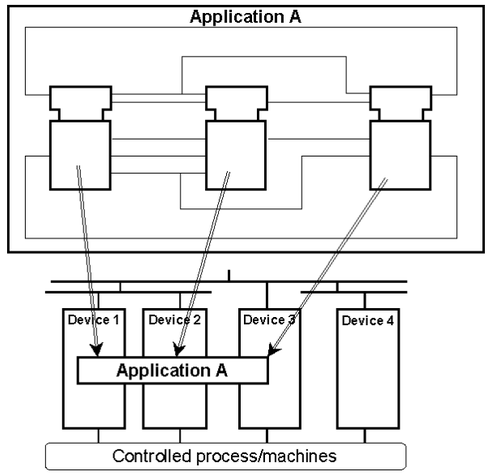



**Figure 8a – Modèle de distribution**

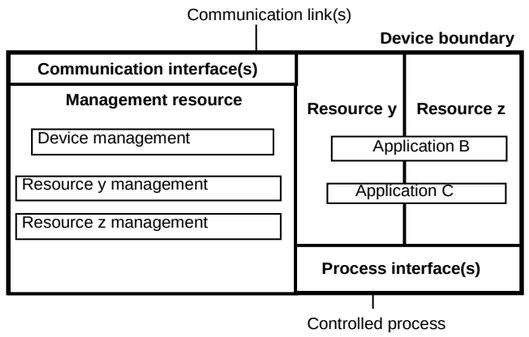



**Figure 8b – Modèle de gestion partagé**

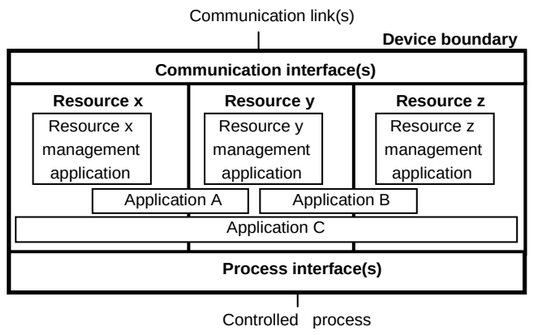



**Figure 8c – Modèle de gestion distribué**





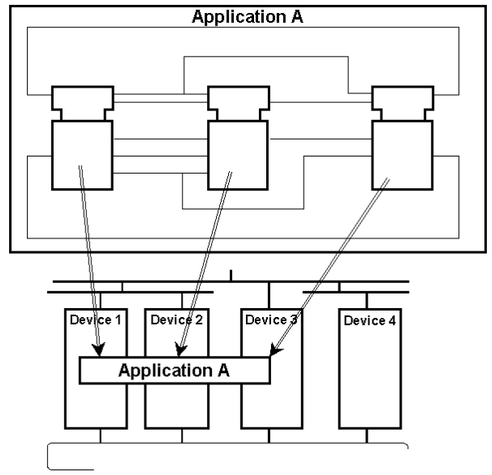



**图8a 分布模式**

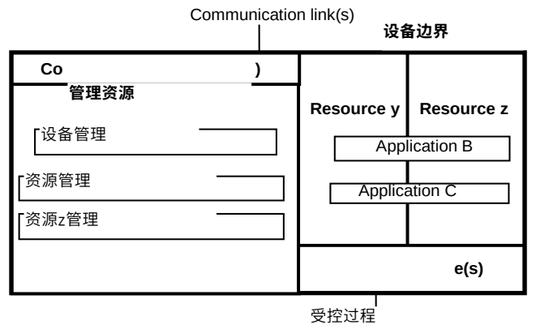



**图8b 管理模式**

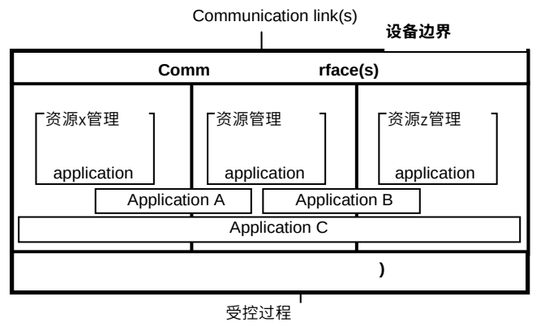



**图8c 模式分布**

**Légende**

| Anglais | Français |
|---|---|
| Application A | Application A |
| Device 1, 2, 3, 4 | Équipement 1, 2, 3, 4 |
| Controlled process/machines | Diagrammes/processus commandé |
| a) Distribution model | a) Modèle de distribution |
| Communication link(s) | Liaison(s) de communication |
| Device boundary | Limite de l'équipement |
| Communication interface(s) | Interface(s) de communication |
| Management resource | Gestion des ressources |
| Resource z Resource y | Ressource z Ressource y |
| Device management | Gestion de l'équipement |
| Application B | Application B |
| Resource y management | Gestion de la ressource y |
| Application C | Application C |
| Resource z management | Gestion de la ressource z |
| Process interface(s) | Interface(s) du processus |
| Controlled process | Processus commandé |
| b) Shared management model | b) Modèle de gestion partagée |
| Communication link(s) | Liaison(s) de communication |
| Device boundary | Limite de l'équipement |
| Communication interface(s) | Interface(s) de communication |
| Resource x | Ressource x |
| Resource y | Ressource y |
| Resource z | Ressource z |
| Process interface(s) | Interface(s) du processus |
| Resource x management application | Gestion de l'application ressource x |
| Resource y management application | Gestion de l'application ressource y |
| Resource z management application | Gestion de l'application ressource z |
| Application A | Application A |
| Application B | Application B |
| Process interface(s) | Interface(s) du processus |
| Controlled process | Processus commandé |
| c) Distributed management model | c) Modèle de gestion distribuée |

**Figure 8 – Modèles de distribution et de gestion**

### 4.8 Modèles d'état opérationnel

Tout *système* donné doit être conçu, mis en service, exploité et maintenu. Cela est modélisé par le concept du "cycle de vie" du système. Réciproquement, un système est composé de plusieurs *unités fonctionnelles* (telles que des *équipements*, des *ressources* et des *applications*), chacune de celles-ci ayant son propre cycle de vie.

Différentes actions peuvent devoir être accomplies pour prendre en charge des *unités fonctionnelles* à chaque étape du cycle de vie. Afin de caractériser quelle action peut être réalisée et de maintenir l'intégrité des unités fonctionnelles, il convient de définir les états opérationnels; par exemple: OPERATIONAL, CONFIGURABLE, LOADED, STOPPED, etc.



61499-1 © CEI:2012 – 147 –

**Légende**

| Anglais | Français |
|---|---|
| Application A | Application A |
| Device 1, 2, 3, 4 | Équipement 1, 2, 3, 4 |
| chines | mmandé |
| a)分销模式 | a)分销模式 |
| k(s) | 通讯链接 |
| 设备边界 | |
| e(s) | 通讯接口 |
| 资源管理 | 资源管理 |
| y | Ressource z Ressource y |
| 设备管理 | Gestion de l'équipement |
| | |
| 资源与管理 | 资源管理y |
| | |
| 资源z管理 | z资源管理 |
| | Interface(s) du processus |
| 受控过程 | Processus commandé |
| b)共享管理模式 | e |
| k(s) | 通讯链接 |
| 设备边界 | |
| Communication interface(s) | 通讯接口 |
| Resource x | Ressource x |
| Resource y | Ressource y |
| Resource z | Ressource z |
| | Interface(s) du processus |
| 资源x管理应用程序 | Gestion de l'application ressource x |
| 资源和管理应用程序 | Gestion de l'application ressource y |
| 资源z管理应用程序 | Gestion de l'application ressource z |
| Application A | Application A |
| Application B | Application B |
| | Interface(s) du processus |
| 受控过程 | Processus commandé |
| c)分布式管理模式 | |

**图8 分配和管理模型**

任何给定的系统都必须进行设计、调试、操作和维护。这是由系统的"生命周期"概念建模的。反之，一个系统由若干功能单元（如设备、资源和应用程序）组成，每个单元都有自己的生命周期。

在生命周期的每个阶段，可能需要执行不同的操作来支持功能单元。为了表征可以执行的操作并保持功能单元的完整性，有必要定义操作状态；例如：OPERATIONAL、CONFIGURABLE、LOADED、STOPPED等。

Chaque état opérationnel d'une unité fonctionnelle spécifie quelles actions sont autorisées, ainsi que le comportement prévu.

Un système peut être organisé de telle manière que certaines unités fonctionnelles puissent posséder ou acquérir le droit de modifier les états opérationnels d'autres unités fonctionnelles.

Des exemples d'utilisation d'états opérationnels sont:

- une unité fonctionnelle dans un état RUNNING, c'est-à-dire en exécution, peut ne pas être capable de recevoir une action de téléchargement;
- une unité fonctionnelle distribuée peut avoir besoin de maintenir un état opérationnel cohérent parmi tous ses éléments composants et développer une stratégie pour propager à travers eux les changements d'état opérationnel.

Des états opérationnels spécifiques pour des *instances de bloc fonctionnel* gérées sont définis en 6.3.2.

## 5 Spécification des types de bloc fonctionnel, de sous-application et d'adaptateurs d'interface

### 5.1 Vue d'ensemble

Comme illustré à la Figure 9, l'Article 5 définit les moyens pour la spécification du type de trois catégories de blocs:

- Le paragraphe 5.2 définit le moyen de spécifier et déterminer le comportement d'instances des *blocs fonctionnels de base*, comme illustré à la Figure 9 a). Dans ce type de bloc fonctionnel, le contrôle d'exécution est spécifié par un *graphe de contrôle d'exécution (ECC)* tandis que les *algorithmes* devant être exécutés sont déclarés comme spécifié dans les Normes conformes telles que définies dans la CEI 61499-4.
- Le paragraphe 5.3 définit le moyen de spécifier les *types de bloc fonctionnel composé*, comme illustré à la Figure 9b. Dans ce type de bloc fonctionnel, les algorithmes et leur contrôle d'exécution sont spécifiés par le biais de connexions d'événements et de données en un ou plusieurs *réseaux de blocs fonctionnels.*
- Le paragraphe 5.4 définit le moyen de spécifier les *types de sous-application*, comme illustré à la Figure 9c. Dans ce type de bloc, les algorithmes et leur contrôle d'exécution sont spécifiés comme dans le cas des types de bloc fonctionnel composé, mais avec la propriété spécifique que les *blocs fonctionnels composants* des sous-applications peuvent être distribués entre plusieurs *ressources.* Les sous-applications peuvent être imbriquées et, de ce fait, le corps d'une sous-application peut aussi contenir des *sous-applications constitutives.*

D'autres moyens peuvent être utilisés pour décrire le comportement d'instances d'un type de bloc fonctionnel. La spécification d'un tel moyen ne relève pas du domaine d'application de la présente norme; par conséquent, l'exigence est la suivante: en cas d'utilisation d'un tel moyen, un *mapping* non ambigü doit être donné entre leurs termes et les concepts et les termes et concepts correspondants de la présente norme.



功能单元的每个操作状态都指定了允许的操作以及预期的行为。

一个系统的组织方式可以使某些功能单元拥有或获得修改其他功能单元运行状态的权利。

De

- 处于RUNNING状态的功能单元，即在执行中，可能无法接收下载动作；
- 分布式功能单元可能需要在其所有组件元素之间保持一致的操作状态，并制定策略以通过它们传播操作状态更改。

托管功能块实例的特定操作状态在6.3.2中定义。

## 5 功能块、子应用和接口适配器类型的规范

如图9所示，第5条定义了指定三类块类型的方法：

- 子条款5.2定义了指定和确定基本功能块实例行为的方法，如图9a)所示。在这种类型的功能块中，执行控制由执行控制图(ECC)指定，而要执行的算法则按照IEC61499-4中定义的符合标准中的规定进行声明。
- 5.3节定义了指定复合功能块类型的方法，如图9b所示。在这种类型的功能块中，算法及其执行控制是通过一个或多个功能块网络中的事件和数据连接来指定的。
- 第5.4节定义了指定子应用程序类型的方式，如图9c所示。在这种类型的块中，算法及其执行控制被指定为复合功能块类型的情况，但具有子应用的组件功能块可以分布在多个资源中的特定属性。子应用程序可以嵌套，因此子应用程序的主体也可以包含组成子应用程序。

可以使用其他方式来描述功能块类型的实例的行为。这种方法的规范超出了本标准的范围；因此，要求是：在使用这些手段时，它们的术语和概念应与本标准的相应术语和概念之间有明确的映射关系。

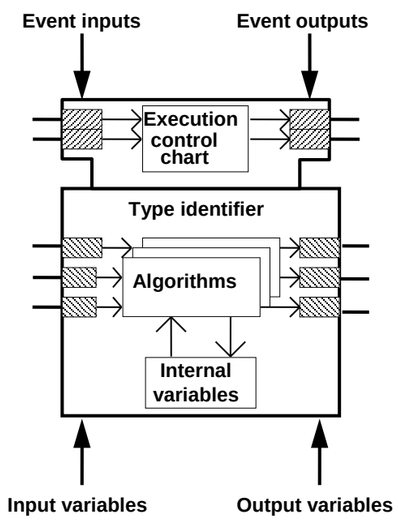



Figure 9a – Bloc fonctionnel de base (5.2)

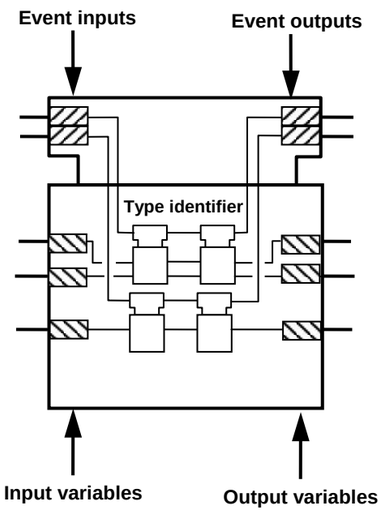



Figure 9b – Bloc fonctionnel composé (5.3)

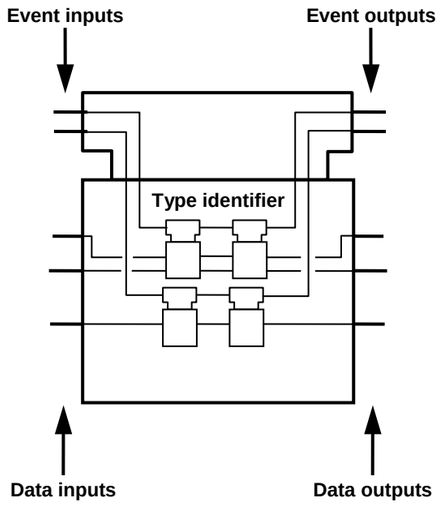



Figure 9c – Sous-application (5.4)

**Légende**

| Anglais | Français |
|---|---|
| Event Inputs | Entrées d'événements |
| Event Outputs | Sorties d'événements |
| Execution Control Chart | Graphe de contrôle d'exécution |
| Type identifier | Identificateur du type |
| Algorithms | Algorithme |
| Internal variables | Variables internes |
| Input variables | Variables d'entrée |
| Output variables | Variables de sortie |
| Data inputs | Entrées des données |
| Data outputs | Sorties des données |

NOTE Cette figure est uniquement illustrative. La représentation graphique n'est pas normative.

**Figure 9 – Types de bloc fonctionnel et de sous-application**





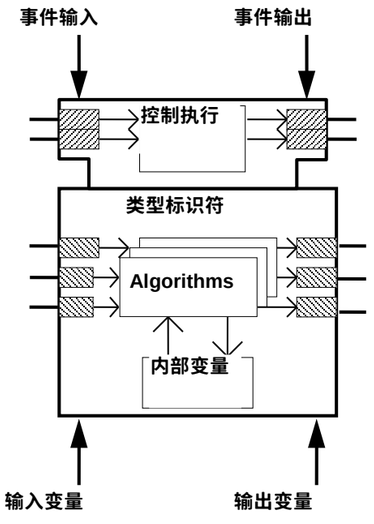



图9a Blocfonctionneldebase(5.2)

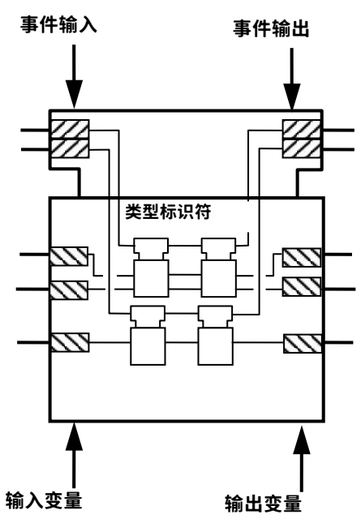



图9b Blocfonctionnelcomposé(5.3)

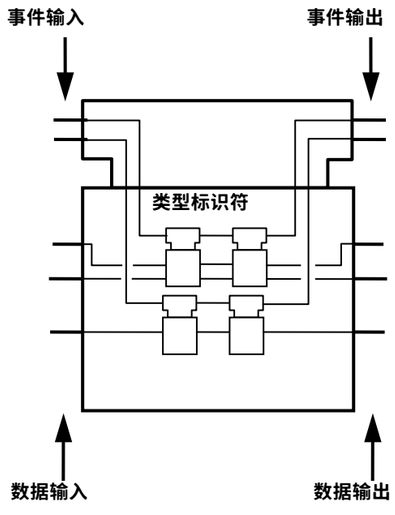



Figure 9c – Sous-application (5.4)

| Anglais | Français |
|---|---|
| 事件输入 | Entrées d'événements |
| 事件输出 | |
| 执行控制图 | exécution |
| 类型标识符 | 类型标识 |
| | Algorithme |
| 内部变量 | |
| 输入变量 | |
| 输出变量 | 变数出击 |
| 数据输入 | |
| 数据输出 | 出击次数 |

注意Cette图est唯一性说明。Lareprésentationgraphiquen'estpasnormative。

**图9 Typesdeblocfonctionneletdesous-application**

## 5.2 Blocs fonctionnels de base

### 5.2.1 Déclaration du type

#### 5.2.1.1 Généralités

Un *bloc fonctionnel de base* utilise un *graphe de contrôle d'exécution (ECC)* pour contrôler l'*exécution* de ses *algorithmes*.

Comme illustré à la Figure 10, un *type* de *bloc fonctionnel de base* peut être déclaré textuellement conformément à la syntaxe spécifiée en B.2 ou graphiquement conformément aux règles suivantes:

a) le *nom de type* du bloc fonctionnel est montré au centre en haut de la partie inférieure du bloc;

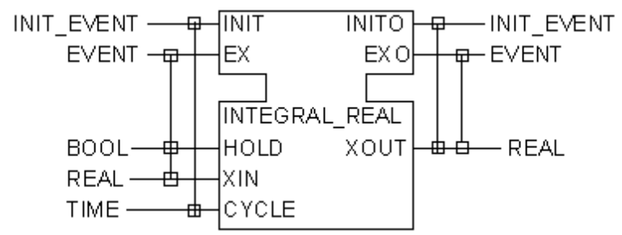
b) les noms et *déclarations de type* des *variables d'entrée* et des *supports adaptateurs* sont montrés sur le bord gauche de la partie inférieure du bloc;

c) les noms et *déclarations de type* des *variables de sortie* et des *fiches d'adaptation* sont montrés sur le bord droit de la partie inférieure du bloc;

d) l'*interface* du type du bloc fonctionnel avec les *événements* est déclarée dans la partie supérieure comme spécifié en 5.2.1.2;

e) les *algorithmes* associés au type du bloc fonctionnel sont déclarés comme spécifié en 5.2.1.3;

f) le contrôle de l'*exécution* des algorithmes associés est déclaré comme spécifié en 5.2.1.4.




NOTE 1 Voir Annexe F pour la déclaration textuelle de cet exemple.

NOTE 2 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

**Figure 10 – Déclaration du type de bloc fonctionnel de base**

#### 5.2.1.2 Déclaration de l'interface événement

Comme montré à la Figure 10, l'*interface événement* d'un *type de bloc fonctionnel de base* peut être déclarée textuellement conformément à la syntaxe donnée à l'Article B.2 ou graphiquement conformément aux règles suivantes:

a) Les *interfaces événements* sont placées dans une zone distincte en haut du bloc.

b) Les noms des *entrées d'événements* sont montrés à gauche de la partie supérieure du bloc.

c) Les noms des *sorties d'événements* sont montrés à droite de la partie supérieure du bloc.

d) Les *types* d'événement sont montrés à l'extérieur du bloc adjacent aux entrées ou sorties d'événements qui leur sont associées.

NOTE 1 Si aucun type d'événement n'est donné pour une entrée ou sortie d'événements, elle est considérée être du type par défaut EVENT.

NOTE 2 Une sortie d'événements de type EVENT peut être reliée à une entrée d'événements de n'importe quel type, et une entrée d'événements de type EVENT peut recevoir un événement de tout type.





## 5.2 基本构建块

### 5.2.1 类型声明

基本功能块使用执行控制图(ECC)来控制其算法的执行。

如图10所示，基本功能块类型可以根据B.2中指定的语法逐字声明或根据以下规则以图形方式声明：

a)功能块的类型名称显示在功能块下部的顶部中心；

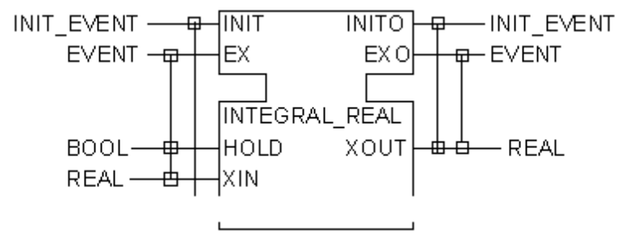
b)输入变量和适配器媒体的名称和类型声明显示在块下部的左边缘；

c)输出变量和适配器插头的名称和类型声明显示在块下部的右边缘；

d)功能块类型的事件接口在5.2.1.2中指定的顶部声明；

e)与功能块类型相关的算法按照5.2.1.3中的规定进行声明；

f) 相关算法的执行控制按照5.2.1.4中的规定进行声明。




注1：本例的文字声明见附录F。

注2这个例子只是说明性的。规格细节不是规定性的。

**图10 基本功能块类型声明**

如图10所示，基本功能块类型的事件接口可以按照B.2中给出的语法逐字声明，也可以按照以下规则图形声明：

a)事件接口放置在块顶部的单独区域中。

b)事件输入名称显示在模块顶部的左侧。

c)事件输出名称显示在块顶部的右侧。

d)事件类型显示在与相关的事件输入或输出相邻的块之外。

注1如果没有为事件输入或输出指定事件类型，则假定它是默认类型EVENT。

注2：EVENT类型的事件输出可以连接到任何类型的事件输入，EVENT类型的事件输入可以接收任何类型的事件。

NOTE 3 Une sortie d'événements de n'importe quel type autre que EVENT peut seulement être reliée à une entrée d'événements du même type ou du type EVENT.

NOTE 4 Un *type* d'événement est déclaré implicitement par son utilisation dans une déclaration d'événements.

Comme illustré à la Figure 10 et dans l'Annexe F, le qualificateur `WITH` ou un équivalent graphique doit être utilisé pour spécifier respectivement une association entre des *variables d'entrée* ou des *variables de sortie* et un *événement* sur l'*entrée d'événements* ou la *sortie d'événements* associée.

Chaque *variable d'entrée* et chaque *variable de sortie* apparaissent dans zéro, une ou plusieurs clauses `WITH` ou équivalents graphiques.

NOTE 5 Ces informations peuventêtre utilisées pour déterminer les *services* de communication requis pour *configurer* une *application* distribuée telle que décrite à l'Article 7.

NOTE 6 Une variable d'entrée qui n'apparaît dans aucune clause `WITH` n'est pas en mesure d'être reliée à une variable de sortie d'un autre bloc fonctionnel. Soit les valeurs de telles variables restent à leurs valeurs initiales déclarées, soit elles sont établies par des commandes de gestion telles que WRITE, comme décrit en 6.3.2.

NOTE 7 Une variable de sortie qui n'apparaît dans aucune clause `WITH` est en mesure d'être reliée à une variable d'entrée d'un autre bloc fonctionnel ou d'être "lue" par des commandes de gestion telles que READ, comme décrit en 6.3.2.

NOTE 8 Voir 4.5.3 pour une application du qualificateur WITH au modèle d'exécution d'un bloc fonctionnel de base.

#### 5.2.1.3 Déclaration d'algorithme

Comme montré à l'Annexe F, des *algorithmes* associés à un *type de bloc fonctionnel de base* peuvent être inclus dans la déclaration du type de bloc fonctionnel conformément aux règles pour la déclaration de la spécification du type de bloc fonctionnel donnée dans l'Annexe B. D'autres moyens peuvent aussi être utilisés pour la spécification des identificateurs et des corps des algorithmes; cependant, la spécification d'un tel moyen ne relève pas du domaine d'application de la présente norme.

La déclaration d'un algorithme peut inclure la déclaration de variables temporaires qui:

- sont seulement visibles dans le corps de l'algorithme;
- sont initialisées à chaque invocation de l'algorithme;
- peuvent être utilisées et modifiées pendant l'exécution de l'algorithme; et
- n'ont pas de valeurs qui persistent entre des exécutions de l'algorithme.

#### 5.2.1.4 Déclaration du contrôle d'exécution d'algorithme

L'ordonnancement des invocations d'algorithmes pour les *types de bloc fonctionnel de base* peut être déclaré dans la spécification type du bloc fonctionnel. Si les algorithmes d'un type de bloc fonctionnel de base sont définis comme spécifié en 5.2.1.3 (ou autrement identifiés), l'ordonnancement des invocations d'algorithme pour un tel bloc fonctionnel peut alors être sous la forme d'un *Graphe de contrôle d'exécution (ECC)* consistant en *états EC, transitions EC* et *actions EC*. Ces éléments sont représentés et interprétés comme suit:

a) l'ECC est inclus dans une section *contrôle d'exécution* de la déclaration type du bloc fonctionnel, considérée résider dans la partie supérieure du bloc;

b) l'ECC doit contenir exactement un seul *état initial EC*, représenté graphiquement comme une forme avec un contour double et un *identificateur* associé. L'état initial EC ne doit avoir aucune action EC associée;

c) l'ECC doit contenir un ou plusieurs *états EC*, représentés graphiquement comme des formes avec un contour simple, chacune avec un *identificateur* associé;

d) l'ECC peut utiliser, mais pas modifier, des variables déclarées dans la spécification type du bloc fonctionnel;



注3：除EVENT之外的任何类型的事件输出只能链接到相同类型或EVENT类型的事件输入。

注4事件类型通过在事件声明中的使用来隐式声明。

如图10和附录F所示，应使用WITH限定符或图形等效项来分别指定输入变量或输出变量与事件输入事件或相关事件输出之间的关联。

每个输入变量和每个输出变量都出现在零个或多个WITH子句或图形等效项中。

注5该信息可用于确定配置分布式应用程序所需的通信服务，如第7章所述。

注6未出现在任何WITH子句中的输入变量不能链接到另一个功能块的输出变量。此类变量的值要么保持其初始声明值，要么由管理命令（如WRITE）设置，如6.3.2所述。

注7：没有出现在任何WITH子句中的输出变量能够绑定到另一个功能块的输入变量或被管理命令（如READ）“读取”，如6.3.2中所述。

注8有关WITH限定符在基本功能块执行模型中的应用，请参见4.5.3。

如附件F所示，与基本功能块类型相关的算法可以根据附件B中给出的功能块类型规范声明规则包含在功能块类型的声明中。算法标识符和主体的规范；但是，这种方法的规范超出了本标准的范围。

算法的声明可以包括临时变量的声明，这些临时变量：

- 仅在算法主体中可见；
- 每次调用算法时都会初始化；
- 可以在算法执行过程中使用和修改；和
- 没有在算法运行之间持续存在的值。

基本功能块类型的算法调用调度可以在功能块类型规范中声明。如果基本功能块类型的算法按照5.2.1.3中的规定定义（或以其他方式标识），则此类功能块的算法调用调度可以采用执行控制图(ECC)的形式，包括EC状态、EC过渡和EC行动。这些元素的表示和解释如下：

a)ECC包含在功能块类型声明的执行控制部分中，被认为位于块的顶部；

b)ECC必须恰好包含一个初始状态EC，用图形表示为具有双轮廓和相关标识符的形状。初始EC状态必须没有关联的EC动作；

c)ECC必须包含一个或多个EC状态，用图形表示为带有简单轮廓的形状，每个都有一个相关的标识符；

d)ECC可以使用但不能修改功能块类型规范中声明的变量；

e) un *état EC* peut avoir zéro, une ou plusieurs *actions EC* associées. L'association des actions EC avec l'état EC peut être exprimée sous forme graphique ou textuelle;

f) l'*algorithme* (s'il y en a) associé à une action EC et l'*événement* (s'il y en a) devant être émis à la fin de l'algorithme doivent être exprimés sous forme graphique ou textuelle;

g) une *transition EC* est représentée sous forme graphique ou textuelle comme une liaison orientée allant d'un état EC à un autre (ou au même état);

h) chaque transition EC doit avoir une condition de transition associée, contenant une référence à un *événement*, à une *condition de garde*, ou les deux, exprimée dans la syntaxe définie pour le `ec_transition_condition` non-terminal en B.2.1.

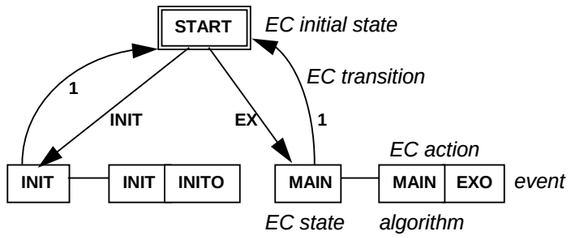
La Figure 11 illustre les éléments d'un ECC. Des déclarations textuelles similaires utilisant la syntaxe de l'Article B.2 sont données dans l'Annexe F.

NOTE 1 La notation `1` (une), illustrée à la Figure 11, est considérée être l'équivalent de `[TRUE]` représentant une condition de transition sans événement associé et avec une condition de garde qui est toujours `TRUE`.

NOTE 2 Dans ce domaine restreint, le même symbole (par exemple: `INIT`) peut être utilisé pour représenter un état EC et un nom d'algorithme, car le référent du symbole peut être déduit facilement à partir de son usage.

NOTE 3 Le texte en *italiques* ne fait pas partie de l'ECC.

NOTE 4 Une relation un à un d'événements avec des algorithmes, comme illustré par cette figure, se rencontre fréquemment, mais n'est pas le seul usage possible. Voir le Tableau A.1 pour des exemples d'autres usages: le bloc E_SPLIT montre une association de deux sorties d'événements avec un seul état, mais pas d'algorithme; E_MERGE montre une association d'un événement de sortie, mais sans algorithme, avec deux entrées d'événements; E_DEMUX montre un algorithme parmi plusieurs associés à un seul événement d'entrée; etc.




**Légende**

| Anglais | Français |
|---|---|
| algorithm | algorithme |
| EC initial state | état initial de l'EC |
| EC state | état EC |
| EC action | action EC |
| EC transition | transition EC |
| event | événement |

**Figure 11 – Exemple d'ECC**

### 5.2.2 Comportement des instances

#### 5.2.2.1 Initialisation

L'initialisation d'une *instance* de bloc fonctionnel de base par une *ressource* doit être fonctionnellement l'équivalent de la procédure suivante:

a) La valeur de chaque variable d'*entrée*, de *sortie* et *variable interne* doit être initialisée à la valeur initiale correspondante donnée dans la spécification *type* du bloc fonctionnel. Si aucune valeur initiale n'est définie, la valeur de la variable doit être initialisée à la valeur initiale par défaut définie pour le type de données de la variable.



e)一个EC状态可能有零个、一个或多个相关的EC动作。EC动作与EC状态的关联可以用图形或文字表示；

与EC动作相关的算法（如果有）和在算法结束时发出的事件（如果有）必须以图形或文本形式表示；

g)EC转换以图形或文本方式表示为从一个EC状态到另一个（或到同一状态）的有向链接；

h)每个EC转换必须有一个关联的转换条件，包含对事件、保护条件或两者的引用，以B.2.1中为非终结ec_transition_condition定义的语法表示。

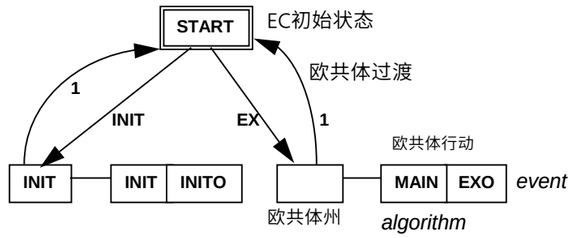
图11说明了ECC的元素。附录F中给出了使用条款B.2语法的类似文本声明。

注1：图11中所示的符号1（一）被认为是[TRUE]的等价物，表示没有关联事件的转换条件和始终为TRUE的保护条件。

注2在此受限域中，可以使用相同的符号（例如：INIT）来表示EC状态和算法名称，因为可以从其用法中很容易地推断出该符号的所指对象。

注3斜体文本不是ECC的一部分。

注4事件与算法的一对一关系，如图所示，经常遇到，但不是唯一可能的用途。有关其他用法的示例，请参见表A.1：E_SPLIT块显示两个事件输出与单个状态的关联，但没有算法；E_MERGE显示输出事件的关联，但没有算法，具有两个事件输入；E_DEMUX显示了与单个输入事件相关的几种算法之一；ETC




**Légende**

| Anglais | Français |
|---|---|
| | |
| EC初始状态 | EC的初始状态 |
| 欧共体州 | |
| 欧共体行动 | 欧共体行动 |
| 欧共体过渡 | 欧共体过渡 |
| event | |

**图11 ECC示例**

### 5.2.2 实例行为

资源对基本功能块实例的初始化在功能上等同于以下过程：

a)每个输入、输出和内部变量的值必须初始化为功能块类型规范中给定的相应初始值。如果未定义初始值，则必须将变量的值初始化为为变量的数据类型定义的默认初始值。

b) Toutes les éventuelles initialisations supplémentaires spécifiques à un algorithme doivent être effectuées; par exemple, toutes les *étapes initiales* des *Graphes séquentiels de fonction (SFC)* de la CEI 61131-3 doivent être activées et toutes les autres *étapes* doivent être désactivées.

c) L'*état initial EC* du *Graphe de contrôle d'exécution (ECC)* du bloc fonctionnel doit être activé, tous les autres *états EC* doivent être désactivés et le diagramme d'états d'opération de l'ECC défini en 5.2.2.2 doit être placé dans son état initial (`s0`).

NOTE Les conditions dans lesquelles une ressource doit accomplir une telle initialisation sont **dépendantes de la mise en œuvre.**

Le *type* de bloc fonctionnel peut aussi spécifier un *algorithme* d'initialisation devant être exécuté à l'apparition d'un événement approprié; par exemple, l'algorithme `INIT` montré à la Figure 11. Une *application* peut spécifier les conditions dans lesquelles cet algorithme doit être exécuté, par exemple en reliant la sortie d'une instance du type `E_RESTART` défini dans l'Annexe A à une entrée d'événements appropriée, par exemple l'entrée `INIT` illustrée à la Figure 10.

### 5.2.2.2 Invocation d'algorithme

L'*exécution* d'un *algorithme* associé à une *instance de bloc fonctionnel* est *invoquée* par une demande à la **fonction de programmation** de la *ressource* de programmer l'exécution des *opérations* de l'algorithme.

NOTE 1 Les opérations accomplies par un algorithme peuvent varier d'une exécution à l'autre en raison d'états internes modifiés du bloc fonctionnel, même si le bloc fonctionnel peut n'avoir qu'un seul algorithme et une seule entrée d'événements déclenchant son exécution.

L'invocation d'algorithme pour une instance d'un *type de bloc fonctionnel de base* doit être accomplie par l'équivalent fonctionnel de l'opération de son *Graphe de contrôle d'exécution (ECC)*. L'opération de l'ECC doit exposer le comportement défini par le diagramme d'états de la Figure 12 et du Tableau 1.

NOTE 2 Une conséquence de ce modèle est qu'une occurrence d'un événement à une entrée d'événements ne provoquera pas le franchissement d'une transition contenant l'événement, si la transition n'est pas associée à un état actuellement actif, c'est-à-dire, si l'événement n'est pas pertinent dans l'état donné. Cependant, l'*échantillonnage* des variables d'entrée associées à l'événement avec la construction WITH aura lieu dans tous les cas.

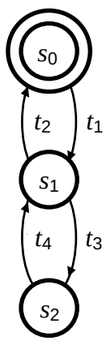



**Figure 12 – Diagrammes d'états des opérations de l'ECC**





b)应执行任何额外的特定于算法的初始化；例如，必须启用IEC61131-3顺序功能图(SFC)的所有初始步骤，并且必须禁用所有其他步骤。

c)功能块的执行控制图(ECC)的初始EC状态应启用，所有其他EC状态应禁用，并且必须将5.2.2.2中定义的ECC操作状态图置于其初始状态(s0)。

注意资源必须执行这种初始化的条件取决于实现。

功能块类型还可以指定在发生适当事件时要执行的初始化算法；例如，图11中所示的INIT算法。应用程序可以指定执行该算法的条件，例如，通过将附录A中定义的E_RESTART类型实例的输出链接到适当事件的输入，例如例如图10中所示的INIT条目。

通过对资源调度功能的请求调用与功能块实例相关联的算法的执行，以调度算法操作的执行。

注1：由于功能块的内部状态发生变化，算法执行的操作可能因运行而异，即使功能块可能只有一种算法和一个事件输入触发其执行。

基本功能块类型实例的算法调用必须通过其执行控制图(ECC)操作的功能等效来完成。ECC的操作应表现出图12和表1中状态图定义的行为。

注2：此模型的一个结果是，如果转换与当前活动状态无关，即如果事件未与在给定的状态下相关。但是，在任何情况下都将使用WITH构造对与事件关联的输入变量进行采样。

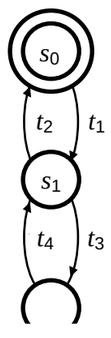



**图12 ECC操作的状态图**

**Tableau 1 – États et transitions du diagramme d'états des opérations de l'ECC**

| Etat | | Opérations |
|---|---|---|
| $s_0$ | | -- |
| $s_1$ | | évaluer les transitions[c,e] |
| $s_2$ | | accomplir des actions[d,e] |
| **Transition** | **Condition** | **Opérations** |
| $t_1$ | un événement d'entrée se produit[a] | Échantillonner des entrées[b,e] |
| $t_2$ | aucune transition n'est franchie | |
| $t_3$ | une transition est franchie | |
| $t_4$ | actions achevées | |

a La ressource doit assurer que pas plus d'un événement d'entrée ne se produit à un instant donné.

b Cette opération consiste en l'*échantillonnage* (ou son équivalent fonctionnel) des variables d'entrée associées à l'événement d'entrée courant par une déclaration WITH telle que décrite en 5.2.1.2.

c Cette opération consiste à évaluer les conditions de transition au niveau des transitions EC suivant l'état EC actif et à franchir la première transition EC (s'il y en a) pour laquelle une condition de garde (guard_condition) TRUE telle que définie en B.2.1 est rencontrée, conformément aux règles suivantes:

1 "Le franchissement de la transition EC" doit consister à désactiver son état EC précédent et à activer son état EC suivant.

2 L'ordre dans lequel les conditions de transition sont évaluées doit correspondre à l'ordre dans lequel les transitions sont déclarées comme défini en B.2.1 ou, de manière équivalente, dans la syntaxe XML définie dans la CEI 61499-2.

3 La guard_condition d'une condition de transition contenant seulement un `event_input_name` doit avoir la valeur par défaut TRUE.

4 Si l'état $s_1$ a été engagé via $t_1$, seules les conditions de transition avec l'événement d'entrée courant via son event_input_name tel que défini en B.2.1, ou les conditions de transition sans associations d'événement, doivent être évaluées.

5 Si l'état $s_1$ a été engagé via $t_4$, seules les conditions de transition sans associations d'événement doivent être évaluées.

d L'opération consiste, pour chaque *action EC* associée à l'*étape EC* active, à exécuter l'algorithme associé (s'il y en a) et à émettre un événement à la sortie d'événements associée (s'il y en a). L'ordre dans lequel les actions sont accomplies correspond à l'ordre dans lequel elles apparaissent graphiquement de haut en bas ou à l'ordre dans lequel elles sont déclarées en suivant la syntaxe textuelle définie en B.2.1 ou, de manière équivalente, dans la syntaxe XML définie dans la CEI 61499-2.

e Toutes les opérations accomplies à partir d'une occurrence de transition $t_1$ jusqu'à l'occurrence de $t_2$ doivent être mises en œuvre comme une *région critique* avec un verrou sur l'instance de bloc fonctionnel.

### 5.2.2.3 Exécution d'algorithme

L'*exécution* d'algorithme dans un bloc fonctionnel de base doit consister en l'exécution d'une séquence finie d'*opérations* déterminées par des règles **dépendant de la mise en œuvre** appropriées au langage dans lequel l'algorithme est écrit, à la *ressource* dans laquelle il s'exécute et au domaine auquel il s'applique. L'exécution d'algorithme se termine après l'exécution de la dernière opération de cette séquence.

Si un algorithme met en œuvre un diagramme d'états, des exécutions répétitives de l'algorithme sont nécessaires pour reconnaître ou accomplir des changements d'états. Normalement, il n'y a pas d'association entre ces changements d'états et l'achèvement de l'algorithme. De telles associations doivent être créées par les moyens de génération de sortie d'événements décrits en 5.2.2.2.





**表1 ECC操作状态图的状态和转换**

| Etat | | Opérations |
|---|---|---|
| $s_0$ | | |
| $s_1$ | | 评估转换c e |
| $s_2$ | | 执行动作d e |
| **Transition** | | **Opérations** |
| $t_1$ | 输入事件发生在 | Échantillonner des entrées[b,e] |
| $t_2$ | 没有过渡 | |
| $t_3$ | 进行了过渡 | |
| | | |

a 资源必须确保在任何给定时间不超过一个输入事件。

b 该操作包括通过5.2.1.2中所述的WITH语句对与当前输入事件关联的输入变量进行采样（或其功能等效项）。

c 此操作包括在活动EC状态之后评估EC转换级别的转换条件，并跨越第一个EC转换（如果有），其中满足B.2.1中定义的保护条件(guard_condition)TRUE，根据以下规则：

1 “跨越EC过渡”必须包括停用其先前的EC状态并激活其下一个EC状态。

2 评估转换条件的顺序应对应于声明转换的顺序，如B.2.1中定义，或者等效地，在IEC61499-2中定义的XML语法中。

3 仅包含event_input_name的转换条件的guard_condition必须默认为TRUE。

4 如果状态s1已通过t1进入，则仅应评估通过B.2.1中定义的event_input_name的当前输入事件的转换条件，或没有事件关联的转换条件。

如果状态s1已通过t4进入，则仅需要评估没有事件关联的转换条件。

d 对于与活动EC步骤相关联的每个EC操作，该操作包括执行相关算法（如果有）并在相关事件输出（如果有）处发出事件。执行动作的顺序对应于它们从上到下以图形方式出现的顺序，或者对应于按照B.2.1中定义的文本语法或等效地在IEC中定义的XML语法中声明它们的顺序61499-2。

e 从转换发生t1到发生t2执行的所有操作都必须实现为具有锁定功能块实例的关键区域。

基本功能块中的算法执行必须包括有限的操作序列的执行，这些操作由适用于编写算法的语言、运行资源和应用领域的依赖于实现的规则确定。算法执行在该序列的最后一个操作执行后结束。

如果算法实现了状态图，则需要重复执行算法来识别或完成状态变化。通常，这些状态变化与算法完成之间没有关联。此类关联应通过5.2.2.2中描述的事件输出生成方式创建。

### 5.3 Blocs fonctionnels composés

#### 5.3.1 Spécification de type

La déclaration des *types de bloc fonctionnel composé* doit suivre les règles données en 5.2.1 avec l'exception que des *entrées d'événements* et des *sorties d'événements* des *blocs fonctionnels composants* peuvent être interconnectées avec les entrées d'événements et les sorties d'événements du bloc fonctionnel composé pour représenter l'ordonnancement et la causalité des invocations de blocs fonctionnels. Les règles suivantes doivent s'appliquer à cet usage:

a) Chaque entrée d'événements du bloc fonctionnel composé est strictement reliée à une seule entrée d'événements d'exactement un seul bloc fonctionnel composant ou à strictement une seule sortie d'événements du bloc fonctionnel composé, avec l'exception que le raccourci graphique pour la division d'événements montrée à la Figure A.1 peut être employé.

b) Chaque entrée d'événements d'un bloc fonctionnel composant est reliée à une sortie d'événements au maximum de strictement un seul autre bloc fonctionnel composant ou à une entrée d'événements au maximum du bloc fonctionnel composé, avec l'exception que le raccourci graphique pour la fusion d'événements montrée à la Figure A.1 peut être employé.

c) Chaque sortie d'événements du bloc fonctionnel composant est reliée à une entrée d'événements au maximum de strictement un seul autre bloc fonctionnel composant ou à une sortie d'événements au maximum du bloc fonctionnel composé, avec l'exception que le raccourci graphique pour la division d'événements montrée à la Figure A.1 peut être employé.

d) Chaque sortie d'événements du bloc fonctionnel composé est reliée à strictement une seule sortie d'événements d'exactement un seul bloc fonctionnel composant ou à partir de strictement une seule entrée d'événements du bloc fonctionnel composé, avec l'exception que le raccourci graphique pour la fusion d'événements montrée à la Figure A.1 peut être employé.

e) L'utilisation du qualificateur `WITH` dans la déclaration d'entrées d'événements des types de bloc fonctionnel composé est exigée. L'utilisation du qualificateur `WITH` peut résulter en l'*échantillonnage* des entrées de données associées comme dans le cas des blocs fonctionnels de base ou d'interface de service, ou des outils logiciels peuvent fournir un moyen d'élimination des échantillonnages redondants dans la phase de mise en œuvre.

f) Les *instances* des *types sous-application* tels que définis en 5.4 ne doivent pas être utilisées dans la spécification d'un type de bloc fonctionnel composé.

Les *entrées de données* et *sorties de données* des *blocs fonctionnels composants* peuvent être interconnectées avec les entrées de données et les sorties de données du bloc fonctionnel composé pour représenter le flot de données au sein du bloc fonctionnel composé. Les règles suivantes doivent s'appliquer à cet usage:

- Chaque entrée de données du bloc fonctionnel composé peut être reliée à zéro, une ou plusieurs entrées de données de zéro, un ou plusieurs blocs fonctionnels composants et/ou à zéro, une ou plusieurs sorties de données du bloc fonctionnel composé.
- Chaque entrée de données d'un bloc fonctionnel composant peut être reliée à une sortie de données au maximum d'exactement un seul autre bloc fonctionnel composant ou à une entrée de données au maximum du bloc fonctionnel composé.
- Chaque sortie de données d'un bloc fonctionnel composant peut être reliée à zéro, une ou plusieurs entrées de données de zéro, un ou plusieurs blocs fonctionnels composants et/ou à zéro, une ou plusieurs sorties de données du bloc fonctionnel composé.
- Chaque sortie de données du bloc fonctionnel composé doit être reliée à strictement une seule sortie de données d'exactement un seul bloc fonctionnel composant ou à partir de strictement une seule entrée de données du bloc fonctionnel composé.



61499-1 © CEI:2012 – 155 –

### 5.3 Bl osés

**型号规格**

复合功能块类型的声明应遵循5.2.1中给出的规则，但组件功能块的事件输入和事件输出可以与事件输入和事件输出互连。复合功能块事件表示顺序和因果关系功能块调用。以下规则应适用于此用途：

a)复合功能块的每个事件输入都严格链接到恰好一个组件功能块的单个事件输入或严格链接到复合功能块的单个事件输出，除了图1中的事件划分快捷图A.1可以使用。

b)一个组件功能块的每个事件输入最多只能连接到一个其他组件功能块的事件输出或最多连接到复合功能块的一个事件输入，但用于合并事件的图形快捷方式除外可以使用图A.1。

c)组件功能块的每个事件输出最多连接到一个严格的另一个组件功能块的一个事件输入或最多连接到复合功能块的一个事件输出，但用于事件划分的快捷图除外可以使用图A.1。

d)复合功能块的每个事件输出都与一个组件功能块的严格单个事件输出或复合功能块的严格单个事件输入相链接，但图A.1可以使用。

e)需要在复合功能块类型的事件输入声明中使用WITH限定符。使用WITH限定符可能会导致对相关数据输入进行采样，就像在基本或服务接口功能块的情况下一样，或者软件工具可以提供一种在实施阶段消除冗余采样的方法。

f) 5.4中定义的子应用类型的实例不得用于复合功能块类型的规范中。

组件功能块的数据输入和数据输出可以与复合功能块的数据输入和数据输出互连，以表示复合功能块内的数据流。以下规则应适用于此用途：

- 复合功能块的每个数据输入可以连接到零、一个或多个零数据输入、一个或多个组件功能块和/或连接到复合功能块的零、一个或多个数据输出。
- 一个组件功能块的每个数据输入端最多可以连接一个其他组件功能块的一个数据输出端，或者最多连接一个复合功能块的一个数据输入端。
- 组件功能块的每个数据输出可以连接到零、一个或多个零、一个或多个组件功能块的数据输入和/或连接到复合功能块的零、一个或多个数据输出。
- 复合功能块的每个数据输出必须严格链接到恰好一个组件功能块的单个数据输出或来自复合功能块的严格单个数据输入。

NOTE 1 Si un élément déclaré dans une construction VAR_INPUT...END_VAR ou VAR_OUTPUT...END_VAR est respectivement associé à un événement d'entrée ou de sortie par une construction WITH, cela se traduira par la création respective d'une variable d'entrée ou de sortie associée, comme dans le cas des types de bloc fonctionnel de base. Si un tel élément n'est pas associé à un événement d'entrée ou de sortie, le flot de données associé est transmis directement à destination ou en provenance des blocs fonctionnels composants par l'intermédiaire des connexions décrites ci-dessus.

NOTE 2 Les règles pour l'interconnexion des entrées et sorties d'événements et de variables des *prises mâles* et *prises femelles* dans le corps du bloc fonctionnel composé sont les mêmes que pour l'interconnexion des entrées et sorties des *blocs fonctionnels composants*. Voir 5.5 pour des exigences supplémentaires concernant l'*adaptateur d'interface*.

La Figure 13 illustre l'application de ces règles à l'exemple du bloc fonctionnel PI_REAL. La Figure 13a montre la représentation graphique des interfaces externes, tandis que 13(b) montre la construction graphique de son corps. La Figure 14 montre les interfaces et le contrôle d'exécution pour le type du bloc fonctionnel PID_CALC utilisé dans le corps de l'exemple PI_REAL.

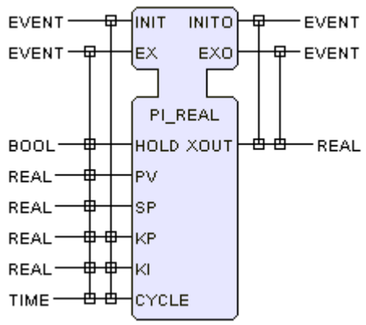



**Figure 13a – Interface externe**

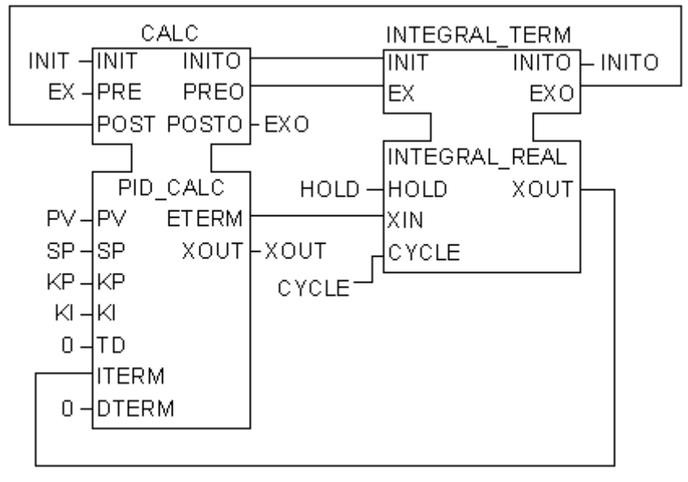



**Figure 13b – Corps graphique**

NOTE 1 Une déclaration textuelle complète de ce type de bloc fonctionnel est donnée dans l'Annexe F.

NOTE 2 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

**Figure 13 – Exemple de bloc fonctionnel composé PI_REAL**

VAR_INPUT…END_VAR或VAR_OUTPUT…END_VAR分别通过WITH构造与输入或输出事件相关联，这将导致相关输入或输出变量的相应创建，就像基本功能块类型的情况一样。如果这样的元素不与输入或输出事件相关联，则相关联的数据流通过上述连接直接传输到组件功能块或从组件功能块传输。

注2：复合功能块主体中插头和插座的事件和变量输入输出互连规则与组件功能块输入输出互连规则相同。有关其他接口适配器要求，请参见5.5。

图13说明了这些规则在PI_REAL功能块示例中的应用。图13a显示了外部接口的图形表示，而13(b)显示了其主体的图形结构。图14显示了PI_REAL示例主体中使用的PID_CALC功能块类型的接口和执行控制。

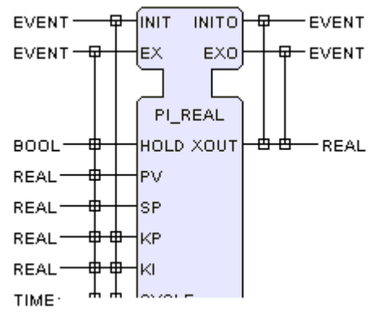



**图13a–外部接口**

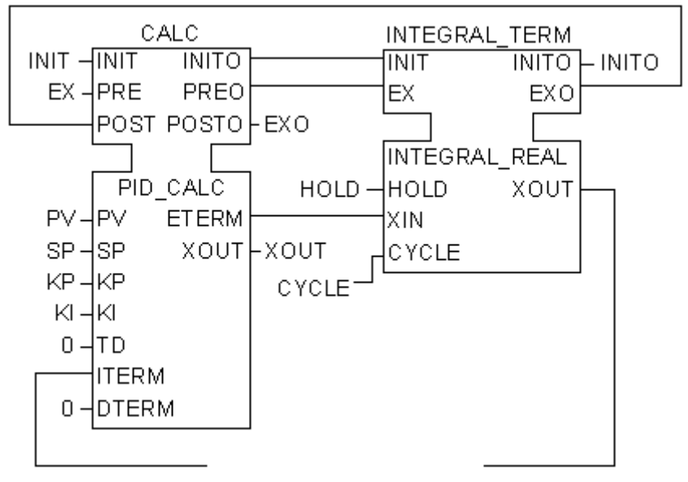



**图13b 车身图形**

注1此类功能块的完整文本声明在附录F中给出。

注2这个例子只是说明性的。规格细节不是规定性的。

**图13–PI_REAL复合功能块示例**

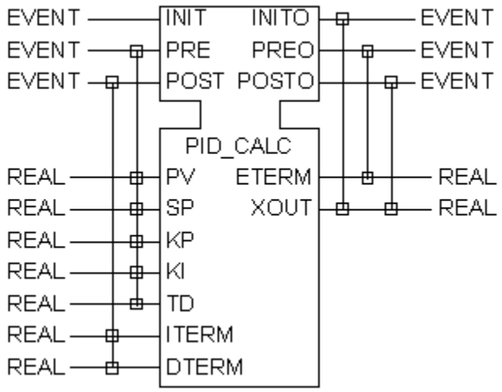



**Figure 14a – Interface externe**

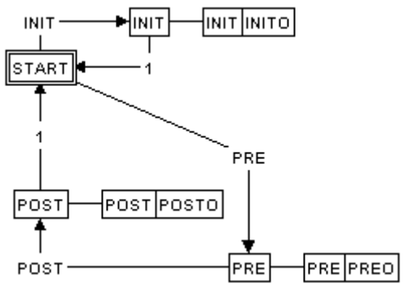



**Figure 14b – Contrôle d'exécution**

NOTE Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

**Figure 14 – Exemple de bloc fonctionnel de base `PID_CALC`**

### 5.3.2 Comportement d'instances

L'*invocation* et l'*exécution* des *blocs fonctionnels composants* dans des blocs fonctionnels composés doivent être accomplies comme suit:

a) Si une *entrée d'événements* du bloc fonctionnel composé est reliée à une *sortie d'événements* du bloc, l'apparition d'un *événement* sur l'entrée d'événements doit entraîner la génération d'un événement à la sortie d'événements associée.

b) Si une entrée d'événements du bloc fonctionnel composé est reliée à une entrée d'événements d'un bloc fonctionnel composant, l'apparition d'un événement sur l'entrée d'événements du bloc fonctionnel composé doit entraîner la programmation d'une invocation de la fonction de contrôle d'exécution du bloc fonctionnel composant, avec une apparition d'un événement sur l'entrée d'événements associée du bloc fonctionnel composant.

c) Si une sortie d'événements d'un bloc fonctionnel composant est reliée à une entrée d'événements d'un second bloc fonctionnel composant, l'apparition d'un événement sur la sortie d'événements du premier bloc doit entraîner la programmation d'une invocation de la fonction de contrôle d'exécution du second bloc, avec une apparition d'un événement sur l'entrée d'événements associée du second bloc.



61499-1 © CEI:2012 – 157 –

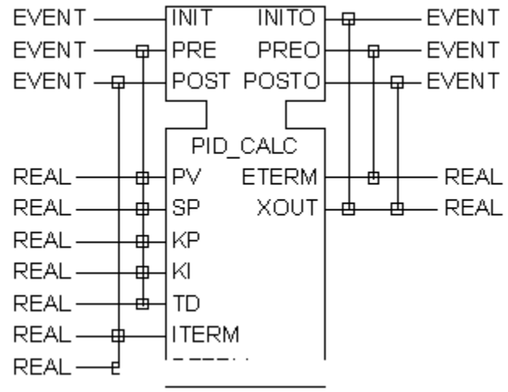
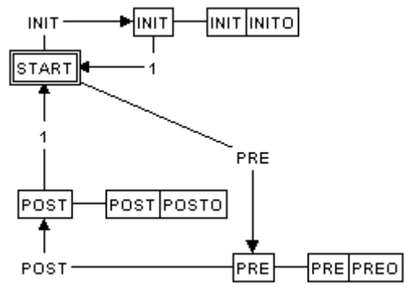



**图14a–外部接口**




注意这个例子只是说明性的。规格细节不是规定性的。

**图14 基本PID_CALC功能块示例**

复合功能块内组件功能块的调用和执行应按如下方式完成：

a)如果复合功能块的事件输入链接到块的事件输出，则事件输入上的事件的出现必须导致在相关事件的输出处产生事件。

b)如果复合功能块的事件输入链接到组件功能块的事件输入，则复合功能块的事件输入上的事件的发生必须引起对执行控制功能的调用的编程组件功能块，在组件功能块的相关事件输入上出现事件。

c)如果一个组件功能块的事件输出链接到第二个组件功能块的事件输入，则在第一个功能块的事件输出上出现事件必须导致编程调用第二个功能块的执行控制功能块，在第二个块的关联事件输入上出现事件。

d) Si une sortie d'événements d'un bloc fonctionnel composant est reliée à une sortie d'événements du bloc fonctionnel composé, l'apparition d'un événement sur la sortie d'événements du bloc composant doit entraîner la génération d'un événement sur la sortie d'événements associée du bloc fonctionnel composé.

L'initialisation d'instances de blocs fonctionnels composés doit équivaloir à l'initialisation de leurs blocs fonctionnels composants conformément aux dispositions en 5.2.2.1.

### 5.4 Sous-applications

#### 5.4.1 Spécification de type

La déclaration des *types de sous-application* est semblable à la déclaration des *types de bloc fonctionnel composé* tels que définis en 5.3.1, à l'exception du fait que les mots-clés de délimitation doivent être `SUBAPPLICATION..END_SUBAPPLICATION`. Les règles suivantes doivent s'appliquer à cet usage:

a) Le qualificateur `WITH` n'est pas utilisé dans la déclaration des entrées d'événements et des sorties d'événements des *types sous-application*.

b) Chaque entrée d'événements de la sous-application doit être reliée strictement à une seule entrée d'événements d'exactement un seul bloc fonctionnel composant ou une seule sous-application constitutive ou bien à strictement une seule sortie d'événements de la sous-application.

c) Chaque entrée d'événements d'un bloc fonctionnel composant ou d'une sous-application constitutive est reliée à une sortie d'événements au maximum de strictement un seul autre bloc fonctionnel composant ou une seule autre sous-application constitutive ou bien au maximum à une entrée d'événements de la sous-application.

d) Chaque sortie d'événements d'un bloc fonctionnel composant ou d'une sous-application constitutive est reliée à une entrée d'événements au maximum de strictement un seul autre bloc fonctionnel composant ou une seule autre sous-application constitutive ou bien au maximum à une sortie d'événements de la sous-application.

e) Chaque sortie d'événements de la sous-application est reliée à partir de strictement une seule sortie d'événements d'exactement un seul bloc fonctionnel composant ou une seule sous-application constitutive ou bien à partir de strictement une seule entrée d'événements de la sous-application.

NOTE 1 Les blocs fonctionnels composants peuvent'inclure des instances des blocs de traitement d'événements définis dans l'Annexe A, par exemple pour "diviser" des événements en utilisant des instances du bloc `E_SPLIT`, pour "fusionner" des événements en utilisant le bloc `E_MERGE`, ou pour les deux cas, en utilisant le raccourci graphique équivalent.

Les *entrées de données* et *sorties de données* des *blocs fonctionnels composants* ou des *sous-applications constitutives* peuvent être interconnectées avec les entrées de données et les sorties de données de la sous-application pour représenter le flot de données au sein de la sous-application. Les règles suivantes doivent s'appliquer à cet usage:

- Chaque entrée de données de la sous-application peut être reliée à zéro, une ou plusieurs entrées de données de zéro, un ou plusieurs blocs fonctionnels composants ou de zéro, une ou plusieurs sous-applications constitutives et/ou à zéro, une ou plusieurs sorties de données de la sous-application.
- Chaque entrée de données d'un bloc fonctionnel composant ou d'une sous-application constitutive peut être reliée à une sortie de données au maximum d'exactement un seul autre bloc fonctionnel composant ou une seule autre sous-application constitutive ou bien à une entrée de données au maximum de la sous-application.
- Chaque sortie de données d'un bloc fonctionnel composant ou d'une sous-application constitutive peut être reliée à zéro, une ou plusieurs entrées de données de zéro, un ou plusieurs blocs fonctionnels composants ou de zéro, une ou plusieurs sous-applications constitutives et/ou à zéro, une ou plusieurs sorties de données de la sous-application.
- Chaque sortie de données de la sous-application doit être reliée à partir d'exactement une seule sortie de données d'exactement un seul bloc fonctionnel composant ou une seule



d)如果组件功能块的事件输出链接到复合功能块的事件输出，则在组件块的事件输出上出现事件必须导致在相关联的事件输出上产生事件复合功能块。

复合功能块实例的初始化应等同于按照5.2.2.1的规定对其组成功能块的初始化。

### 5.4 S

**型号规格**

子应用类型的声明与5.3.1中定义的复合功能块类型的声明类似，只是分隔关键字必须是SUBAPPLICATION..END_SUBAPPLICATION。以下规则应适用于此用途：

a)WITH限定符不用于子应用程序类型的事件输入和事件输出的声明。

b)子应用程序的每个事件输入必须严格链接到恰好一个组件功能块或单个组成子应用程序的单个事件输入，或者严格链接到子应用程序的单个事件输出。

c)组件功能块或组成子应用程序的每个事件输入都链接到最多一个其他组件功能块或单个其他组成子应用程序的事件输出，或者最多链接到一个子应用程序事件输入。

d)组件功能块或组成子应用程序的每个事件输出都链接到最严格的一个其他组件功能块或一个其他组成子应用程序的事件输入，或者最多链接到一个子应用程序事件输出。

e)子应用程序的每个事件输出都与一个组件功能块或单个组成子应用程序的严格的单个事件输出或子应用程序事件的严格的单个输入相链接。

注1：组件功能块可以包括附件A中定义的事件处理块的实例，例如使用E_SPLIT块的实例"拆分"事件，使用E_MERGE块"合并"事件，或在这两种情况下，使用等效的图形快捷方式。

组件功能块或组成子应用程序的数据输入和数据输出可以与子应用程序的数据输入和数据输出互连，以表示子应用程序内的数据流。以下规则应适用于此用途：

- 子应用程序的每个数据输入可以链接到零、一个或多个零、一个或多个组件功能块或零、一个或多个组件子应用程序的数据输入和/或零、一个或多个输出子应用程序。应用数据。
- 组件功能块或组成子应用程序的每个数据输入可以连接到最多恰好一个其他组件功能块或单个其他组成子应用程序的数据输出，或者连接到最大数据输入。子应用。
- 组件功能块或组成子应用程序的每个数据输出可连接到零、一个或多个数据输入，零、一个或多个组件功能块或零、一个或多个组成子应用程序和/或零，子应用程序的一个或多个数据输出。
- 子应用程序的每个数据输出必须与恰好一个组件功能块的一个数据输出或一个

sous-application constitutive ou bien à partir d'exactement une seule entrée de données de la sous-application.

NOTE 2 Bien que les constructions VAR_INPUT...END_VAR et VAR_OUTPUT...END_VAR soient utilisées pour la déclaration des entrées et sorties de données des types de sous-application, cela ne se traduit pas par la création de variables d'entrée et de sortie; le flot de données est au contraire transmis aux blocs fonctionnels composants ou aux sous-applications constitutives par les connexions décrites ci-dessus.

NOTE 3 Les règles pour l'interconnexion des entrées et sorties d'événements et de variables des *prises mâles* et *prises femelles* dans le corps de la sous-application sont les mêmes que pour l'interconnexion des entrées et sorties des *blocs fonctionnels composants*. Voir 5.5 pour des exigences supplémentaires concernant l'*adaptateur d'interface*.

EXEMPLE La Figure 15 illustre l'application de ces règles à l'exemple de sous-application PI_REAL_APPL. La Figure 15 a) montre la représentation graphique de ses interfaces externes, tandis que la Figure 15 b) montre la construction graphique de son corps. Le corps de la sous-application PI_REAL_APPL utilise le type de bloc fonctionnel PID_CALC issu de l'exemple de bloc fonctionnel composé en 5.3.1, qui est montré à la Figure 14.

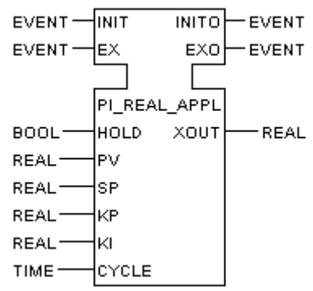



**Figure 15a – Interface externe**

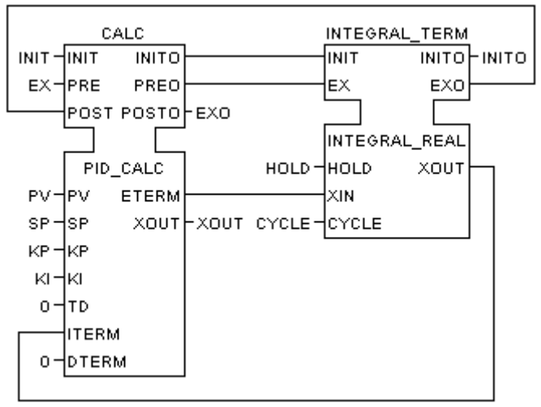



**Figure 15b – Corps graphique**

NOTE 1 Une déclaration textuelle complète de cette sous-application est donnée dans l'Annexe F.

NOTE 2 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

**Figure 15 – Exemple de sous-application PI_REAL_APPL**

### 5.4.2 Comportement d'instances

L'*invocation* des *opérations* des *blocs fonctionnels composants* ou des *sous-applications constitutives* au sein des *sous-applications* doit être accomplie comme suit:





子应用程序或子应用程序的一个数据条目。

注2虽然VAR_INPUT…END_VAR和VAR_OUTPUT…END_VAR构造用于声明子应用程序类型的数据输入和输出，但这不会导致输入变量的创建和退出；相反，数据流通过上述连接传输到组件功能块或组成子应用程序。

注3：子应用主体中插头和插座的事件和变量输入输出互连规则与组件功能块输入输出互连规则相同。有关其他接口适配器要求，请参见5.5。

示例图15说明了将这些规则应用于示例PI_REAL_APPL子应用程序。图15a)显示了其外部接口的图形表示，而图15b)显示了其主体的图形结构。PI_REAL_APPL子应用程序主体使用5.3.1中复合功能块示例中的PID_CALC功能块类型，如图14所示。

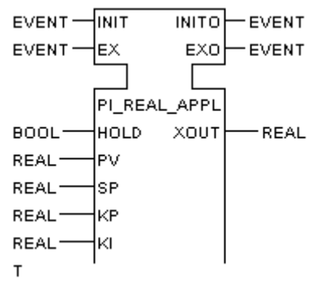



**图15a–外部接口**

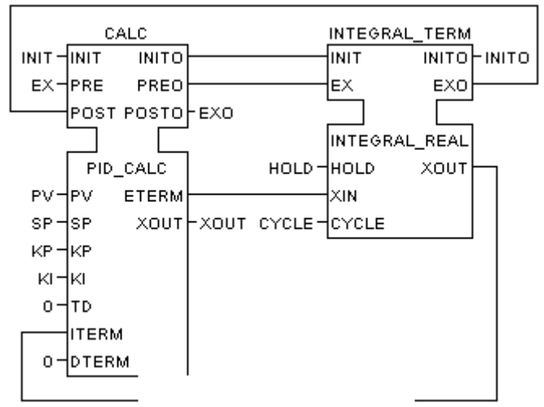



**图15b 车身图形**

注1：附件F中给出了该子应用的完整文本声明。

注2这个例子只是说明性的。规格细节不是规定性的。

**图15 PI_REAL_APPL子应用示例**

子应用程序内的组件功能块或组成子应用程序的调用操作应完成如下：

a) Si une *entrée d'événements* de la sous-application est reliée à une *sortie d'événements* du bloc, l'apparition d'un *événement* sur l'entrée d'événements doit entraîner la génération d'un événement sur la sortie d'événements associée.

b) Si une entrée d'événements de la sous-application est reliée à une entrée d'événements d'un bloc fonctionnel composant ou d'une sous-application constitutive, l'apparition d'un événement sur l'entrée d'événements de la sous-application doit entraîner la programmation d'une invocation de la fonction de contrôle d'exécution du bloc fonctionnel composant ou de la sous-application constitutive, avec une apparition d'un événement sur l'entrée d'événements associée du bloc fonctionnel composant ou de la sous-application constitutive.

c) Si une sortie d'événements d'un élément bloc fonctionnel composant ou d'une sous-application constitutive est reliée à une entrée d'événements d'un second élément bloc fonctionnel composant ou d'une seconde sous-application constitutive, l'apparition d'un événement sur la sortie d'événements du premier bloc doit entraîner la programmation d'une invocation de la fonction de contrôle d'exécution du second bloc, avec une apparition d'un événement sur l'entrée d'événements associée du second bloc.

d) Si une sortie d'événements d'un bloc fonctionnel composant ou d'une sous-application constitutive est reliée à une sortie d'événements de la sous-application, l'apparition d'un événement sur la sortie d'événements du bloc fonctionnel composant doit entraîner la génération d'un événement sur la sortie d'événements associée de la sous-application.

Étant donné que les sous-applications ne créent pas des variables de façon explicite, aucune procédure d'initialisation spécifique n'est applicable aux instances de sous-application.

### 5.5 Adaptateur d'interface

#### 5.5.1 Principes généraux

Des *adaptateurs d'interface* peuvent être utilisés pour donner une représentation compacte d'un ensemble spécifié de flots d'événements et de données. Comme illustré à la Figure 16, un *type adaptateur d'interface* fournit un moyen de définir un sous-ensemble (la *fiche d'adaptation)* des *entrées* et des *sorties* d'un bloc fonctionnel du *fournisseur* qui peut être inséré en un sous-ensemble conjugué de *sorties* et d'*entrées* (le *support adaptateur)* d'un bloc fonctionnel *utilisateur*. Ainsi, l'adaptateur interface représente les chemins d'événements et de données par lesquels le fournisseur fournit un *service* vers l'utilisateur, ou vice versa, en fonction des profils d'interactions fournisseur/utilisateur, qui peuvent être représentées par des séquences de *primitives de service* telles que décrites en 6.1.3.

NOTE Un *type de bloc fonctionnel* donné pourrait fonctionner comme un *fournisseur,* un *utilisateur*, ou les deux, ou ni l'un ni l'autre, et de contenir plus d'une instance de *prise mâle* ou de *prise femelle* d'un ou plusieurs *types d'adaptateur d'interface.*



a)如果子应用程序的事件输入链接到块的事件输出，则事件输入上的事件的出现必须导致相关事件的输出上的事件的产生。

b)如果子应用程序的事件输入链接到组件功能块或组成子应用程序的事件输入，则事件发生在

必须导致对组件功能块或组成子应用程序的执行控制功能的调用进行编程，并在组件功能块或组成子应用程序的相关事件输入上出现事件。

c)如果组件功能块元素或组成子应用程序的事件输出连接到第二组件功能块元素或第二组成子应用程序的事件输入，则事件输出上的事件发生第一个块的执行必须导致对第二个块的执行控制功能的调用的编程，并在第二个块的相关事件输入上出现一个事件。

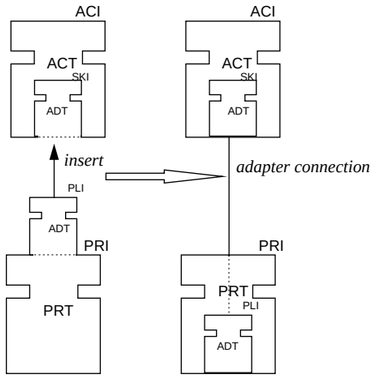
d)如果组件功能块或组成子应用程序的事件输出链接到子应用程序的事件输出，则组件功能块的事件输出上的事件发生必须导致事件在子应用程序的关联事件输出上生成。

由于子应用程序没有显式创建变量，因此没有适用于子应用程序实例的特定初始化过程。

### 5.5 Adaptateur d'interface

接口适配器可用于给出一组指定事件和数据流的紧凑表示。如图16所示，接口适配器类型提供了一种定义提供程序功能块的输入和输出子集（适配器插头）的方法，该提供程序功能块可以插入到输出和输入的子集共轭（适配器支持）中。一个用户功能块。因此，接口适配器代表事件和数据路径，提供者通过它们向用户提供服务，反之亦然，基于提供者–用户交互配置文件，它可以由6.1.3中描述的原语服务序列表示。

注：一个给定的功能块类型可以作为提供者、用户或两者兼而有之，或者两者都不是，并且包含一种或多种接口适配器类型的多个插头或插座实例。




**Légende**

| Anglais | Français |
|---|---|
| insert | insertion |
| adapter connection | connexion de l'adaptateur |

**Légende**

PRT Provider type (*type de fournisseur*),

PRI Provider instance (*instance de fournisseur*),

ACT Acceptor type (*type d'utilisateur*),

ACI Acceptor instance (*instance de l'utilisateur*),

ADT Adapter type (*type d'adaptateur*),

PLI Plug instance (*instance de prise mâle*),

SKI Socket instance (*instance de prise femelle*)

NOTE Cette figure est uniquement illustrative. La représentation graphique n'est pas normative.

**Figure 16 – Adaptateur d'interface – Modèle conceptuel**

### 5.5.2 Spécification de type

Une *déclaration de type adaptateur d'interface* doit définir seulement le nom du *type d'interface* et ses interfaces d'*événements* et de *données* contenues. Ceux-ci sont définis graphiquement ou textuellement de la même manière que le *nom de type, les interfaces d'événement* et les *interfaces de données* d'un *type de bloc fonctionnel de base* tels que définis en 5.2.1.1 et 5.2.1.2, à l'exception du fait que les mots-clés pour le début et la fin de la déclaration textuelle de type doivent être ADAPTER...END_ADAPTER. La syntaxe textuelle pour la déclaration des adaptateurs d'interface est donnée à l'Article B.7.

EXEMPLE L'adaptateur d'interface illustré à la Figure 17 représente le fonctionnement du transfert d'un équipement de transfert "amont" représenté par un *fournisseur* de la *fiche* d'adaptation vers un équipement "aval" représenté par un *utilisateur* avec un *support* adaptateur correspondant. Comme illustré à la Figure 17(b), le fonctionnement typique de cette interaction consiste en la séquence suivante:

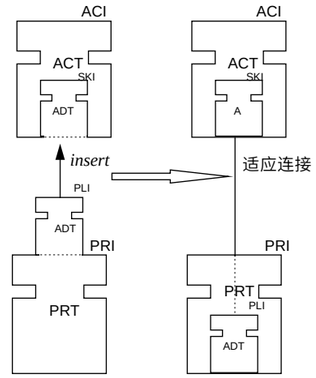
a) Un événement dans l'équipement amont (par exemple: l'arrivée d'une pièce à travailler à la position de déchargement) entraîne qu'un événement LD, typiquement interprété comme une commande de "chargement", est émis vers l'équipement aval. Associée à cet événement, une valeur de capteur WO indique si, oui ou non, une pièce à travailler est effectivement présente pour le transfert, plus une certaine propriété mesurée ou un certain ensemble de propriétés de la pièce à travailler, en l'occurrence sa couleur.

b) Un événement ultérieur dans l'équipement aval (par exemple: l'achèvement du montage de la charge) entraîne qu'un événement UNLD, typiquement interprété comme une commande de libération de la pièce à travailler, est envoyé vers l'équipement amont.

c) Ensuite, un événement CNF, typiquement interprété comme une confirmation de la libération de la pièce à travailler, est transmis de l'équipement amont vers l'équipement aval afin de parachever l'opération. À ce stade, la sortie WO est typiquement FALSE et la valeur de la sortie WKPC n'a pas de signification.



61499-1 © CEI:2012 – 161 –




**Légende**

| Anglais | Français |
|---|---|
| | |
| 适应连接 | 适配器连接 |

**Légen**

PRT 提供者类型，

PRI 提供者实例（提供者实例），

ACT

ACI 接受者实例（用户实例），

ADT 适配器类型（适配器类型），

PLI 插件实例，

套接字实例

注意此图仅用于说明。图形表示不是规定性的。

**aptateur d'interface – Modèle conceptuel**

**型号规格**

接口适配器类型声明只需要定义接口类型的名称及其包含的事件和数据接口。它们以图形或文本的方式定义，与5.2.1.1和5.2.1.2中定义的基本功能块类型的类型名称、事件接口和数据接口相同，除了文本开始和结束的关键字类型声明必须是ADAPTER…END_ADAPTER。声明接口适配器的文本语法在第B.7节中给出。

示例图17所示的接口适配器表示将由适配器插头的供应商代表的"上游"传输设备转移到由具有相应适配器支持的用户代表的"下游"设备的操作。如图17(b)所示，这种交互的典型操作由以下序列组成：

a)上游设备中的事件（例如：工件到达卸载位置）导致LD事件，通常解释为"加载"命令，被发布到下游设备。与此事件相关联，WO传感器值指示工件是否实际存在用于转移，加上工件的一些测量属性或一组属性，在本例中为它的颜色。

b)下游设备中的后续事件（例如：负载安装完成）导致UNLD事件（通常解释为工件释放命令）被发送到上游设备。

c)接下来，一个CNF事件，通常解释为工件释放的确认，从上游设备传输到下游设备，以完成操作。此时，WO输出通常为FALSE，并且WKPC输出的值没有意义。

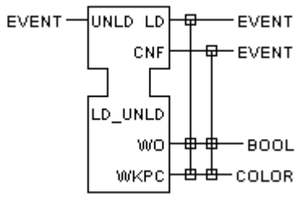



**Figure 17a – Interface**

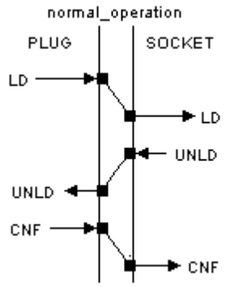



**Figure 17b – Séquence de service**

NOTE 1 Une déclaration textuelle complète de ce type d'adaptateur est donnée dans l'Annexe F.

NOTE 2 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

NOTE 3 Voir 6.1.2 pour une explication des séquences de service.

**Figure 17 – Déclaration du type d'adaptateur – Exemple graphique**

### 5.5.3 Usage

L'usage des *types* et des *instances adaptateur d'interface* doit être conforme aux règles suivantes:

a) Les instances adaptateur d'interface devant être utilisées comme *prises mâles* dans des instances d'un *type de bloc fonctionnel* doivent être déclarées dans sa *déclaration de type* dans un bloc `PLUGS...END_PLUGS`, déclarant le *nom d'instance* et le *type adaptateur d'interface* de chaque prise mâle. Dans la représentation graphique des *types de bloc fonctionnel* et des *instances*, les fiches sont montrées comme *variables de sortie* avec une indication textuelle ou graphique spécialisée pour signifier qu'elles ne sont pas des variables de sortie ordinaires.

b) Les instances adaptateur d'interface devant être utilisées comme *prises femelles* dans des instances d'un *type bloc fonctionnel* doivent être déclarées dans sa *déclaration de type* dans un bloc `SOCKETS...END_SOCKETS`, déclarant le *nom d'instance* et le *type d'interface d'adaptateur* de chaque prise femelle. Dans la représentation graphique des *types bloc fonctionnel* et des *instances*, les prises femelles sont montrées comme *variables d'entrée* avec une indication textuelle ou graphique spécialisée pour signifier qu'elles ne sont pas des variables d'entrée ordinaires.

c) Les *entrées* et les *sorties* d'une *prise mâle* doivent être utilisées au sein de sa *déclaration de type bloc fonctionnel* de la même manière que les entrées et les sorties du bloc fonctionnel.

d) Les *entrées* et les *sorties* d'une *prise femelle* doivent respectivement être utilisées au sein de sa *déclaration de type bloc fonctionnel* de la même manière que les *sorties* et les *entrées* du bloc fonctionnel.

e) L'insertion de *prises mâles* dans des *prises femelles* doit être spécifiée dans un bloc `ADAPTER_CONNECTIONS ... END_CONNECTIONS` dans la *déclaration* de *l'application, de la sous-application, du type de ressource, de l'instance de ressource* ou du *type bloc fonctionnel composé* contenant les instances respectives de *fournisseur* et d'*utilisateur*.

f) Dans le corps d'un *type bloc fonctionnel composé* ou d'une *sous-application*, une *prise femelle* est représentée comme un *bloc fonctionnel* avec les mêmes entrées et sorties que le *type adaptateur d'interface* correspondant. De manière similaire, dans ce cas une *prise mâle* est représentée comme un bloc fonctionnel avec les entrées et les sorties du type d'interface d'adaptateur correspondant inversées.

g) L'insertion des prises mâles dans des prises femelles doit être soumise aux contraintes suivantes:



– 162 – 61499-1 © CEI:2012

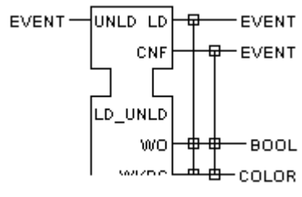



**图17a–界面**

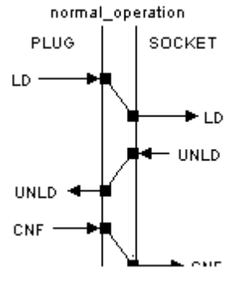



**图17b 服务顺序**

注1此类适配器的完整文本声明在附录F中给出。

注2这个例子只是说明性的。规格细节不是规定性的。

注3见6.1.2对服务序列的解释。

**图17 适配器类型声明 图形示例**

接口适配器类型和实例的使用必须符合以下规则：

a)在功能块类型实例中用作插头的接口适配器实例必须在PLUGS...END_PLUGS块中的类型声明中声明，声明实例名称和每个公插头的类型接口适配器。在功能块类型和实例的图形表示中，插头显示为输出变量，带有专门的文本或图形指示，表示它们不是普通的输出变量。

b)在功能块类型的实例中用作套接字的接口适配器实例必须在其类型声明中的SOCKETS...END_SOCKETS块中声明，声明实例名称和功能块类型。每个套接字的适配器接口。在功能块类型和实例的图形表示中，套接字显示为输入变量，带有专门的文本或图形指示，表示它们不是普通的输入变量。

c)插头的输入和输出必须在其功能块类型声明中以与功能块的输入和输出相同的方式使用。

d)套接字的输入和输出必须分别在其功能块类型声明中以与功能块的输出和输入相同的方式使用。

e)插入插座的插头必须在应用程序、子应用程序、资源类型、资源实例或包含相应提供者和用户实例的类型复合功能块的声明中的ADAPTER_CONNECTIONS...END_CONNECTIONS块中指定。

f) 在复合功能块类型或子应用程序的主体中，套接字表示为具有与相应接口适配器类型相同的输入和输出的功能块。类似地，在这种情况下，插头显示为构建块，相应适配器接口类型的输入和输出被反转。

g)将公插头插入母插头必须受到以下限制：

1) une prise mâle peut seulement être insérée dans une prise femelle du même *type adaptateur d'interface*;
2) une prise mâle peut seulement être insérée dans zéro ou une seule prise femelle à la fois;
3) une prise femelle peut seulement accepter zéro ou une seule prise mâle à la fois;
4) une prise mâle peut seulement être insérée dans une fiche femelle si toutes les deux sont dans le même *bloc fonctionnel composé*, dans la même *ressource*, dans la même *application* ou dans la même *sous-application*.

Une connexion allant d'une prise mâle à une prise femelle peut être montrée dans une *application* ou *sous-application* même si les instances de bloc fonctionnel correspondantes peuvent être *mises en correspondance* avec des *ressources* distinctes. Dans ce cas, des moyens appropriés, tels que des blocs fonctionnels interfaces de services de communication décrits en 6.2, doivent être utilisés pour mettre en œuvre le transfert correspondant d'événements et de données entre les ressources.

Les *blocs fonctionnels de gestion* tels que décrits en 6.3 peuvent fournir des moyens pour la création, la suppression et l'interrogation dynamiques des connexions d'adaptateur.

EXEMPLE 1 Une instance du type XBAR_MVCA illustré à la Figure 18 agit tant comme fournisseur d'une interface de prise mâle (LDU_PLG) que comme un utilisateur avec une interface de prise femelle (LDU_SKT). Ce faisant, elle sert à abstraire et encapsuler les interactions d'une instance du type XBAR_MVC avec des unités fonctionnelles "amont" et "aval".

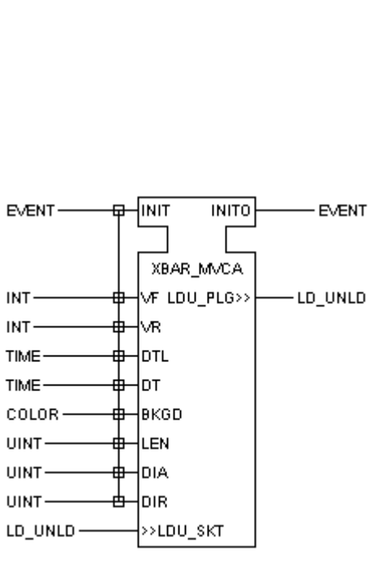



**Figure 18a – Interface**

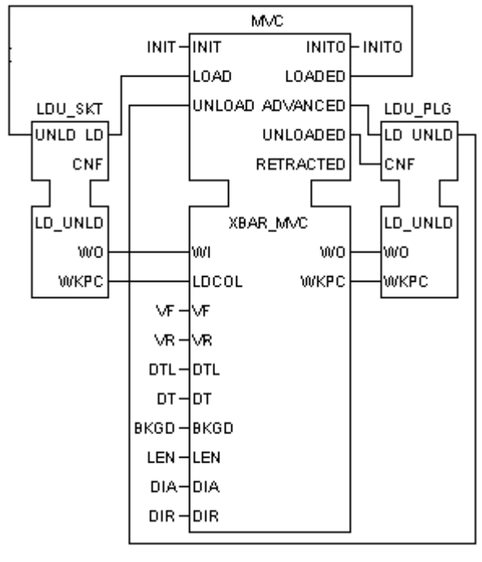



**Figure 18b – Corps**

NOTE 1 Une déclaration textuelle complète de cet exemple est donnée dans l'Annexe F.

NOTE 2 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

NOTE 3 Bien que cet exemple présente seulement un type composé, les *types* de bloc fonctionnel *fournisseur* et *utilisateur* sont susceptibles d'être soit *de base*, soit *composés*.

**Figure 18 – Illustration des déclarations des types de bloc fonctionnel fournisseur et utilisateur**





1)公插头只能插入相同接口适配器类型的母插座；

2）一个公插头一次只能插入零个或一个母插座；

3)一个母座一次只能接受零个或一个公座；

4)只有在同一个复合构建块、同一个资源、同一个应用程序或同一个子应用程序中，插头才能插入插座。

即使相应的构建块实例可以映射到单独的资源，从插头到插座的连接也可以在应用程序或子应用程序中显示。在这种情况下，应采用适当的手段，如6.2中描述的通信服务接口功能块，来实现资源之间相应的事件和数据传输。

6.3中描述的管理功能块可以为适配器连接的动态创建、删除和查询提供手段。

示例1图18中所示的XBAR_MVCA类型的实例既充当插头接口(LDU_PLG)的提供者，又充当具有套接字接口(LDU_SKT)的用户。在这样做时，它用于抽象和封装XBAR_MVC类型的实例与“上游”和“下游”功能单元的交互。

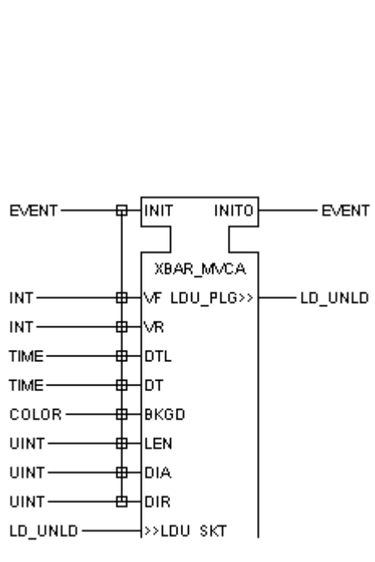



**图18a–界面**

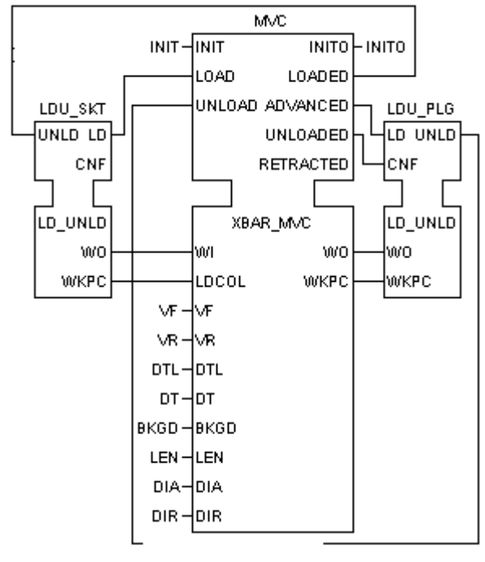



**图18b–身体**

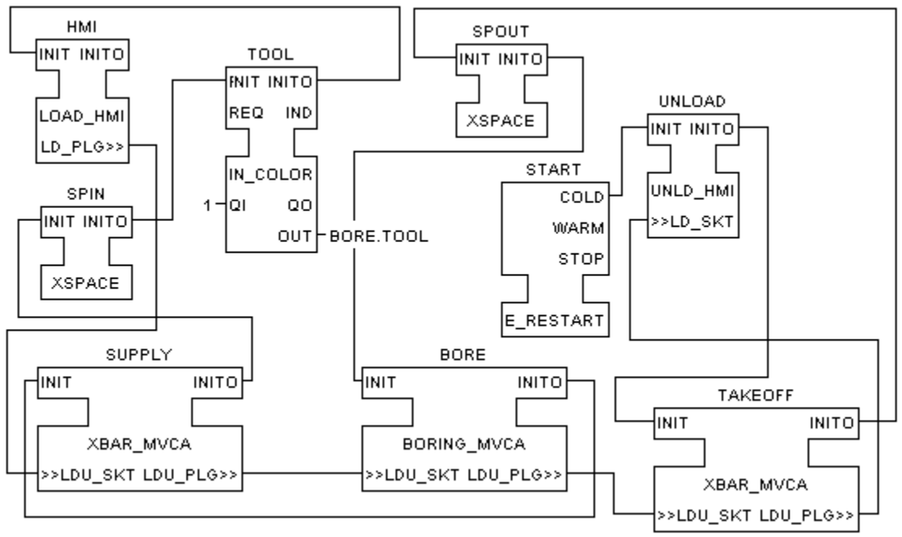
注1：附录F中给出了该示例的全文说明。

注2这个例子只是说明性的。规格细节不是规定性的。

注3虽然这个例子只显示了一种复合类型，但提供者和用户构建块类型可能是基本的或复合的。

**图18 提供者和用户功能块类型的声明说明**

EXEMPLE 2
La Figure 19 illustre une configuration de ressource contenant deux instances du type XBAR_MVCA illustré à la Figure 18. L'instance SUPPLY agit comme un utilisateur ("unité aval") pour le bloc HMI et comme un fournisseur ("unité amont") pour le bloc BORE, alors que l'instance TAKEOFF remplit respectivement des rôles correspondants pour les blocs BORE et UNLOAD.




NOTE 1 Cet exemple est uniquement illustratif. Les détails de la spécification ne sont pas normatifs.

NOTE 2 Par souci de clarté, les connexions de *paramètre* sont omises dans ce diagramme.

NOTE 3 Les déclarations de types pour blocs autres que le type XBAR_MVCA ne sont pas données dans l'Annexe F.

**Figure 19 – Illustration des connexions d'adaptateur**

### 5.6 Traitement des exceptions et des défauts

Des moyens supplémentaires pour la prévention, la reconnaissance et le traitement des *exceptions* et des *défauts* peuvent être fournis par des *ressources.* De telles capacités peuvent être modélisées sous la forme de *blocs fonctionnels interface de service*. La définition de types spécifiques de bloc fonctionnel pour la prévention, la reconnaissance et le traitement des exceptions et des défauts ne relève pas du domaine d'application de la présente norme. Cependant, les sorties `INIT, CNF` et `IND` des blocs fonctionnels interface de service, et les valeurs associées de `STATUS`, peuvent être utilisées pour indiquer l'occurrence et le type des exceptions et des défauts, comme noté en 6.1.3.

## 6 Blocs fonctionnels interface de service

### 6.1 Principes généraux

#### 6.1.1 Généralités

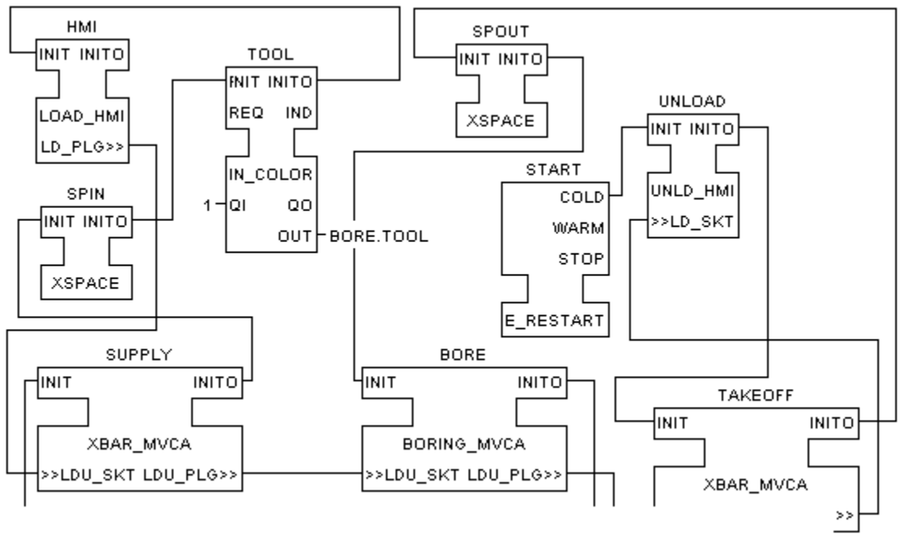
Un *bloc fonctionnel interface de service* fournit un ou plusieurs *services* à une application, en se basant sur un *mapping* de *primitives de service* avec les *entrées d'événements, sorties d'événements, entrées de données* et *sorties de données* du bloc fonctionnel.

Les interfaces externes des *types de bloc fonctionnel interface de service* ont la même apparence générale que les *types de bloc fonctionnel de base*. Cependant, certaines entrées





图19说明了包含两个XBAR_MVCA类型实例的资源配置，如图18所示。SUPPLY实例充当HMI块的用户（“下游单元”）和BORE块的供应商（“上游单元”），而TAKEOFF实例分别履行BORE和UNLOAD块的相应角色。




注1这个例子只是说明性的。规格细节不是规定性的。

注2为清楚起见，此图中省略了参数连接。

注3附录F中没有给出XBAR_MVCA类型以外的块的类型声明。

**图19–适配器连接图示**

**处理异常和故障**

资源可以提供用于防止、识别和处理异常和故障的附加手段。这种能力可以建模为服务接口功能块。用于预防、识别和处理异常和故障的特定类型功能块的定义超出了本标准的范围。但是，服务接口功能块的INIT、CNF和IND输出以及相关的STATUS值可用于指示异常和故障的发生和类型，如6.1.3中所述。

## 6 服务接口功能块

### 6.1 Principes généraux

服务接口功能块基于服务原语到功能块的事件输入、事件输出、数据输入和数据输出的映射向应用程序提供一个或多个服务。

服务接口功能块类型的外部接口与基本功能块类型具有相同的外观。然而，一些条目

et sorties de type bloc fonctionnel interface de service ont une sémantique spécialisée et le comportement d'*instances* de ces types est défini par une notation graphique spécialisée pour les séquences de *primitives de service.*

NOTE La spécification des opérations internes des blocs fonctionnels interface de service ne relève pas du domaine d'application de la présente norme.

### 6.1.2 Spécification de type

La *déclaration* des *types bloc fonctionnel interface de service* peut utiliser les *entrées d'événements, sorties d'événements, entrées de données* et *sorties de données* normalisées qui sont énumérées dans le Tableau 2, selon ce qui est approprié au service particulier fourni. Lorsque celles-ci sont utilisées, leur sémantique doit être telle que définie en 6.1.2. Le nom du *type* de bloc fonctionnel doit indiquer le service fourni.

EXEMPLE Les Figures 20(a) et (b) montrent des exemples de blocs fonctionnels interface de service dans lesquels l'interaction principale est respectivement déclenchée par l'application et par la ressource.

NOTE 1 Les services peuvent fournir des interactions déclenchées tant par la ressource que par l'application dans le même bloc fonctionnel interface de service.

NOTE 2 Les types interface de service peuvent également utiliser des entrées et des sorties, y compris les *prises mâles* et les *prises femelles*, avec des noms différents de ceux donnés ici; dans un tel cas, leur usage doit être défini en termes de séquences appropriées de primitives de service.

**Tableau 2 – Entrées et sorties normalisées pour les blocs fonctionnels interface de service** (1 de 2)

| **Entrées d'événements** |
|---|
| **INIT**<br>Cette entrée d'événements doit être *mappée avec* une *primitive* "*request*" qui demande une initialisation du service fourni par l'instance de bloc fonctionnel, par exemple, l'initialisation locale d'une *connexion de la communication* ou un module d'interface de processus. |
| **REQ**<br>Cette entrée d'événements doit être mappée avec une *primitive* "*request*" du service fourni par l'instance bloc fonctionnel. |
| **RSP**<br>Cette entrée d'événements doit être mappée avec une *primitive* "*response*" du service fourni par l'instance bloc fonctionnel. |
| **Sorties d'événements** |
| **INITO**<br>Cette sortie d'événements doit être mappée avec une *primitive* "*confirm*" qui indique la fin d'une procédure d'initialisation du service. |
| **CNF**<br>Cette sortie d'événements doit être mappée avec une *primitive* "*confirm*" du service fourni par l'instance bloc fonctionnel. |
| **IND**<br>Cette sortie d'événements doit être mappée avec une *primitive* "*indication*" du service fourni par l'instance bloc fonctionnel. |



和服务接口功能块类型输出具有专门的语义，并且这些类型的实例的行为由服务原语序列的专门图形符号定义。

注：服务接口功能块的内部操作规范超出了本标准的范围。

**型号规格**

服务接口功能块类型的声明可以使用表2中列出的标准事件输入、事件输出、数据输入和数据输出，以适合所提供的特定服务。当使用这些时，它们的语义应如6.1.2中所定义。功能块类型的名称必须表明所提供的服务。

示例图20(a)和(b)显示了服务接口功能块的示例，其中主要交互分别由应用程序和资源触发。

注1服务可以在同一个服务接口功能块中提供资源发起的和应用程序发起的交互。

注2服务接口类型也可以使用输入和输出，包括插头和插座，其名称与此处给出的名称不同；在这种情况下，它们的使用必须根据适当 的服务原语序列来定义。

表2–服务接口功能块的标准输入和输出（2个中的1个）

| **Entrées d'événements** |
|---|
| 该事件输入应映射到"请求"原语，该原语请求对功能块实例提供的服务进行初始化，例如，通信连接或通信接口模块的本地初始化过程。 |
| 该事件输入必须映射到功能块实例提供的服务的"请求"原语。 |
| 该事件输入必须映射到功能块实例提供的服务的"响应"原语。 |
| **Sorties d'événements** |
| 此事件输出应映射到指示服务初始化过程结束的"确认"原语。 |
| 该事件输出必须映射到功能块实例提供的服务的"确认"原语。 |
| 该事件输出必须映射到功能块实例提供的服务的"指示"原语。 |

**Tableau 2** (2 de 2)

| Entrées de données |
|---|
| **QI**: **BOOL**<br><br>Cette entrée représente un qualificateur sur les *primitives de service* mises en correspondance avec les *entrées d'événements*. Par exemple, si cette entrée est TRUE à l'apparition d'un événement INIT, l'initialisation du service est demandée; si elle est FALSE, l'arrêt du service est demandé. |
| **PARAMS**: **ANY**<br><br>Cette entrée contient un ou plusieurs *paramètres* associés au service, typiquement comme éléments d'une *instance* d'un *type de données structuré*. Lorsque cette entrée est présente, la spécification du *type* de *bloc fonctionnel* doit définir son *type de données* et sa/ses valeur(s) initiale(s) par défaut.<br><br>Une spécification du type bloc fonctionnel d'interface service peut remplacer cette entrée par une ou plusieurs entrées de paramètres de service. |
| **SD_1, ..., SD_m: ANY**<br><br>Ces entrées contiennent les données associées aux *primitives* "*request*" et "*response*". La spécification du *type* de *bloc fonctionnel* doit définir les *types de données* et les valeurs par défaut de ces entrées et doit définir leurs associations avec des entrées d'événements dans un diagramme de séquences d'événement tel que déclaré en 6.1.3.<br><br>La spécification du type de bloc fonctionnel peut définir d'autres noms pour ces entrées. |
| **Sorties de données** |
| **QO: BOOL**<br><br>Cette variable représente un qualificateur sur les *primitives de service* mises en correspondance avec les *sorties d'événements*. Par exemple, une valeur TRUE de cette sortie à l'apparition d'un événement INITO indique une initialisation réussie du service; une valeur FALSE indique une initialisation infructueuse. |
| **STATUS: ANY**<br><br>Cette sortie doit être d'un *type de données* approprié pour exprimer le statut du service à l'apparition d'une sortie d'événements.<br><br>Une spécification de service peut indiquer que la valeur de cette sortie n'est pas pertinente pour certaines situations, par exemple, pour `INITO+, IND+` et `CNF+` comme décrit en 6.1.3. |
| **RD_1, ..., RD_n: ANY**<br><br>Ces sorties contiennent les données associées aux primitives "*confirm*" et "*indication*". La *spécification du type* du bloc fonctionnel doit définir les *types de données* et les valeurs initiales de ces sorties et doit définir leurs associations avec des sorties d'événements dans un diagramme de séquences d'événement tel que décrit en 6.1.3.<br><br>La spécification du type de bloc fonctionnel peut définir d'autres noms pour ces sorties. |





表2（2个中的2个）

| Entrées de données |
|---|
| 此条目表示映射到事件条目的服务原语的限定符。例如，如果在INIT事件发生时该条目为TRUE，则请求服务初始化；如果为FALSE，则请求停止服务。 |
| 此条目包含一个或多个与服务关联的参数，通常作为结构化数据类型实例的一部分。当该条目出现时，功能块类型规范应定义其默认数据类型和初始值。<br><br>服务接口功能块类型的规范可以用一个或多个服务参数条目替换该条目。 |
| 这些条目包含与“请求”和“响应”原语相关的数据。功能块类型规范应定义这些输入的数据类型和默认值，并应在6.1.3中声明的事件序列图中定义它们与事件输入的关联。<br><br>功能块类型规范可以为这些输入定义其他名称。 |
| **数据输出** |
| 此变量表示映射到事件输出的服务原语的限定符。例如，在发生INITO事件时此输出的TRUE值表示服务初始化成功；FALSE值表示初始化不成功。 |
| 此输出必须是适当的数据类型，以在发生事件输出时表达服务的状态。<br><br>服务规范可能表明此输出的值与某些情况无关，例如，对于6.1.3中所述的INITO+、IND+和CNF+。 |
| 这些输出包含与“确认”和“指示”原语相关的数据。功能块类型规范应定义这些输出的数据类型和初始值，并应在事件序列图中定义它们与事件输出的关联，如6.1.3所述。<br><br>功能块类型规范可以为这些输出定义其他名称。 |

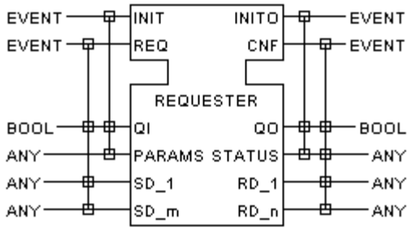



**Figure 20a – Interactions déclenchées par une application**

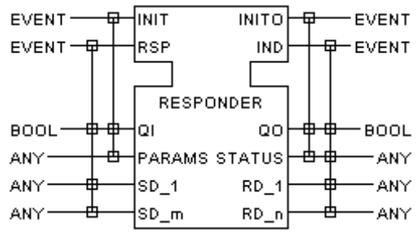



**Figure 20b – Interactions déclenchées par une ressource**

NOTE 1 REQUESTER et RESPONDER représentent les services particuliers fournis par des instances des types de bloc fonctionnel.

NOTE 2 Les *types de données* des entrées SD_1, . . . , SD_n et des sorties RD_1, . . . RD_m seront typiquement fixés comme un certain type de données non générique, par exemple INT ou WORD, dans des mises en œuvre concrètes des types de bloc fonctionnel génériques illustrés ici.

NOTE 3 Voir l'Annexe F pour une déclaration textuelle complète du type de bloc fonctionnel REQUESTER.

**Figure 20 – Exemples de blocs fonctionnels interface de service**

### 6.1.3 Comportement des instances

Le comportement des *instances* des *blocs fonctionnels interface de service* doit être défini dans la spécification correspondante du *type* de *bloc fonctionnel*, qui peut utiliser des *diagrammes de séquences de service* assujettis aux règles suivantes:

a) La sémantique suivante doit s'appliquer:
   1) Le temps augmente dans le sens descendant.
   2) Les événements qui sont liés séquentiellement sont reliés ensemble au travers ou au sein des ressources.
   3) S'il n'y a pas de relation spécifique entre des événements, en ce qu'il est impossible de prévoir lequel se produira le premier, mais que tous les deux doivent se produire dans un délai fini, un tilde (~) ou une notation textuelle similaire est utilisé(e).

b) Dans le cas où le service est représenté par un seul bloc fonctionnel interface de service, le diagramme doit être partagé par une seule ligne verticale en deux champs tels qu'illustrés en Figure 21:
   1) Dans le cas où le service est fourni principalement par une interaction déclenchée par une application, l'*application* doit être dans le champ de gauche et la *ressource* dans le champ de droite (voir Figure 21 a)).
   2) Dans le cas où le service est fourni principalement par une interaction déclenchée par une ressource, la *ressource* doit être dans le champ de gauche et l'*application* dans le champ de droite (voir Figure 21 b)).

c) Dans le cas où le service est représenté par deux ou plusieurs blocs fonctionnels interface de service, la notation illustrée en E.2.2 et en E.2.3 peut être utilisée.

d) Les *primitives de service* doivent être indiquées par des flèches horizontales. Le nom de l'*événement* représentant la primitive de service doit être écrit au voisinage de la flèche et des moyens doivent être fournis pour déterminer les noms des *variables* d'entrée et/ou de sortie représentant les *données* associées à la primitive.

e) Lorsqu'une entrée QI est présente dans la définition du type du bloc fonctionnel, le suffixe "+" doit être utilisé conjointement à un nom d'*entrée d'événements* pour indiquer que la





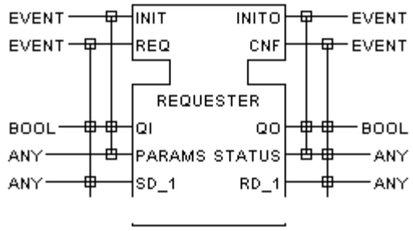



**图20a 由a触发的交互**

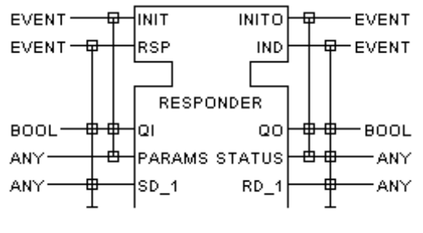



**图20b 由a触发的交互**

注1REQUEST和RESPONDER代表功能块类型实例提供的特定服务。

注2输入SD_1 … SD_n和输出RD_1 …RD_m的数据类型通常会被固定为一些非通用数据类型，例如INT或WORD，在所示的通用功能块类型的具体实现中这里。

注3：REQUESTER功能块类型的完整文本声明见附录F。

**图20–服务接口功能块示例**

**实例行为**

服务接口功能块实例的行为应在相关功能块类型规范中定义，可以使用服务序列图，但须遵守以下规则：

a)应适用以下语义：

1)时间递减。

2)顺序链接的事件跨资源或在资源内链接在一起。

3)如果事件之间没有特定的关系，即无法预测哪个会先发生，但两者都必须在有限时间内发生，则使用波浪号(~)或类似的文本符号。

b)在服务由单个服务接口功能块表示的情况下，该图应由一条垂直线分成两个字段，如图21所示：

1)在服务主要由应用程序触发的交互提供的情况下，应用程序应位于左侧字段中，资源应位于右侧字段中（参见图21a））。

2)在服务主要由资源触发的交互提供的情况下，资源应位于左侧字段中，应用程序位于右侧字段中（参见图21b））。

c)在服务由两个或多个服务接口功能块表示的情况下，可以使用E.2.2和E.2.3中说明的符号。

d)服务原语应该用水平箭头表示。表示服务原语的事件名称必须写在箭头附近，并且必须提供方法来确定表示与原语相关的数据的输入和/或输出变量的名称。

e)当功能块类型定义中出现QI条目时，后缀“+”应与事件条目名称一起使用，以指示

valeur de l'entrée QI est TRUE à l'apparition de l'événement associé, et le suffixe "-" doit être utilisé pour indiquer qu'elle est FALSE.

f) Lorsqu'une sortie QO est présente dans la définition du type du bloc fonctionnel, le suffixe "+" doit être utilisé conjointement à un nom de *sortie d'événements* pour indiquer que la valeur de la sortie QO est TRUE à l'apparition de l'événement associé, et le suffixe "-" doit être utilisé pour indiquer qu'elle est FALSE.

g) La sémantique normalisée des événements affirmés (+) et niés (-) doit être telle que spécifiée dans le Tableau 3.

La Figure 21 illustre des séquences normales de lancement du service, de transfert de données et d'arrêt du service. Des spécifications du *type de bloc fonctionnel interface de service* peuvent utiliser des diagrammes similaires pour spécifier toutes les séquences pertinentes des primitives de service et leurs données associées dans des conditions normales et anormales.

NOTE Des diagrammes de séquences sont également susceptibles d'être utilisés pour documenter les comportements observables de l'extérieur des types du bloc fonctionnel de base et composé.

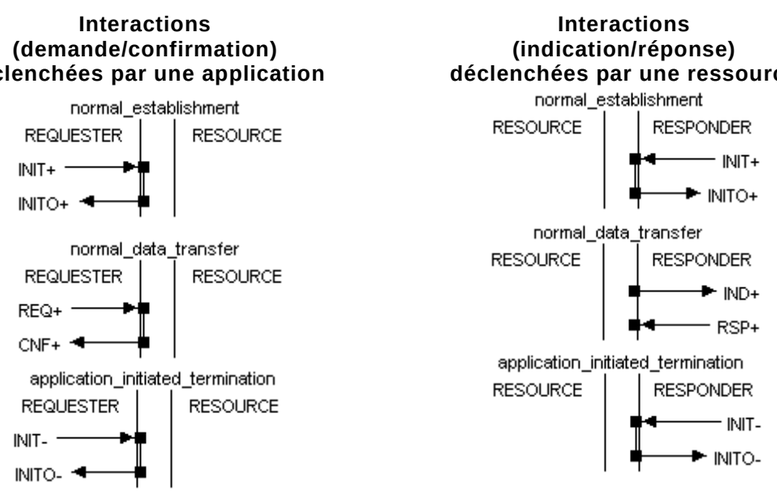



**Figure 21 – Exemples de diagrammes des séquences de service**



当关联事件发生时，QI入口的值为TRUE，必须使用后缀“–”表示为FALSE。

f) 当功能块类型定义中存在QO输出时，后缀“+”必须与事件输出名称结合使用，以表示事件发生时QO输出的值为TRUE。关联事件，以及后缀必须使用“–”来表示它是FALSE。

g)断言(+)和拒绝(–)事件的标准化语义应如表3中规定。

图21说明了正常的服务启动、数据传输和服务关闭序列。服务接口功能块类型的规范可以使用类似的图表来指定所有相关的服务原语序列及其在正常和异常情况下的相关数据。

注意序列图也可能用于记录从基本和复合功能块类型之外观察到的行为。

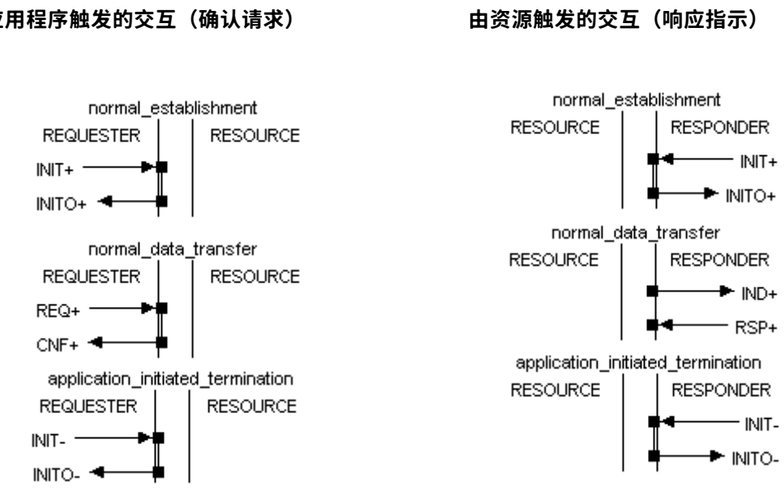



**图21 服务序列图示例**

**Tableau 3 – Sémantique des primitives de service**

| Primitive | Sémantique |
|---|---|
| **INIT+** | Demande d'établissement d'un service |
| **INIT-** | Demande d'arrêt d'un service |
| **INITO+** | Indication d'établissement d'un service normal |
| **INITO-** | Rejet de demande d'établissement d'un service ou indication d'arrêt d'un service |
| **REQ+** | Demande normale pour un service |
| **REQ-** | Demande désactivée pour un service |
| **CNF+** | Confirmation normale d'un service |
| **CNF-** | Indication d'état anormal d'un service |
| **IND+** | Indication d'arrivée normale d'un service |
| **IND-** | Indication d'état anormal d'un service |
| **RSP+** | Réponse normale par une application |
| **RSP-** | Réponse anormale par une application |

## 6.2 Blocs fonctionnels de communication

### 6.2.1 Spécification de type

Les blocs fonctionnels de communication fournissent des *interfaces* entre des *applications* et les fonctions de "mapping de la communication" des *ressources* telles que définies en 4.3; aussi sont-ils des *blocs fonctionnels interface de service* tels que décrits en 6.1.

Comme d'autres blocs fonctionnels interface de service, un bloc fonctionnel de communication peut être de type *basic* (c'est-à-dire: de base) ou *composite* (c'est-à-dire: composé), tant que son opération peut être représentée par un *mapping* de *primitives de service* avec les *entrées d'événements, les sorties d'événements, les entrées de données* et les *sorties de données* du bloc fonctionnel.

Le paragraphe 6.2.1 donne des règles pour la *déclaration* des *types bloc fonctionnel de communication.* Le paragraphe 6.2.2 donne des règles pour le comportement des *instances* de ces types de bloc fonctionnel. L'Article E.2 définit des types génériques de bloc fonctionnel de communication pour les transactions *unidirectionnelles* et *bidirectionnelles* et donne des règles de personnalisation dépendant de la mise en œuvre de ces types.

La *déclaration* des *types de bloc fonctionnel communication* doit utiliser le moyen défini en 6.1 pour la déclaration des *types de bloc fonctionnel interface de service*, avec la sémantique spécialisée montrée dans le Tableau 4 pour les variables d'*entrée* et de *sortie*.





**表3 服务原语的语义**

| Primitive | que |
|---|---|
| **INIT+** | 请求建立服务 |
| | 请求停止服务 |
| INITO+正常服务建立指示 | |
| INITOR拒绝建立服务的请求或停止服务的指示 | |
| **REQ+** | 正常的服务请求 |
| **REQ-** | 禁用服务请求 |
| **CNF+** | 服务的正常确认 |
| **CNF-** | 指示服务的异常状态 |
| **IND+** | 服务的正常到达指示 |
| **IND-** | 指示服务的异常状态 |
| **RSP+** | 应用程序的正常响应 |
| **RSP-** | 应用程序的异常响应 |

## 6.2 通讯积木

### 型号规格

通信功能块提供应用程序和4.3中定义的资源"通信映射"功能之间的接口；因此它们是6.1中描述的服务接口功能块。

与其他服务接口功能块一样，通信功能块可以是基本的（即：基本的）或复合的（即：复合的），只要其操作可以通过服务原语与事件输入、事件输出、数据输入的映射来表示和功能块的数据输出。

6.2.1节给出了声明通信功能块类型的规则。第6.2.2节给出了这些功能块类型实例的行为规则。条款E.2定义了单向和双向事务的通用通信功能块类型，并为这些类型提供了依赖于实现的定制规则。

通信功能块类型的声明应使用6.1中定义的服务接口功能块类型声明的方式，输入和输出变量的专用语义如表4所示。

**Tableau 4 – Sémantique des variables pour les blocs fonctionnels de communication**

| Variable | Sémantique |
|---|---|
| **PARAMS** | Cette entrée fournit des *paramètres* pour la *connexion de communication* associée à l'*instance de bloc fonctionnel de communication*. Elle doit inclure un moyen d'identifier le protocole de communication et la connexion de communication et peut inclure d'autres paramètres de connexion de communication tels que les contraintes de temporisation, etc. |
| **SD_1,..., SD_m** | Ces entrées représentent des *données* devant être transférées sur la *connexion de communication* spécifiée par l'entrée PARAMS à l'apparition d'une *primitive* REQ+ ou RSP+, selon ce qui est approprié. [a] |
| **STATUS** | Cette sortie représente le statut de la *connexion de communication*, par exemple:<br>- Fin normale du lancement, de l'arrêt, ou de transfert de données<br>- Causes de lancement anormal, d'arrêt anormal ou de transfert anormal de données |
| **RD_1,..., RD_n** | Ces sorties représentent des *données* reçues sur la *connexion de communication* spécifiée par l'entrée PARAMS à l'apparition d'une primitive IND+ ou CNF+ selon ce qui est approprié. [a] |
| NOTE Les déclarations de type de bloc fonctionnel de communication peuvent définir des contraintes entre les sorties RD_1,...,RD_n et les entrées SD_1,..., SD_m d'instances correspondantes de bloc fonctionnel. Par exemple, le nombre et les types des sorties RD pourraient'être contraints pour concorder avec le nombre et les types des entrées SD correspondantes. | |
| [a] *Les déclarations de type de bloc fonctionnel de communication* définissent le nombre et le type des entrées SD_1,..., SD_m et des sorties RD_1,...,RD_n et peuvent leur attribuer d'autres noms. | |

### 6.2.2 Comportement des instances

Comme illustré à l'Article E.2, le comportement des *instances* des *types de bloc fonctionnel de communication* doit être défini dans la *déclaration* correspondante de type bloc fonctionnel de communication, en utilisant le moyen spécifié pour les *blocs fonctionnels interface de service* en 6.1 avec la sémantique spécialisée des primitives de service donnée dans le Tableau 5. Une telle spécification doit inclure des séquences de *primitives de service* pour:

- l'établissement et la libération normaux et anormaux des *connexions de communication;*
- le transfert normal et anormal de données.

**Tableau 5 – Sémantique des primitives de service pour les blocs fonctionnels de communication**

| Primitive | Sémantique |
|---|---|
| **INIT+** | Demande pour établissement d'une connexion de communication |
| **INIT-** | Demande pour libération d'une connexion de communication |
| **INITO+** | Indication d'établissement d'une connexion de communication |
| **INITO-** | Rejet d'une demande d'établissement de connexion de communication ou indication d'une libération d'une connexion de communication |
| **REQ+** | Demande normale pour transfert de données |
| **REQ-** | Demande désactivée pour transfert de données |
| **CNF+** | Confirmation normale de transfert de données |
| **CNF-** | Indication de transfert anormal de données |
| **IND+** | Indication d'arrivée normale de données |
| **IND-** | Indication d'arrivée anormale de données |
| **RSP+** | Réponse normale par l'application à l'arrivée de données |
| **RSP-** | Réponse anormale par l'application à l'arrivée de données |





**表4–通信功能块变量的语义**

| Variable | |
|---|---|
| **PARAMS** | 此条目提供与通信功能块实例关联的通信连接的参数。它必须包括一种识别通信协议和通信连接的方法，并且可能包括其他通信连接参数，例如超时约束等。 |
| **SD_1,..., SD_m** | 这些条目表示在适当的REQ+或RSP+原语出现时通过由PARAMS条目指定的通信连接传输的数据。有 |
| **STATUS** | 此输出表示通信连接的状态，例如：正常启动、停止或数据传输原因异常启动、异常停止或数据传输异常原因 |
| **RD_1,..., RD_n** | 这些输出表示在适当的IND+或CNF+原语出现时，在由PARAMS条目指定的通信连接上接收到的数据。有 |
| 注意通信功能块类型声明可以定义相应功能块实例的RD_1 … RD_n输出和SD_1 … SD_m输入之间的约束。例如，可以限制RD输出的数量和类型以匹配相应SD输入的数量和类型。 | |
| a通信功能块类型声明定义输入SD_1、…、SD_m和输出RD_1、…、RD_n的数量和类型，并且可以给它们其他名称。 | |

**实例行为**

如第E.2节所示，通信功能块类型实例的行为应在相应的通信功能块类型声明中定义，使用6.1中为服务接口功能块指定的方式以及表中给出的服务原语的专用语义5.这样的规范应包括服务原语序列：

- 正常和异常建立和释放通信连接；
- 正常和异常数据传输。

**表5 通信功能块的服务原语的语义**

| Primitive | |
|---|---|
| | 请求建立通信连接 |
| | 请求释放通信连接 |
| INITO+指示 | 建立通信连接 |
| INITO | Rejection通信连接建立请求或通信连接释放的指示 |
| **REQ+** | 数据传输的正常请求 |
| **REQ-** | 数据传输请求被禁用 |
| **CNF+** | 正常数据传输确认 |
| **CNF-** | 异常数据传输的指示 |
| **IND+** | 正常数据到达指示 |
| **IND-** | 数据到达指示异常 |
| **RSP+** | 应用程序对数据到达的正常响应 |
| **RSP-** | 应用程序对数据到达的异常响应 |

## 6.3 Blocs fonctionnels de gestion

### 6.3.1 Exigences

L'extension des exigences fonctionnelles pour la "gestion d'application" en 8.3.2 de l'ISO/CEI 7498-1:1994 au modèle d'application distribuée de la présente norme indique qu'il convient que les *services* pour la gestion des ressources et d'applications dans les systèmes IPMCS puissent accomplir les *fonctions* suivantes*:*

a) Dans une *ressource,* créer, initialiser, démarrer, arrêter, supprimer, vérifier l'existence et les *attributs* des changements de disponibilité et de statut des éléments ci-après et en donner notification:
   1) types de données
   2) types et instances de bloc fonctionnel
   3) connexions entre des instances de bloc fonctionnel
b) Dans un *équipement,* créer, initialiser, démarrer, arrêter, supprimer, vérifier l'existence et les *attributs* des changements de disponibilité et de statut des *ressources* et en donner notification.

NOTE 1 Les dispositions de la présente norme ne visent pas à satisfaire aux exigences pour la *gestion de système* traitées dans l'ISO/CEI 7498-4 et l'ISO/CEI 10040, avec l'exception que de telles exigences sont traitées par les fonctions énumérées ci-dessus.

NOTE 2 La présente norme traite seulement de l'élément a) ci-dessus, c'est-à-dire la gestion des ressources des *applications*. Un cadre de travail pour la gestion des équipements est décrit dans la CEI 61499-2.

NOTE 3 Les associations entre les *ressources*, les *applications* et les *instances de bloc fonctionnel* sont définies dans les *configurations de système* telles que décrites en 7.3.

NOTE 4 Le démarrage et l'arrêt d'une *application* distribuée sont accomplis par un *outil logiciel* approprié.

### 6.3.2 Spécification de type

La Figure 22 illustre la forme générale des *types de bloc fonctionnel de gestion* dont les *instances* satisfont aux exigences de gestion d'application définies ci-dessus.

NOTE 1 Dans des mises en œuvre particulières, le nom de type (`MANAGER` dans cet exemple) pourrait représenter le type de la ressource gérée.

NOTE 2 Pour ces types de bloc fonctionnel, les entrées spécifiques `CMD` et `OBJECT` et la sortie spécifique `RESULT` remplacent les entrées génériques `SD_1` et `SD_2` et la sortie générique `RD_1` décrites en 6.1.

NOTE 3 Les entrées `INIT` et `PARAMS` et la sortie `INITO` pourraientêtre présentes ou non dans une mise en œuvre particulière.

NOTE 4 Lorsqu'elles sont présentes, le type et les valeurs de l'entrée `PARAMS` sont des paramètres **dépendants de la mise en œuvre** du type de ressource.

NOTE 5 Une spécification textuelle complète de ce type de bloc fonctionnel, y compris toutes les séquences de service, est donnée dans l'Annexe F.



61499-1 © CEI:2012 – 171 –

## 6.3 Blocs fonctionnels de gestion

将ISOIEC7498-1:1994的8.3.2中"应用程序管理"的功能要求扩展到本标准的分布式应用程序模型，表明IPMCS系统中用于管理资源和应用程序的服务可以执行以下功能：

a)在一个资源中，创建、初始化、启动、停止、删除、检查以下元素的可用性和状态变化的存在和属性并给出通知：

1)数据类型
2)功能块类型和实例
3)功能块实例之间的连接

b)在设备中，创建、初始化、启动、停止、删除、验证资源可用性和状态变化的存在和属性并给出通知。

注1：本标准的规定不旨在满足ISOIEC7498-4和ISOIEC10040中提出的系统管理要求，除非这些要求由上面列出的功能解决。

注2：本标准仅涉及上述a)项，即应用资源管理。IEC61499-2中描述了管理设备的框架。

注3资源、应用程序和功能块实例之间的关联在系统配置中定义，如7.3所述。

注4：启动和停止分布式应用程序是由适当的软件工具完成的。

**型号规格**

图22说明了管理功能块类型的一般形式，其实例满足上面定义的应用程序管理要求。

注1在特定实现中，类型名称（本例中为MANAGER）可以表示受管资源的类型。

注2对于这些类型的功能块，特定的CMD和OBJECT输入以及特定的输出RESULT替换6.1中描述的通用输入SD_1和SD_2以及通用输出RD_1。

注3：INIT和PARAMS输入以及INITO输出在特定实现中可能存在也可能不存在。

注4当存在时，PARAMS条目的类型和值是资源类型的实现相关参数。

注5此类功能块的完整文本规范，包括所有服务序列，在附件F中给出。

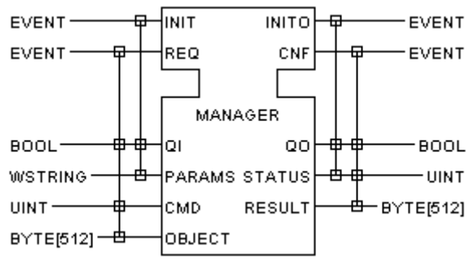
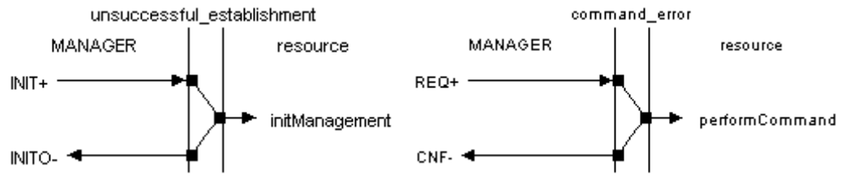



**Figure 22 – Type générique d'un bloc fonctionnel de gestion**

Le comportement des instances et la sémantique d'entrée/sortie des types de bloc fonctionnel de gestion doivent suivre les règles données en 6.1 pour les *types de bloc fonctionnel interface de service* avec des interactions lancées par une application, avec les comportements complémentaires montrés à la Figure 23 pour un lancement et des demandes de service infructueuses.




NOTE Une spécification textuelle complète de ce type de bloc fonctionnel, y compris toutes les séquences de service, est donnée dans l'Annexe F.

**Figure 23 – Séquences des primitives de service pour un service infructueux**

L'*opération* de gestion devant être *exécutée* doit être exprimée par la valeur de l'entrée CMD d'un bloc fonctionnel de gestion conformément à la sémantique définie dans le Tableau 6.

**Tableau 6 – Valeurs et sémantique de l'entrée CMD**

| Valeur | Commande | Sémantique |
|---|---|---|
| **0** | CREATE | Créer un objet spécifié |
| **1** | DELETE | Supprimer un objet spécifié |
| **2** | START | Démarrer un objet spécifié |
| **3** | STOP | Arrêter un objet spécifié |
| **4** | READ | Lire des données paramètre |
| **5** | WRITE | Écrire des données paramètre |
| **6** | KILL | Rendre inexécutable un objet spécifié |
| **7** | QUERY | Demander des informations relatives à un objet spécifié |
| **8** | RESET | Réinitialiser un objet spécifié |

Les valeurs et la sémantique correspondante de la sortie STATUS d'un bloc fonctionnel de gestion doivent être telles que décrites dans le Tableau 7 pour exprimer le résultat de l'exécution de la commande spécifiée.





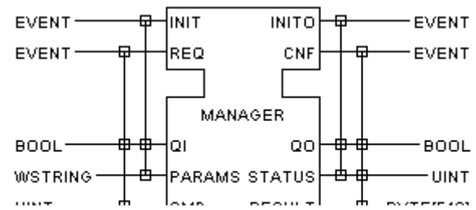
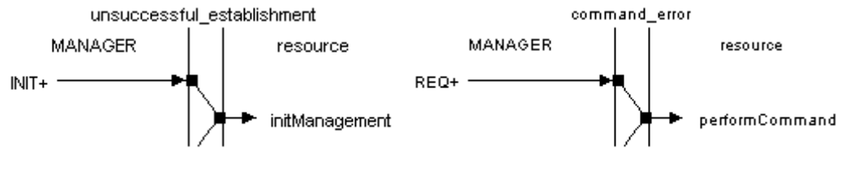



**图22 管理功能块的通用类型**

管理功能块类型的实例行为和输入–输出语义应遵循6.1中为具有应用程序启动交互的服务接口功能块类型给出的规则，对于不成功的启动和服务请求，附加行为如图23所示。




注：此类功能块的完整文本规范，包括所有服务序列，在附录F中给出。

**图23 不成功服务的服务原语序列**

根据表6中定义的语义，要执行的管理操作应由管理功能块的CMD输入值表示。

**表6 CMD入口值和语义**

| Valeur | Commande | ntique |
|---|---|---|
| **0** | CREATE | 创建指定对象 |
| **1** | DELETE | 删除指定对象 écifié |
| **2** | START | 启动指定对象 ifié |
| **3** | STOP | 停止指定对象 fié |
| **4** | READ | 读取参数数据 |
| **5** | WRITE | |
| **6** | KILL | 使指定对象不可执行 |
| **7** | QUERY | 请求有关指定对象的信息 ›jet spécifié |
| | | 重置指定对象 |

管理功能块的STATUS输出的值和对应的语义如表7所述，以表达指定命令的执行结果。

**Tableau 7 – Valeurs et sémantique de la sortie STATUS**

| Valeur | Statut | Sémantique |
|---|---|---|
| **0** | RDY | Absence d'erreurs |
| **1** | BAD_PARAMS | Valeur non valide de l'entrée PARAMS |
| **2** | LOCAL_TERMINATION | Arrêt déclenché par l'application |
| **3** | SYSTEM_TERMINATION | Arrêt déclenché par le système |
| **4** | NOT_READY | Le gestionnaire n'est pas capable de traiter la commande. |
| **5** | UNSUPPORTED_CMD | La commande demandée n'est pas prise en charge. |
| **6** | UNSUPPORTED_TYPE | Le type d'objet demandé n'est pas pris en charge. |
| **7** | NO_SUCH_OBJECT | L'objet référencé n'existe pas. |
| **8** | INVALID_OBJECT | Syntaxe non valide pour la spécification d'objet |
| **9** | INVALID_OPERATION | L'opération commandée n'est pas valide pour l'objet spécifié. |
| **10** | INVALID_STATE | L'opération commandée n'est pas valide pour l'état actuel de l'objet. |
| **11** | OVERFLOW | Transaction précédente toujours en cours |

Les longueurs effectives de l'entrée OBJECT et de la sortie RESULT des instances de bloc fonctionnel de gestion sont **dépendantes de la mise en œuvre**.

L'entrée OBJECT doit spécifier l'objet devant subir l'opération conformément à l'entrée CMD et la sortie RESULT doit contenir une description de l'objet résultant de l'opération si elle a réussi. Le contenu de ces chaînes doit inclure des codages **dépendants de la mise en œuvre** des objets définis comme des symboles non terminaux dans l'Annexe B et référencés dans le Tableau 8.

NOTE 6 La longueur maximale admissible de l'entrée OBJECT et de la sortie RESULT est un **paramètre dépendant de la mise en œuvre**; la valeur de 512 donnée dans la Figure 22 est illustrative.

**Tableau 8 – Syntaxe de commande**

| CMD | OBJECT | RESULT |
|---|---|---|
| **CREATE** | type_declaration | data_type_name |
| | fb_type_declaration | fb_type_name |
| | fb_instance_definition | fb_instance_reference |
| | connection_definition | connection_start_point |
| **DELETE** | data_type_name | data_type_name |
| | fb_type_name | fb_type_name |
| | fb_instance_reference | fb_instance_reference |
| | connection_definition | connection_definition |
| **START** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **STOP** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **KILL** | fb_instance_reference | fb_instance_reference |





**表7–STATUS输出值和语义**

| Valeur | Statut | Sémantique |
|---|---|---|
| **0** | RDY | |
| **1** | BAD_PARAMS | PARAMS输入值无效 PARAMS |
| **2** | LOCAL_TERMINATION | plication |
| **3** | SYSTEM_TERMINATION | 系统触发关机 |
| **4** | NOT_READY | 处理程序无法处理该命令。 |
| **5** | UNSUPPORTED_CMD | 不支持请求的命令。 |
| **6** | UNSUPPORTED_TYPE | 不支持请求的对象类型。 |
| **7** | NO_SUCH_OBJECT | |
| **8** | INVALID_OBJECT | 对象规范的无效语法 |
| **9** | INVALID_OPERATION | 有序操作对指定对象无效。 |
| **10** | INVALID_STATE | 命令的操作对于对象的当前状态无效。 |
| **11** | OVERFLOW | 之前的交易仍在进行中 |

管理功能块实例的OBJECT输入和RESULT输出的有效长度取决于实现。

OBJECT输入应根据CMD输入指定要进行操作的对象，如果操作成功，则RESULT输出应包含对操作产生的对象的描述。这些字符串的内容必须包括在附件B中定义为非终结符号并在表8中引用的对象的实现相关编码。

**注6：OBJECT输入和RESULT输出的最大允许长度是一个依赖于实现的参数；图22中给出的512的值是说明性的。**

**表8–命令语法**

| CMD | OBJECT | RESULT |
|---|---|---|
| **CREATE** | type_declaration | data_type_name |
| | fb_type_declaration | fb_type_name |
| | fb_instance_definition | fb_instance_reference |
| | connection_definition | connection_start_point |
| **DELETE** | data_type_name | data_type_name |
| | fb_type_name | fb_type_name |
| | fb_instance_reference | fb_instance_reference |
| | connection_definition | connection_definition |
| **START** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **STOP** | fb_instance_reference | fb_instance_reference |
| | application_name | application_name |
| **KILL** | fb_instance_reference | fb_instance_reference |

| CMD | OBJECT | RESULT |
|---|---|---|
| **QUERY** | all_data_types | data_type_list |
| | all_fb_types | fb_type_list |
| | data_type_name | type_declaration |
| | fb_type_name | fb_type_declaration |
| | fb_instance_reference | fb_status |
| | connection_start_point | connection_end_points |
| | application_name | fb_instance_list |
| **READ** | parameter_reference | parameter |
| **WRITE** | referenced_parameter | parameter_reference |
| **RESET** | fb_instance_reference | fb_status |

NOTE Voir Tableau 6 pour les valeurs entières de l'entrée CMD correspondant aux commandes énumérées ci-dessus.

Cela doit être considéré comme une **erreur**, donnant lieu à un code STATUS égal à INVALID_OBJECT, si une commande CREATE tente de créer

- un *bloc fonctionnel* dont le *nom d'instance* duplique celui d'un bloc fonctionnel existant au sein de la même *ressource*,
- un doublon de *connexion*, ou
- plusieurs connexions sur une même *entrée de données.*

La seule exception à la règle ci-dessus consiste en ce qu'une commande CREATE peut remplacer une connexion d'un *parameter* (paramètre) à une *entrée de données* avec une nouvelle connexion de paramètre.

Cela doit être considéré comme une **erreur**, donnant lieu à un code de STATUS égal à UNSUPPORTED_TYPE, si une commande CREATE tente de créer une instance de bloc fonctionnel ou un paramètre d'un *type* qui n'est pas connu du bloc fonctionnel de gestion.

Cela doit être considéré comme une **erreur**, donnant lieu à un code STATUS égal à INVALID_OPERATION, si une commande DELETE tente de supprimer un *type bloc fonctionnel*, une instance de bloc fonctionnel, un *type de données* ou une connexion qui est défini(e) dans la *spécification type* de la *ressource* gérée.

La sémantique des commandes START et STOP doit être comme suit:

- les commandes START et STOP d'une *instance de bloc fonctionnel* doivent être telles que définies en 6.3.2;
- les commandes START et STOP d'une *application* doivent respectivement équivaloir à START et STOP de toutes les *instances du bloc fonctionnel* dans l'application contenue dans la *ressource* gérée;
- la commande STOP d'une *instance de bloc fonctionnel de gestion* doit être l'équivalent de STOP de toutes les *instances de bloc fonctionnel* au sein de la *ressource* gérée;
- la commande START d'une *instance de bloc fonctionnel de gestion* doit être l'équivalent de START de toutes les *instances de bloc fonctionnel* au sein de la *ressource* gérée. Si la ressource gérée a été arrêtée précédemment, cela doit être suivi de l'émission d'un événement sur la sortie appropriée de chaque instance du type de bloc fonctionnel E_RESTART défini dans l'Annexe A. Ces événements doivent se produire aux sorties WARM des blocs E_RESTART si la ressource a été arrêtée en raison d'une précédente commande STOP, ou bien sur les sorties COLD.





| CMD | OBJECT | RESULT |
|---|---|---|
| **QUERY** | all_data_types | data_type_list |
| | all_fb_types | fb_type_list |
| | data_type_name | type_declaration |
| | fb_type_name | fb_type_declaration |
| | fb_instance_reference | fb_status |
| | connection_start_point | connection_end_points |
| | application_name | fb_instance_list |
| **READ** | parameter_reference | parameter |
| **WRITE** | referenced_parameter | parameter_reference |
| | | |

注意上面列出的命令对应的CMD输入整数值见表6。

这应该被认为是一个错误，导致STATUS代码等于
INVALID_OBJECT，如果CREATE命令尝试创建

- 一个功能块，其实例名称与同一资源中现有功能块的名称重复，
- 重复连接，或
  同一数据条目上的多个连接。

上述规则的唯一例外是CREATE命令可以用新的参数连接替换与数据输入的参数连接。

如果CREATE命令尝试创建管理功能块不知道的类型的功能块实例或参数，这应该被视为错误，导致STATUS代码等于UNSUPPORTED_TYPE。

如果DELETE命令尝试删除托管资源的类型规范中定义的功能块类型、功能块实例、数据类型或连接，则应将其视为错误，从而导致STATUS代码等于INVALID_OPERATION。

START和STOP命令的语义应如下所示：

- 功能块实例的START和STOP命令应如6.3.2中所定义；
- 应用程序的START和STOP命令必须分别等效于托管资源中包含的应用程序中功能块的所有实例的START和STOP；
- 管理功能块实例的STOP命令必须等同于受管资源内所有功能块实例的STOP；
- 管理功能块实例的START命令必须等同于受管资源内所有功能块实例的START。如果受管资源先前已关闭，则应在附录A中定义的E_RESTART功能块类型的每个实例的适当输出上发出事件。这些事件应在E_RESTART块的WARM输出处发生如果资源由于先前的STOP命令或COLD输出而停止。

La sémantique spécialisée pour la commande QUERY doit être comme suit:

- lorsque l'entrée OBJECT spécifie une *entrée d'événements, sortie d'événements* ou *sortie de données,* la sortie RESULT doit contenir zéro, un ou plusieurs points d'extrémité opposés;
- lorsque l'entrée OBJECT spécifie une *entrée de données,* la sortie RESULT doit énumérer zéro ou un seul point d'extrémité opposé;
- lorsque l'entrée OBJECT spécifie le nom d'une *application,* la sortie RESULT doit énumérer les noms de tous les blocs fonctionnels dans l'application contenus au sein de la *ressource* gérée.

### 6.3.3 Comportement des blocs fonctionnels gérés

Les blocs fonctionnels qui sont sous le contrôle d'un *bloc fonctionnel de gestion* doivent manifester des comportements opérationnels équivalant à celui montré dans le diagramme de transitions d'états de la Figure 24, en respectant les règles suivantes:

a) Les conditions de transitions en majuscules dans la Figure 24 renvoient à une valeur de l'entrée CMD, telle que spécifiée dans le Tableau 6, du bloc fonctionnel de gestion à l'apparition d'une primitive de service REQ+.

b) La séquence de primitives command_error pour le type de bloc fonctionnel MANAGER doit apparaître, avec la valeur indiquée de la sortie STATUS telle que définie dans le Tableau 7, dans les conditions suivantes:

   1) UNSUPPORTED_CMD: Aucun état n'existe dans la Figure 24 avec une condition de transition pour la valeur CMD spécifiée;

   2) INVALID_STATE: L'état actif courant n'a pas de condition de transition pour la valeur de CMD spécifiée;

   3) UNSUPPORTED_TYPE: La valeur de CMD est CREATE et l'instance de bloc fonctionnel n'existe pas, mais le type de bloc fonctionnel n'est pas connu de l'instance MANAGER, c'est-à-dire que la condition de garde **type_defined** est FALSE;

   4) INVALID_OPERATION: La valeur de CMD est DELETE et l'instance de bloc fonctionnel est dans l'état STOPPED ou KILLED, mais l'instance de bloc fonctionnel est *déclarée* dans la spécification de *type* de l'*équipement ou des ressources*, c'est-à-dire que la condition de garde **is_deletable** est FALSE.

c) La séquence normal_command_sequence de primitives montrées pour le type de bloc fonctionnel MANAGER doit suivre une primitive de service CMD+ dans toutes les autres conditions, avec une valeur de RDY pour la sortie STATUS telle que définie dans le Tableau 7 et avec une valeur correspondante pour la sortie RESULT telle que définie dans le Tableau 8.

d) La sémantique des actions montrées à la Figure 24 doit être telle que montrée dans le Tableau 9 pour les *blocs fonctionnels de base* et *interface de service* gérés.

e) Les actions décrites dans la règle précédente s'appliquent de manière récursive à tous les *blocs fonctionnels composants* des *blocs fonctionnels composés* gérés.

NOTE 1 Les comportements des blocs fonctionnels qui ne sont pas sous le contrôle des blocs fonctionnels de gestion ne relèvent pas du domaine d'application de la présente norme.

NOTE 2 La spécification du comportement des blocs fonctionnels gérés dans des conditions de coupure et de rétablissement de puissance électrique ne relève pas du domaine d'application de la présente norme. Un tel comportement est susceptible d'être spécifié par le fabricant d'un équipement conforme, par exemple par référence à une norme appropriée.

NOTE 3 Des *applications* peuevent'utiliser des *instances* du bloc E_RESTART décrit dans l'Annexe A pour générer des événements susceptibles d'être utilisés pour déclencher des algorithmes appropriés en cas de coupure et de rétablissement de puissance électrique.

NOTE 4 Comme décrit en 5.4.2, le contrôle d'exécution dans des *sous-applications* est entièrement cédé aux mécanismes de contrôle d'exécution de leurs blocs fonctionnels composants et des sous-applications constitutives.



QUERY命令的特殊语义应如下所示：

- 当OBJECT输入指定事件输入、事件输出或数据输出时，RESULT输出必须包含零个或多个相反端点；
- 当OBJECT输入指定数据输入时，RESULT输出必须列出零个或仅列出一个相反的端点；
- 当OBJECT输入指定应用程序的名称时，RESULT输出应列出托管资源中包含的应用程序中所有功能块的名称。

受管理功能块控制的功能块应表现出与图24中状态转换图所示的操作行为等效的操作行为，遵守以下规则：

a)图24中的大写转换条件指的是在REQ+服务原语出现时管理功能块的CMD条目的值，如表6中规定的。

b)MANAGER功能块类型的command_error原语序列应在以下条件下出现，并具有表7中定义的STATUS输出的指示值：

1)UNSUPPORTED_CMD：图24中不存在具有指定CMD值的转换条件的状态；

2）INVALID_STATE：当前活动状态对于指定的CMD值没有转移条件；

3）UNSUPPORTED_TYPE：CMD的值为CREATE，功能块实例不存在，但功能块类型不为MANAGER实例所知，即guardtype_defined的条件为FALSE；

4)INVALID_OPERATION：CMD的值为DELETE，功能块实例处于STOPPED或KILLED状态，但功能块实例在设备或资源的类型规范中声明，即is_deletable保护条件为FALSE。

c)在所有其他条件下，为MANAGER功能块类型显示的原语的normal_command_sequence应遵循CMD+服务原语，表7中定义的STATUS输出值为RDY，而RESULT输出的值如表7中定义。表8。

d)图24所示动作的语义应如表9所示，用于管理的基本功能块和服务接口。

e)前面规则中描述的动作递归地应用于托管复合功能块的所有组件功能块。

注1：不受管理功能块控制的功能块的行为超出了本标准的范围。

注2：电源故障和恢复条件下受管功能块的行为规范超出了本标准的范围。这种行为可能由合格设备的制造商指定，例如通过参考适当的标准。

注3应用程序可以使用附件A中描述的E_RESTART块的实例来生成事件，这些事件可用于触发有关断电和恢复的适当算法。

注4如5.4.2所述，子应用程序中的执行控制完全割让给其组件功能块和组成子应用程序的执行控制机制。

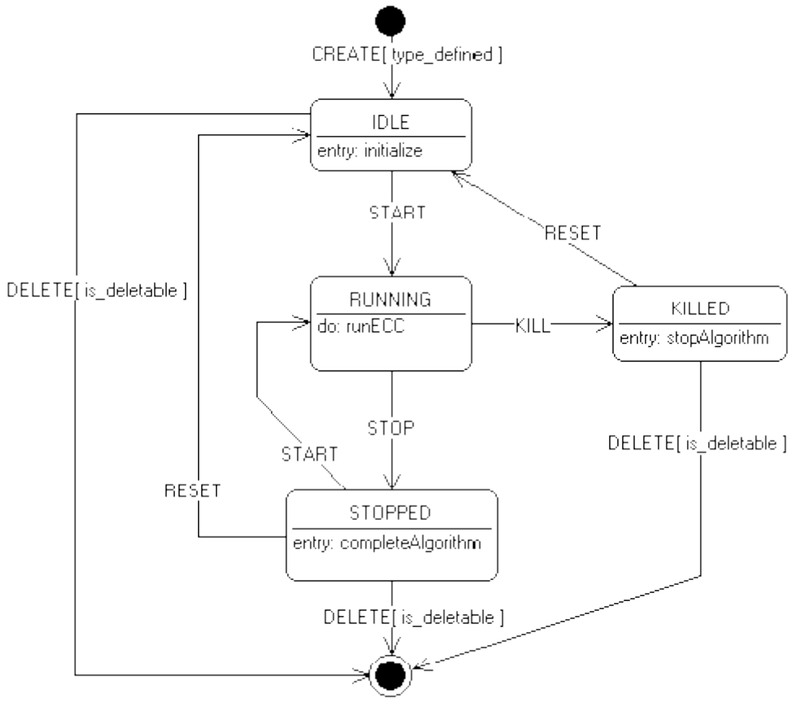



**Figure 24 – Diagramme d'états opérationnels d'un bloc fonctionnel géré**

**Tableau 9 – Sémantique des actions de la Figure 24**

| Action | Blocs fonctionnels de base | Bloc fonctionnel interface de service |
|---|---|---|
| **initialize** | Initialiser toutes les variables comme défini en 5.2.2.1. | |
| | Accomplir d'autres opérations d'initialisation comme défini en 5.2.2.1. | Placer le service dans l'état adéquat pour répondre correctement à une primitive `INIT+`. |
| **runECC** | Activer le diagramme d'états d'opération de l'ECC défini en 5.2.2.2 | Activer l'invocation des primitives de service par des événements aux entrées d'événements, et création d'événements aux sorties d'événements. |
| **completeAlgorithm** | Permettre l'algorithme actuellement actif (s'il y en a) sans autre génération d'événements de sortie. | Laisser se terminer la primitive de service actuellement active. |
| **stopAlgorithm** | Mettre immédiatement fin aux opérations de l'algorithme actuellement actif (s'il y en a). | Mettre immédiatement fin à toutes les opérations du service. |

## 7 Configuration d'unités fonctionnelles et de systèmes

### 7.1 Principes de configuration

L'Article 7 contient des règles pour la *configuration* des *systèmes* de mesure et commande dans les processus industriels (les IPMCS) conformément au modèle suivant:

a) un IPMCS est constitué d'*équipements* connectés entre eux;

b) un *équipement* est une *instance* d'un *type* de l'équipement correspondant;





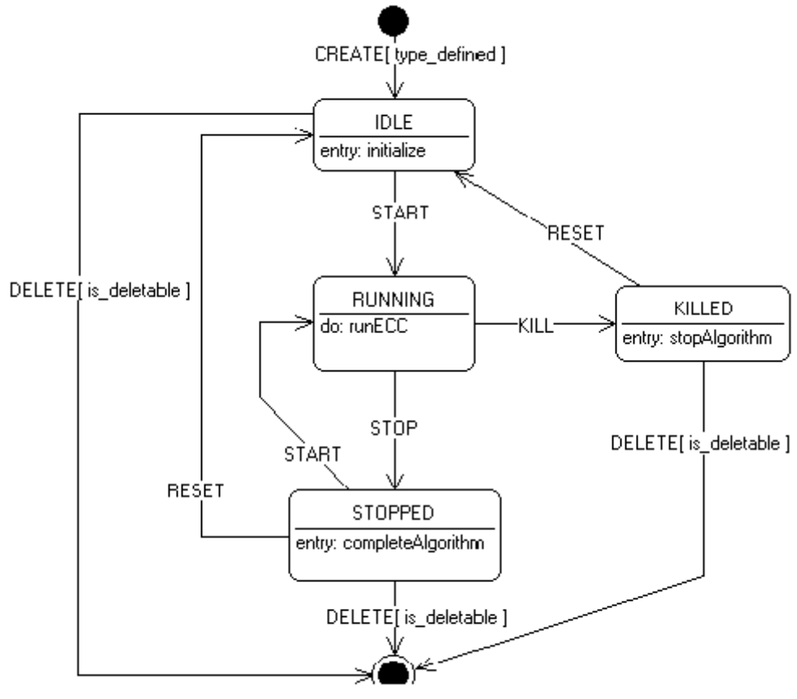



**图24 托管功能块的操作状态图**

**表9 图24中动作的语义**

| Action | 基本构建块 | 服务接口功能块 |
|---|---|---|
| **initialize** | 初始化5.2.2.1中定义的所有变量。 | |
| | 执行5.2.2.1中定义的其他初始化操作。 | 将服务置于正确状态以正确响应INIT+原语。 |
| **runECC** | 激活5.2.2.2定义的ECC运行状态图 | 在事件输入处启用事件调用服务原语，并在事件输出处创建事件。 |
| completeAlgorithm在不进一步生成输出事件的情况下启用当前活动的算法（如果有）。 | | 让当前活动的服务原语完成。 |
| **stopAlgorithm** | 立即终止当前活动算法的操作（如果有）。 | 立即终止所有服务操作。 |

## 7 功能单元和系统的配置

### 配置原则

第7条包含根据以下模型配置工业过程测量和控制系统(IPMCS)的规则：

a)IPMCS由连接在一起的设备组成；

b)一个设备是相应设备的一个类型的一个实例；

c) les capacités fonctionnelles d'un *type d'équipement* sont décrites en termes des *ressources* qui lui sont associées;

d) une *ressource* est une *instance* d'un *type de ressource* correspondant;

e) les capacités fonctionnelles d'un *type de ressource* sont décrites en termes des *types de blocs fonctionnels* qui peuvent être *instanciés* et des *instances de blocs fonctionnels* qui existent, dans toutes les *instances* du *type* de ressource.

On considère donc que la *configuration* d'un IPMCS consiste en la *configuration* de ses *équipements* et *applications,* associés, y compris l'allocation d'*instances de bloc fonctionnel* dans chaque *application* aux *ressources* associées aux *équipements*. L'Article 7 définit les ensembles de règles suivantes pour venir à l'appui de ce processus:

- les règles pour la spécification fonctionnelle des *types* de *ressources* et des *équipements* sont définies en 7.2;
- les règles pour la *configuration* d'un IPMCS en termes des *équipements* et d'*applications* associés sont définies en 7.3.

### 7.2 Spécification fonctionnelle des types de ressources, d'équipements et de segments

#### 7.2.1 Spécification fonctionnelle des types de ressources

La spécification fonctionnelle d'un *type de ressource* comprend:

- le *nom du type* de la ressource;
- le *nom d'instance,* le *type de données* et l'initialisation de chacun des *paramètres* de la ressource;
- une déclaration des *types de données* et des *types de bloc fonctionnel* que chaque *instance* du *type de ressource* est capable d'*instancier*;
- les noms d'instance, les types et les valeurs initiales de toutes les instances de bloc fonctionnel qui sont toujours présentes dans chaque instance du type de la ressource;
- toutes les *connexions de données, connexions d'adaptateurs* et *connexions d'événements* qui sont toujours présentes dans chaque instance du type de la ressource.

NOTE 1 Des informations complémentaires peuvent'être fournies avec les spécifications types des ressources, y compris:

– les nombres maximum de *connexions de données,* de *connexions d'adaptateurs* et de *connexions d'événements* qui peuvent exister dans une instance type de la ressource;

– le temps (identifié comme étant "$T_{alg}$" à la Figure 7) requis pour l'*exécution* de chaque *algorithme* des blocs fonctionnels d'un type spécifié dans une instance de la ressource;

– le nombre maximal d'instances de type de blocs fonctionnels spécifiés qui sont susceptibles d'exister dans chaque instance de la ressource;

– des compromis entre les instances de blocs fonctionnels, par exemple: savoir si deux instances du type bloc fonctionnel "A" peuvent être échangées contre une seule instance de type "B", etc.

NOTE 2 Les spécifications fonctionnelles des *interfaces* de communication et des processus d'une ressource, y compris le genre et le degré de conformité à des normes applicables, ne relèvent pas du domaine d'application de la présente norme, sauf lorsque ces interfaces sont représentées par des *blocs fonctionnels interface de service.*

#### 7.2.2 Spécification fonctionnelle des types d'équipements

La spécification fonctionnelle d'un *type d'équipement* comprend:

a) le *nom de type* de l'équipement;

b) le *nom d'instance,* le *type de données* et l'initialisation de chacun des *paramètres* de l'équipement;

c) le nom d'instance, le nom du type et l'initialisation de chaque *instance de bloc fonctionnel* qui est toujours présente dans chaque *instance* du type d'équipement;



c)设备类型的功能能力根据与其相关的资源进行描述；

d)资源是相应资源类型的实例；

e)资源类型的功能能力是根据可以实例化的功能块的类型以及资源类型的所有实例中存在的功能块的实例来描述的。

因此认为，IPMCS的配置包括其相关设备和应用程序的配置，包括将每个应用程序中的功能块实例分配给与设备相关联的资源。第7条定义了以下规则集以支持此过程：

- 资源类型和设备功能规范的规则在7.2中定义；
- 在7.3中定义了根据相关设备和应用配置IPMCS的规则。

### 7.2 资源、设备和设备类型的功能规范
**se**

#### 资源类型的功能规范

资源类型的功能规范包括：

- 资源类型名称；
- 每个资源参数的实例名称、数据类型和初始化；
- 资源类型的每个实例能够实例化的数据类型和功能块类型的声明；
- 资源类型的每个实例中始终存在的所有功能块实例的实例名称、类型和初始值；
- 始终存在于资源类型的每个实例中的所有数据连接、适配器连接和事件连接。

注1附加信息可随典型资源规范提供，包括：

– 资源的典型实例中可以存在的最大数据连接数、适配器连接数和事件连接数；

– 执行每个块算法所需的时间（在图7中标识为"Talg"）
在资源实例中指定的类型的函数；

– 资源的每个实例中可能存在的指定功能块类型的实例的最大数量；

功能块实例之间的权衡，例如：功能块类型"A"的两个实例是否可以交换为"B"类型的单个实例等。

注2：资源的通信和过程接口的功能规范，包括适用标准的种类和符合程度，不在本标准的范围之内，除非这些接口由服务接口功能块描述。

#### 设备类型的功能规范

一类设备的功能规范包括：

(a)设备的型号名称；

b)每个设备参数的实例名称、数据类型和初始化；

c)每个设备类型实例中始终存在的每个功能块实例的实例名称、类型名称和初始化；

d) toutes *connexions de données, connexions d'adaptateurs* et *connexions d'événements* qui sont toujours présentes dans chaque instance du type d'équipement.

e) déclarations des *instances des ressources* qui sont présentes dans chaque instance du type d'équipement. Chaque déclaration doit contenir:

   1) le nom d'instance et le nom du type de la ressource;

   2) le nom d'instance, le nom du type et l'initialisation de chaque *instance de bloc fonctionnel* qui est toujours présente dans l'instance de ressource dans chaque instance du type d'équipement;

   3) toutes les *connexions de données, connexions d'adaptateurs* et *connexions d'événements* qui sont toujours présentes dans l'instance de ressource dans chaque instance du type d'équipement.

NOTE 1 Les points (2) et (3) ci-dessus sont considérés comme un ajout aux éléments correspondants déclarés dans la spécification du type de ressources telle que définie en 7.2.1.

NOTE 2 Les spécifications fonctionnelles des *interfaces* de communication et de processus d'un équipement, y compris le genre et le degré de conformité à des normes applicables, ne relèvent pas du domaine d'application de la présente norme, sauf lorsque ces interfaces sont représentées par des *blocs fonctionnels interface de service*.

NOTE 3 Un type d'équipement est susceptible de contenir un réseau de blocs fonctionnels seulement si l'on considère qu'il est constitué d'une seule ressource (non déclarée); dans ce cas, le type d'équipement ne contient aucune déclaration d'instances de ressource.

### 7.2.3 Spécification fonctionnelle des types de segments

La spécification fonctionnelle d'un *type de segment* comprend:

- le *nom du type* de segment;
- le *nom d'instance,* le *type de données* et l'initialisation de chacun des *paramètres* du segment.

## 7.3 Exigences relatives à la configuration

### 7.3.1 Configuration des systèmes

La configuration d'un *système* comprend:

- le *nom* du système;
- la spécification de chaque *application* dans le système, telle que définie en 7.3.2;
- la configuration de chaque *équipement* et de ses *ressources* associées, telle que définie en 7.3.3;
- la configuration de chaque *segment de réseau* et de ses *liaisons* associées à des équipements ou à des ressources, telle que spécifiée en 7.3.4.

### 7.3.2 Spécification d'applications

La spécification d'une *application* consiste en:

- son nom sous la forme d'un *identificateur*;
- le *nom d'instance,* le *nom du type,* les *connexions de données,* les *connexions d'événements* et les *connexions d'adaptateur* de chaque *bloc fonctionnel* et *sous-application* dans l'application.

Cela doit être considéré comme une **erreur** si le nom d'une application n'est pas unique au sein de la portée du *système*.

### 7.3.3 Configuration des équipements et des ressources

La configuration d'un *équipement* comprend:

- le *nom d'instance* et le *nom du type* d'équipement;



d)始终存在于设备类型的每个实例中的任何数据连接、适配器连接和事件连接。

e)设备类型的每个实例中存在的资源实例的声明。每个声明必须包含：

1）资源的实例名和类型名；

2)每个设备类型实例的资源实例中始终存在的每个功能块实例的实例名称、类型名称和初始化；

3)在设备类型的每个实例中，始终存在于资源实例中的所有数据连接、适配器连接和事件连接。

注1上述项目(2)和(3)被认为是对7.2.1中定义的资源类型规范中声明的相应项目的补充。

注2：设备的通信和过程接口的功能规范，包括符合适用标准的种类和程度，不在本标准的范围之内，除非这些接口由服务接口功能块表示。

注3：一个设备类型可能包含一个功能块网络，只有当它被认为是由一个单一的（未声明的）资源组成时；在这种情况下，设备类型不包含资源实例的声明。

#### 段类型的功能规范

段类型的功能规范包括：

- 段类型名称；
- 每个段参数的实例名称、数据类型和初始化。

## 7.3 设置要求

#### 系统配置

一个系统的配置包括：

- 系统名称；
- 系统中每个应用程序的规范，如7.3.2中所定义；
- 7.3.3中定义的每件设备及其相关资源的配置；
- 每个网段的配置及其与设备或资源的关联链接，如7.3.4中所述。

应用程序的规范包括：

- 它的名称以标识符的形式；
- 应用程序中的每个功能块和子应用程序的实例名称、类型名称、数据连接、事件连接和适配器连接。

如果应用程序的名称在系统范围内不是唯一的，则应将其视为错误。

#### 设备和资源的配置

设备配置包括：

- 实例名称和设备类型名称；

- les valeurs spécifiques à la configuration pour les *paramètres* de léquipement;
- les *types de ressources* pris en charge par l'*instance* de l'équipement en plus de ceux qui sont spécifiés pour le *type* d'équipement;
- le *nom d'instance* et le *nom du type* de chaque *instance de bloc fonctionnel* qui est présente dans l'instance de l'équipement en plus de ceux qui sont définis pour le type d'équipement;
- toutes les *connexions de données, connexions d'adaptateurs* et *connexions d'événements* qui sont présentes dans l'instance de l'équipement en plus de celles qui sont définies pour le type d'équipement;
- les *types de ressources* pris en charge par l'*instance* de l'équipement en plus de ceux qui sont spécifiés pour le *type* d'équipement;
- la configuration de chacune des *ressources* dans l'équipement. Celles-ci sont constituées de toutes les instances de ressources définies dans la spécification des *types* d'équipements plus toutes les éventuelles ressources associées à l'*instance* de l'équipement spécifique.

NOTE Une instance d'équipement est susceptible de contenir un réseau de blocs fonctionnels seulement si l'on considère qu'il est constitué d'une seule ressource (non déclarée); dans ce cas, la déclaration de l'instance de l'équipement ne contient aucune déclaration d'instances de ressource.

Cela doit être considéré comme une **erreur** si le nom d'instance dans chaque équipement n'est pas unique au sein de la portée du système.

La configuration d'une *ressource* comprend:

a) son *nom d'instance* et son *nom de type*;

b) les *types de données* et les *types de bloc fonctionnel* pris en charge par l'*instance* ressource*;*

c) le *nom d'instance*, le *nom de type* et l'initialisation de chaque instance de bloc fonctionnel qui est présente dans l'instance ressource;

d) toutes *connexions de données, connexions d'événements* et *connexions d'adaptateur* qui sont présentes dans l'instance ressource.

La configuration des ressources est assujettie aux règles suivantes:

– Les points b), c), et d) ci-dessus sont considérés comme un ajout aux éléments correspondants déclarés dans les spécifications des types d'équipements et de ressources telles que respectivement définies en 7.2.2 et 7.2.1.

– Les points c) et d) comprennent des *instances de bloc fonctionnel*, des *connexions de données, connexions d'adaptateurs* et des *connexions d'événements* issues des parties d'*applications* allouées à la ressource.

– Les points c) et d) comprennent des *blocs fonctionnels de communication, des connexions de données,* des *connexions d'événements* et des *connexions d'adaptateurs*, selon les besoins, pour établir et maintenir les flots de données et d'événements pour les *applications* associées.

– Les éléments dans le point c) peuvent inclure le *mapping* d'instances de bloc fonctionnel dans l'application à des instances de bloc fonctionnel existant dans la ressource en raison de la définition de types décrite en 7.2.1.

– Cela doit être considéré comme une **erreur** si le nom d'instance d'une ressource n'est pas unique dans la portée de l'équipement la contenant ou si une instance de bloc fonctionnel quelconque dans une application n'est pas allouée à exactement une seule ressource.

Un moyen automatisé peut être fourni pour satisfaire aux exigences ci-dessus. Les fournisseurs de tels moyens doivent donner des règles non ambiguës permettant de déterminer leur opération ou bien ils doivent fournir un moyen permettant d'examiner et de modifier les résultats de l'application de ces moyens.





- 设备参数的配置特定值；
- 除了为设备类型指定的资源类型之外，设备实例支持的资源类型；
- 除了为设备类型定义的那些之外，设备实例中存在的每个功能块实例的实例名称和类型名称；
- 除了为设备类型定义的连接之外，设备实例中存在的所有数据连接、适配器连接和事件连接；
- 除了为设备类型指定的资源类型之外，设备实例支持的资源类型；
- 设备中每个资源的配置。这些包括在设备类型规范中定义的所有资源实例以及与特定设备实例相关的任何资源。

注：一个设备实例只有在被认为由单个（未声明的）资源组成时才可能包含功能块网络；在这种情况下，设备实例声明不包含任何资源实例声明。

如果每个设备中的实例名称在系统范围内不是唯一的，则应将其视为错误。

配置资源包括：

a)它的实例名称和类型名称；

b)资源实例支持的数据类型和功能块类型；

c)资源实例中存在的每个功能块实例的实例名称、类型名称和初始化；

d)资源实例中存在的任何数据连接、事件连接和适配器连接。

资源配置遵循以下规则：

– 上述b)、c)和d)点被认为是对分别在7.2.2和7.2.1中定义的设备和资源类型规范中声明的相应要素的补充。

– 项c)和d)包括来自分配给资源的部分应用程序的功能块实例、数据连接、适配器连接和事件连接。

– 项目c)和d)包括通信构建块、数据连接、事件连接和适配器连接，根据需要，为关联的应用程序建立和维护数据和事件流。

– 由于7.2.1中描述的类型定义，项c)中的项可能包括应用程序中的功能块实例到资源中现有功能块实例的映射。

– 如果资源的实例名称在包含它的设备范围内不是唯一的，或者应用程序中的任何功能块实例未分配给仅一个资源，则应将其视为错误。

可以提供一种自动化装置来满足上述要求。此类手段的提供者必须提供确定其操作的明确规则，或者他们必须提供一种手段来检查和修改此类手段的应用结果。

### 7.3.4 Configuration des segments et des liaisons réseau

La configuration d'un *segment de réseau* comprend:

- le *nom d'instance* et le *nom du type* du segment;
- des valeurs spécifiques à une configuration pour les paramètres du segment du réseau.

Cela doit être considéré comme une **erreur** si le *nom d'instance* de chaque segment de réseau n'est pas unique dans la portée du *système* ou si les valeurs déclarées des paramètres de segment sont incompatibles avec la déclaration (éventuelle) du *type de segment* défini en 7.2.3.

La configuration d'une *liaison* comprend:

– le nom d'un *équipement* ou le nom hiérarchique d'une "*ressource* de communication" à l'intérieur d'un équipement et le nom du segment du réseau auquel l'équipement ou la ressource est relié(e);
– des valeurs spécifiques à une configuration pour les *paramètres* de la liaison.





**配置网段和链路**

配置网段包括：

- 段的实例名称和类型名称；
  网段参数的配置特定值。

如果每个网段的实例名称在系统范围内不是唯一的，或者段参数的声明值与7.2中定义的段类型的（如果有的话）声明不一致，这应该被认为是一个错误.3.

配置链接包括：

– 设备名称或设备内"通信资源"的分层名称，以及该设备或资源所连接的网段名称；

– 绑定参数的配置特定值。

# Annexe A
(normative)

## Blocs fonctionnels d'événements

Des *instances* des *types* de bloc fonctionnel montrées dans le Tableau A.1 peuvent être utilisées pour la génération et le traitement d'*événements* dans des *blocs fonctionnels composés*; dans des *sous-applications*; pour la définition des types de *ressource* et d'*équipement*; et dans la *configuration* d'*applications*, des *ressources* et des *équipements*.

Les types de bloc fonctionnel montrés dans l'Annexe A qui utilisent des *graphiques de contrôle d'exécution* sont des *types de bloc fonctionnel de base*. Lorsque des déclarations textuelles d'*algorithmes* sont données pour ces types de bloc fonctionnel, le langage utilisé est le langage Structured Text (ST) défini dans la CEI 61131-3.

Des mises en œuvre de référence pour certains des types de bloc fonctionnel dans l'Annexe A sont données comme des définitions de *type de bloc fonctionnel composé*. Ces mises en œuvre sont normatives seulement dans le sens où les comportements fonctionnels des mises en œuvre conformes doivent être équivalents à ceux de la mise en œuvre de référence, avec les considérations suivantes qui s'appliquent aux paramètres de temporisation définis en 4.5.3:

- Les paramètres $T_{\text{setup}}$, $T_{\text{start}}$ et $T_{\text{finish}}$ sont considérés être égaux à zéro (0) pour tous les *blocs fonctionnels composants* dans la mise en œuvre de référence.
- Le paramètre $T_{\text{alg}}$ est considéré égal au paramètre DT pour toutes les instances du type E_DELAY utilisées comme *blocs fonctionnels composants* dans la mise en œuvre de référence, tandis qu'il est considéré égal à zéro (0) pour tous les autres blocs fonctionnels composants dans la mise en œuvre de référence.

Tous les autres types de bloc fonctionnel donnés dans l'Annexe A sont des *blocs fonctionnels de type interface de service*.

NOTE L'Annexe F donne des spécifications textuelles complètes de tous les types de bloc fonctionnel montrés dans le Tableau A.1.



61499-1 © CEI:2012 – 181 –

Annexe A
(normative)

Blocs fonctionnels d'événements

表A.1所示功能块类型的实例可用于复合功能块中的事件生成和处理；在子应用程序中；用于定义资源和设备类型；在应用程序、资源和设备的配置中。

附录A中显示的使用执行控制图的功能块类型是基本功能块类型。当为这些类型的功能块给出算法的文本声明时，使用的语言是IEC61131-3中定义的结构化文本(ST)语言。

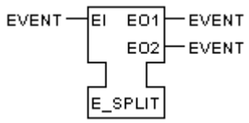
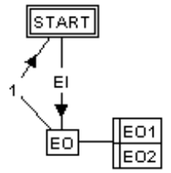
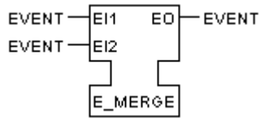
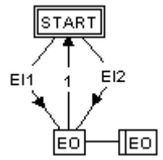
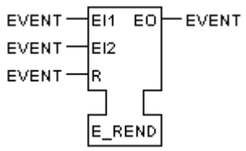
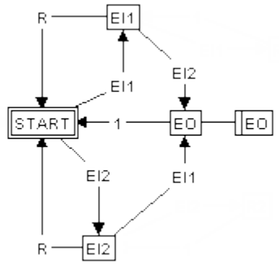
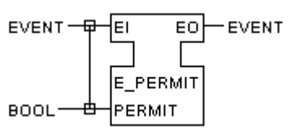
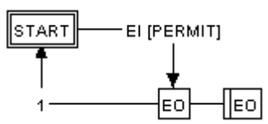
附录A中某些功能块类型的参考实现以复合功能块类型定义的形式给出。这些实现是规范的，仅在符合实现的功能行为应与参考实现的功能行为等效的意义上，以下考虑适用于4.5.3中定义的时序参数：

- 对于参考实现中的所有组件功能块，假定Tsetup、Tstart和Tfinish参数为零(0)。
- 对于在参考实现中用作组件功能块的所有E_DELAY类型实例，Talg参数被认为等于DT参数，而对于参考实现中的所有其他组件功能块，它被认为等于零(0)。

附录A中给出的所有其他功能块类型都是服务接口类型功能块。

注：附录F给出了表A.1中所有功能块类型的完整文本规范。

**Tableau A.1 – Blocs fonctionnels d'événements** (1 de 7)

| No. | Description | |
|---|---|---|
| | **Interface** | **Séquences ECC/Algorithmes/Service** |
| **1** | **Diviser un événement** | |
| |  |  |
| L'apparition d'un événement en EI entraîne l'apparition d'événements en EO1, EO2,…,Eon (n=2 dans l'exemple ci-dessus). | | |
| **2** | **Fusion (OR) de plusieurs événements** | |
| |  |  |
| L'apparition d'un événement en l'une quelconque des entrées EI1, EI2,…,EIn entraîne l'apparition d'un événement en EO (n=2 dans l'exemple ci-dessus). | | |
| **3** | **Rendez-vous de deux événements** | |
| |  |  |
| **4** | **Propagation permissive d'un événement** | |
| |  |  |





表A.1–事件功能块(1of7)

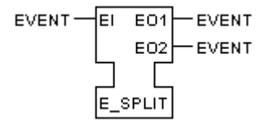
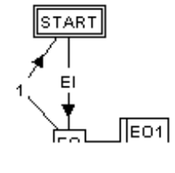
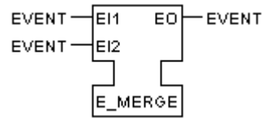
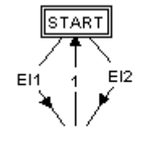
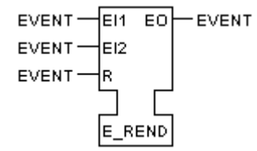
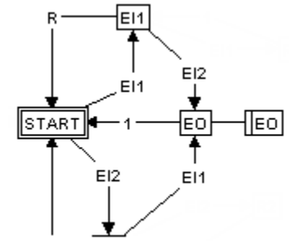
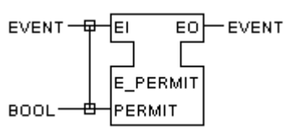
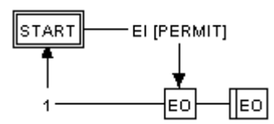
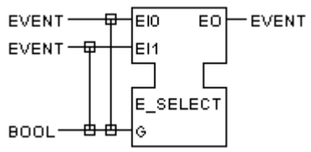
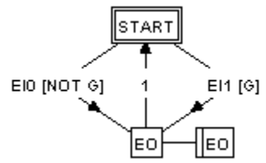
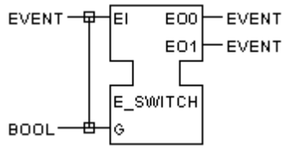
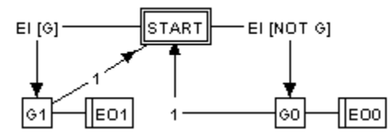
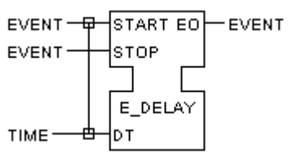
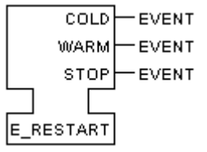
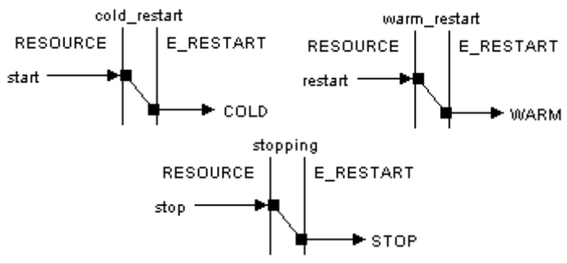
| No. | Description | |
|---|---|---|
| | **Interface** | **Séquences ECC/Algorithmes/Service** |
| **1** | **Diviser un événement** | |
| |  |  |
| EI中事件的出现导致EO1、EO2、…、Eon中的事件出现（在上面的示例中n=2）。 | | |
| **2** | **多个事件的合并(OR)** | |
| |  |  |
| 在输入EI1、EI2、…、EIn中的任何一个处出现事件会导致在EO处出现事件（在上面的示例中，n=2）。 | | |
| **3** | **两个活动的会议** | |
| |  |  |
| **4** | **事件的许可传播** | |
| |  |  |

**Tableau A.1** (2 de 7)

| No. | Description | |
|---|---|---|
| | **Interface** | **Séquences ECC/Algorithmes/Service** |
| **5** | **Sélection entre deux événements** | |
| |  |  |
| **6** | **Commutation (démultiplexage) d'un événement** | |
| |  |  |
| **7** | **Propagation différée d'un événement** | |
| |  | Un événement en EO est généré à un intervalle de temps DT après l'apparition d'un événement sur l'entrée START. Le retard pour l'événement est annulé par l'apparition d'un événement sur l'entrée STOP. Si plusieurs événements apparaissent sur l'entrée START avant l'apparition d'un événement en EO, seul un événement simple apparaît en EO, avec un temps DT après l'apparition du premier événement sur l'entrée START. Aucun retard d'événement ne sera déclenché si un événement apparaît sur l'entrée START avec une valeur de DT qui n'est pas supérieure à t#0s. |
| **8** | **Génération d'événements de redémarrage** | |
| |  |  |

a) Un événement est émis sur la sortie COLD à la suite d'un «redémarrage à froid» de la ressource associée.

b) Un événement est émis sur la sortie WARM à la suite d'un «redémarrage à chaud» de la ressource associée.

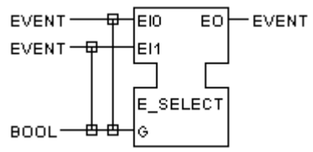
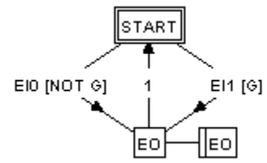
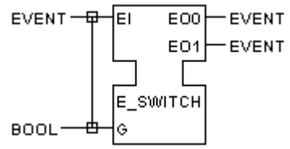
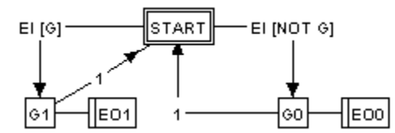
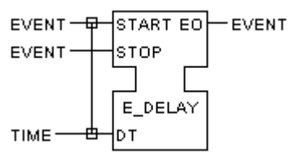
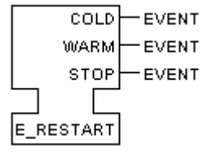
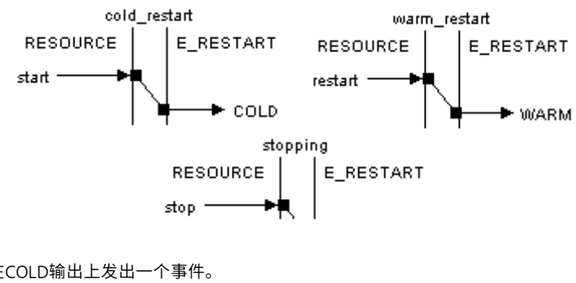
c) Un événement est émis sur la sortie STOP (si possible) avant «l'arrêt» de la ressource associée.

NOTE 1 Voir la CEI 61131-1 pour un débat relatif au «redémarrage à froid» et «redémarrage à chaud».





表A.1（2个，共7个）

| No. | Description | |
|---|---|---|
| | **Interface** | **Igorithmes/Service** |
| **5** | **两个事件之间的选择** | |
| |  |  |
| **6** | **Commutation (démultiplexage) d'un événement** | |
| |  |  |
| **7** | | |
| |  | 在START输入上出现事件后，在时间间隔DT生成EO中的事件。事件的延迟因STOP输入上出现事件而被取消。如果在一个事件出现在EO之前，多个事件出现在START输入上，则只有一个事件出现在EO中，时间DT在第一个事件出现在START输入之后。如果START输入上出现事件且DT值不大于t#0s，则不会触发事件延迟。 |
| **8** | **Génération d'événements de redémarrage** | |
| |  |  |

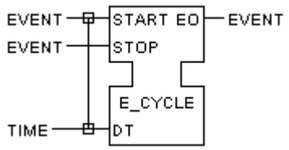
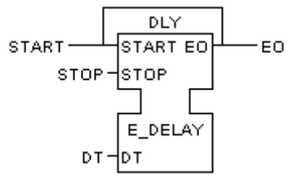
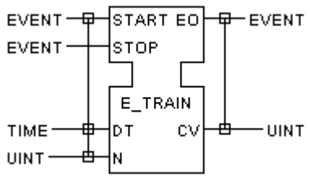
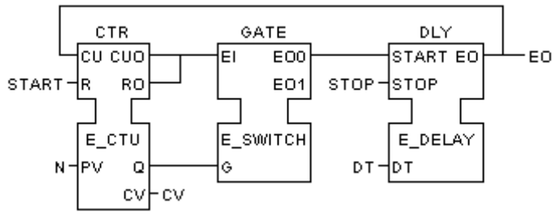
a) 在关联资源的"冷重启"之后，在COLD输出上发出一个事件。

b) 在关联资源的"热重启"之后，在WARM输出上发出一个事件。

在关联资源的"停止"之前，在STOP输出（如果可能）上发出一个事件。

注1有关"冷启动"和"热启动"的讨论，请参见IEC61131-1。

**Tableau A.1** (3 de 7)

| No. | Description |
|---|---|
| Interface | Séquences ECC/Algorithmes/Service |
| 9 | Génération périodique (cyclique) d'un événement |
|  Un événement apparaît en EO avec un intervalle DT après l'apparition d'un événement en START, et avec des intervalles DT par la suite jusqu'à l'apparition d'un événement en STOP. |  |
| 10 | Génération d'un train fini d'événements |
|  |  NOTE 2 Voir l'entrée 18 du tableau pour une définition du type E_CTU. |
| Un événement apparaît en EO avec un intervalle DT après l'apparition d'un événement en START et à des intervalles DT par la suite jusqu'à ce que N apparitions aient été générées ou qu'un événement apparaisse sur l'entrée STOP.<br><br>NOTE 3 Le compte CV est réinitialisé chaque fois qu'un événement apparaît sur l'interface START, mais le retard ne redémarre pas à moins qu'il n'ait déjà été arrêté. Ce comportement maintient l'intervalle entre EO lors du redémarrage du compte. | |





**表A.1（3个，共7个）**

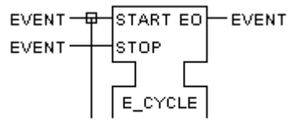
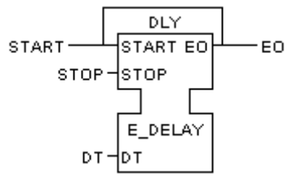
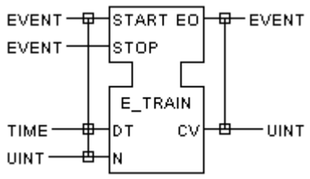
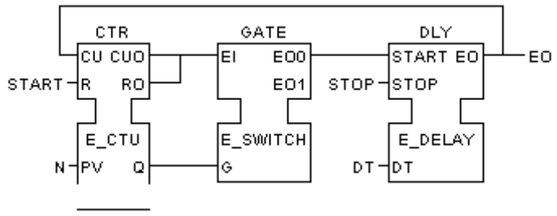
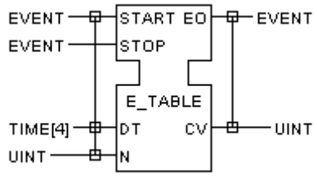
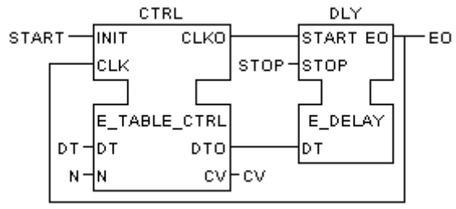
| No. | Description |
|---|---|
| Interface | Séquences ECC/Algorithmes/Service |
| 9 | Génération périodique (cyclique) d'un événement |
|  事件发生在EO中，事件发生在START之后有一个DT间隔，之后有一个DT间隔，直到事件发生在STOP中。 |  |
| 10 | 生成有限的事件序列 |
|  |  注2类型定义见表条目18 |
| 一个事件出现在EO中，在START中的事件发生后有一个DT间隔，此后在DT间隔中出现，直到N次出现或事件出现在STOP输入上。<br><br>注3每次在START界面上出现事件时，CV计数都会重置，但延迟不会重新开始，除非它已经停止。此行为在重新启动帐户时保持EO之间的间隔。 | |

**Tableau A.1** (4 de 7)

| No. | Description | |
|---|---|---|
| | Interface | Séquences ECC/Algorithmes/Service |
| 11 | Génération d'un train d'événements (guidée par table) | |
| |  |  |
| | Un événement apparaît en EO avec un intervalle DT[0] après l'apparition d'un événement en START. Un deuxième événement apparaît avec un intervalle DT[1] après le premier, etc., jusqu'à ce que N apparitions aient été générées ou qu'un événement apparaisse sur l'entrée STOP. Le compte d'événements courant est maintenu sur la sortie CV.<br><br>NOTE 4 Dans cet exemple de mise en œuvre, N <= 4.<br><br>NOTE 5 La mise en œuvre utilisant le type de bloc fonctionnel E_TABLE_CTRL illustré ci-dessous n'est pas une exigence normative. Une fonctionnalité équivalente peut être mise en œuvre par divers moyens. | |
| |  |  |
| | ALGORITHM INIT IN ST:<br>  CV:= 0;<br>  DTO:= DT[0];<br>END_ALGORITHM | ALGORITHM STEP IN ST:<br>  CV:= CV+1;<br>  DTO:= DT[CV];<br>END_ALGORITHM |





**表A.1（4个，共7个）**

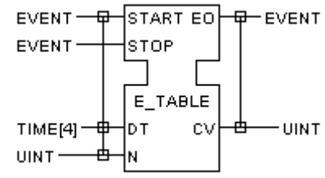
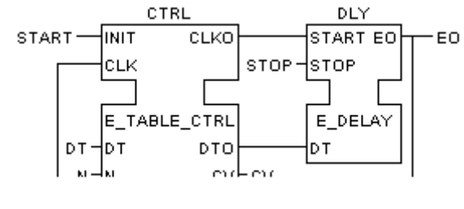
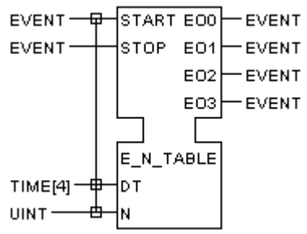
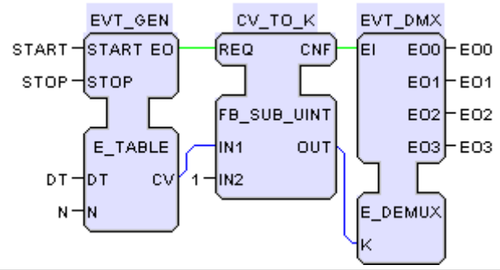
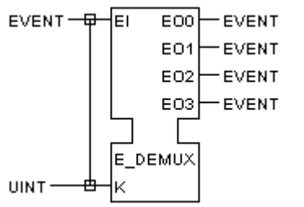
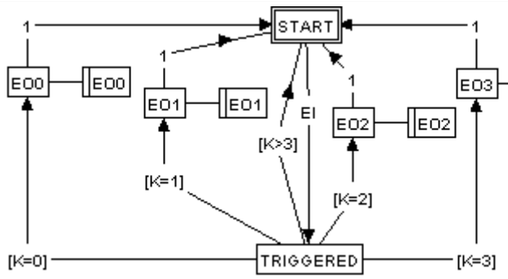
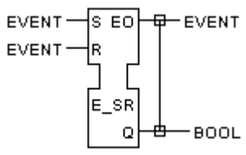
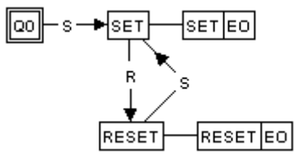
| No. | Description | |
|---|---|---|
| | Interface | /Service |
| 11 | 生成事件序列（由表引导） | |
| |  |  |
| | 在START中发生事件之后，EO中出现一个事件，间隔DT[0]。第二个事件出现在第一个事件之后，间隔DT[1]，依此类推，直到产生N次事件或事件出现在STOP输入上。当前事件计数保持在CV输出上。<br><br>注4在此示例实现中，N<=4。<br><br>注5使用如下所示的E_TABLE_CTRL功能块类型的实现不是标准要求。等效功能可以通过多种方式实现。 | |
| |  |  |
| | ST中的算法初始化：<br>  CV:= 0;<br>  DTO:= DT[0];<br>END_ALGORITHM | ST中的算法步骤：<br>  CV:= CV+1;<br>  DTO:= DT[CV];<br>END_ALGORITHM |

**Tableau A.1** (5 de 7)

| No. | Description | |
|---|---|---|
| | **Interface** | **Séquences ECC/Algorithmes/Service** |
| **12** | **Génération d'un train d'événements distincts (guidée par table)** | |
| |  |  |
| Un événement apparaît en EO0 avec un intervalle DT[0] après l'apparition d'un événement en START. Un événement apparaît en EO1 avec un intervalle DT[1] après l'apparition de l'événement en EO0, etc., jusqu'à ce que N apparitions aient été générées ou qu'un événement apparaisse sur l'entrée STOP.<br><br>NOTE 6 Dans cet exemple de mise en œuvre, N <= 4.<br><br>NOTE 7 La mise en œuvre utilisant le type de bloc fonctionnel E_DEMUX illustré ci-dessous n'est pas une exigence normative. Une fonctionnalité équivalente peut être mise en œuvre par divers moyens. | | |
| |  |  |
| **13** | **Bistable piloté par des événements** | |
| |  |  |
| La sortie Q est mise à 1(TRUE) à la suite de l'apparition d'un événement sur l'entrée S, et elle est réinitialisée à 0 (FALSE) à la suite de l'apparition d'un événement sur l'entrée R. Un événement est émis sur la sortie EO lorsque la valeur de Q change. | | |
| ALGORITHM SET IN ST: (* Set Q *)<br>Q:= TRUE;<br>END_ALGORITHM | ALGORITHM RESET IN ST: (* Reset Q *)<br>Q:= FALSE;<br>END_ALGORITHM | |





**表A.1（5个，共7个）**

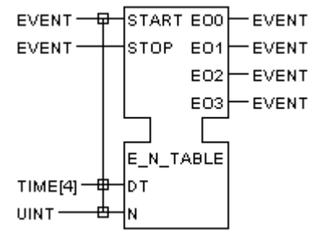
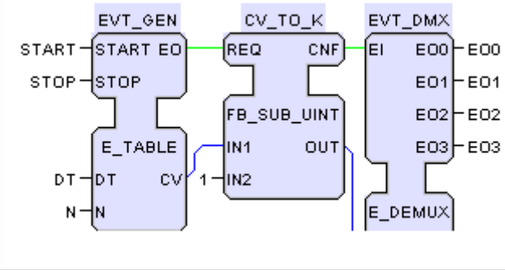
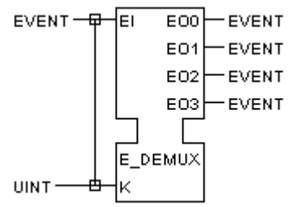
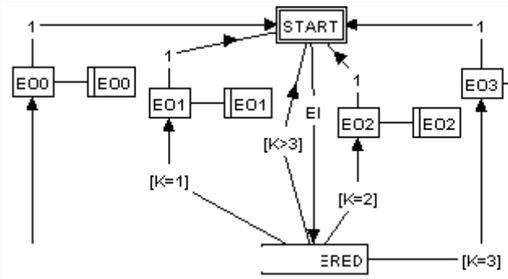
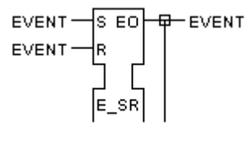
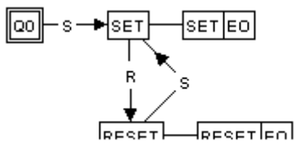
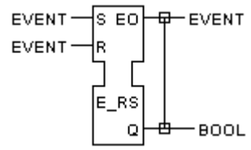
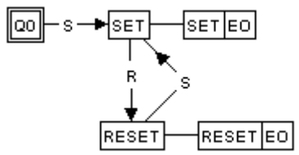
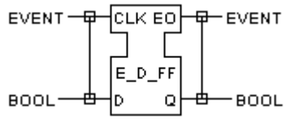
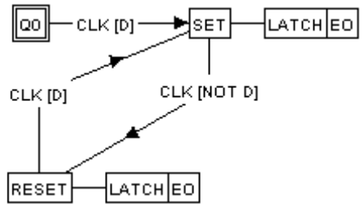
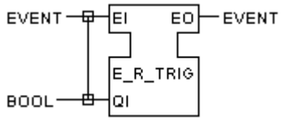
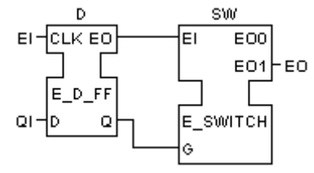
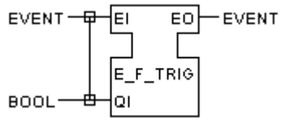
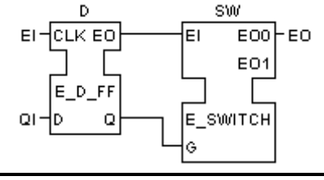
| No. | Description | |
|---|---|---|
| | **Interf** | **vice** |
| **12** | **生成一系列不同的事件（由表格引导）** | |
| |  |  |
| 事件出现在EO0中，间隔DT[0]在START中出现事件之后。在EO0中的事件发生之后，以间隔DT[1]的时间在EO1中出现一个事件，依此类推，直到发生N次或在STOP输入上出现一个事件。<br><br>注6在此示例实现中，N<=4。<br><br>注7使用如下所示的E_DEMUX功能块类型的实现不是标准要求。等效功能可以通过多种方式实现。 | | |
| |  |  |
| **13** | **事件驱动双稳态** | |
| |  |  |
| 输出Q在输入S上出现事件后设置为1(TRUE)，并在R条目上出现事件后重置为0(FALSE)。当Q的值改变时，输出EO上会发出一个事件。 | | |
| ST中的算法集：（*集Q*）<br>Q:= TRUE;<br>END_ALGORITHM | ST中的算法重置：（*重置Q*）<br>Q:= FALSE;<br>END_ALGORITHM | |

**Tableau A.1** (6 de 7)

| No. | Description | |
|---|---|---|
| | **Interface** | **Séquences ECC/Algorithmes/Service** |
| **14** | **Bistable piloté par des événements** | |
| |  |  |
| La sortie Q est mise à 1 (TRUE) à la suite de l'apparition d'un événement sur l'entrée S, et elle est réinitialisée à 0 (FALSE) à la suite de l'apparition d'un événement sur l'entrée R. Un événement est émis sur la sortie EO lorsque la valeur de Q change.<br><br>NOTE La mise en œuvre de ce type de bloc fonctionnel est identique à E_SR. E_SR et E_RS sont tous les deux mis en œuvre pour la cohérence avec les types SR et RS de la CEI 61131-3, bien qu'il n'y ait aucune «dominance» d'événements comme il y en aurait pour des entrées R et S commandées par niveau. | | |
| **15** | **Bistable D (Verrou de données)** | |
| |  |  |
| | | ALGORITHM LATCH IN ST:<br>  Q:= D;<br>END_ALGORITHM |
| **16** | **Détection booléenne de front montant** | |
| |  |  |
| **17** | **Détection booléenne de front descendant** | |
| |  |  |





**表A.1（6个，共7个）**

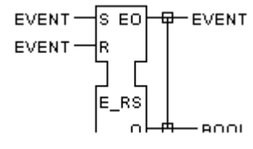
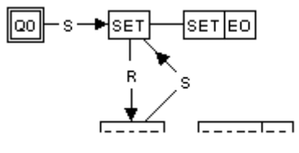
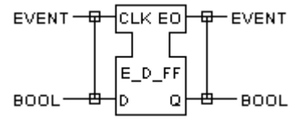
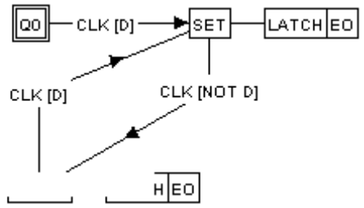
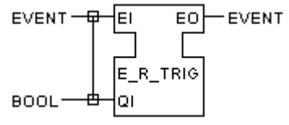
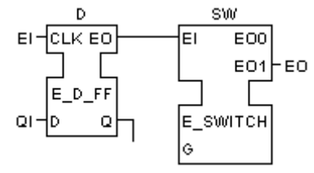
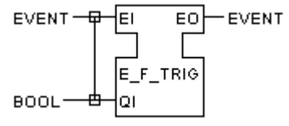
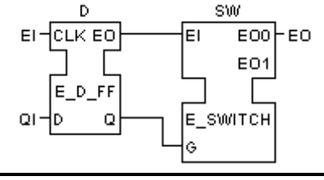
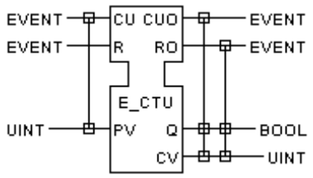
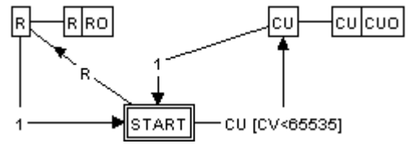
| No. | Description | |
|---|---|---|
| | **Interface** | **Algorithmes/Service** |
| **14** | **事件驱动双稳态** | |
| |  |  |
| 输出Q在输入S上出现事件后设置为1(TRUE)，并在R条目上出现事件后重置为0(FALSE)。当Q的值改变时，输出EO上会发出一个事件。<br><br>注意此类功能块的实现与E_SR相同。E_SR和E_RS都是为了与IEC61131-3的SR和RS类型保持一致而实现的，尽管没有像R输入和S由电平控制的事件"支配性"。 | | |
| **15** | **Bistable D (Verrou de données)** | |
| |  |  |
| | | ST中的算法锁存器： |
| **16** | **布尔上升沿检测** | |
| |  |  |
| **17** | **布尔下降沿检测** | |
| |  |  |

**Tableau A.1** (7 de 7)

<table>
<tr><th>No.</th><th colspan="2">Description</th></tr>
<tr><th colspan="2">Interface</th><th>Séquences ECC/Algorithmes/Service</th></tr>
<tr><td>18</td><td colspan="2">Compteur progressif événementiel</td></tr>
<tr><td colspan="2"> 
</td><td> 
</td></tr>
<tr><td colspan="2">ALGORITHM R IN ST: (* Reset *)<br>  CV:= 0;<br>  Q:= 0;<br>END_ALGORITHM</td><td>ALGORITHM CU IN ST: (* Count Up *)<br>  CV:= CV + 1;<br>  Q:= (CV >= PV);<br>END_ALGORITHM</td></tr>
</table>

Des notations graphiques raccourcies peuvent remplacer les blocs E_SPLIT et E_MERGE définis dans le Tableau A.1. Par exemple, la représentation (implicite) raccourcie montrée à la Figure A.1b est l'équivalent de la représentation explicite de la Figure A.1a.

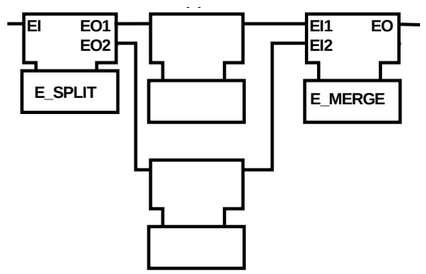



**Figure A.1a – Représentation explicite**

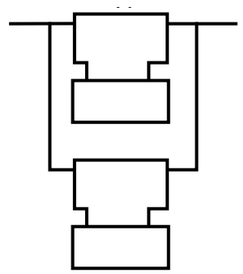


**Figure A.1b – Représentation implicite**

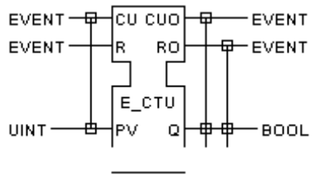
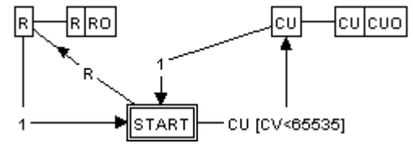
NOTE Les détails non pertinents ont été supprimés de la figure ci-dessus.

**Figure A.1 – Division et fusion d'événements**





**表A.1（7个中的7个）**

<table>
<tr><th>No.</th><th colspan="2">Description</th></tr>
<tr><th colspan="2">Interface</th><th>Séquences ECC/Algorithmes/Service</th></tr>
<tr><td>18</td><td colspan="2">Compteur progressif événementiel</td></tr>
<tr><td colspan="2"> 
</td><td> 
</td></tr>
<tr><td colspan="2">算法R安装：（*重置*）<br>  CV:= 0;<br>  Q:= 0;<br>END_ALGORITHM</td><td>ST中的算法CU：（*向上计数*）<br>  CV:= CV + 1;<br>  Q:= (CV >= PV);<br>END_ALGORITHM</td></tr>
</table>

缩短的图形符号可以替换表A.1中定义的E_SPLIT和E_MERGE块。例如，图A.1b所示的缩短（隐式）表示等效于图A.1a中的显式表示。

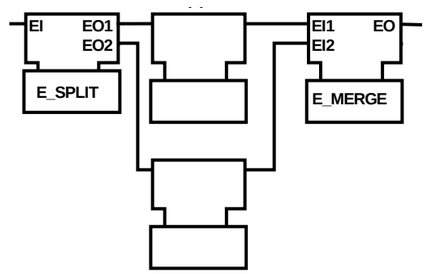
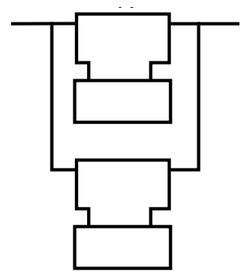



**Figure A.1a**



**n implicite**

注意不相关的细节已从上图中删除。

**图A.1 拆分和合并事件**

# Annexe B
(normative)

# Syntaxe textuelle

## B.1 Technique de spécification d'une syntaxe

Les constructions textuelles de l'Annexe B sont spécifiées en termes de *syntaxe*, qui spécifie les combinaisons admissibles de symboles qui peuvent être utilisées pour définir un programme; et d'un jeu de *sémantiques*, qui spécifient les significations des combinaisons de symboles définies par la syntaxe.

Une **syntaxe** est définie par un ensemble de *symboles terminaux* devant être utilisés pour la spécification du programme; un ensemble de *symboles non terminaux* définis en termes de symboles terminaux; et un ensemble de *règles de production* spécifiant ces définitions.

Les **symboles terminaux** pour des spécifications textuelles d'entités définies dans la présente norme sont constitués de combinaisons de caractères parmi l'ensemble de caractères donnés dans le Tableau 2 – Rangée 00 de la table de «Compatibilité Latin de Base et idéogrammes CJK» liée à l'Article 33 défini dans l'ISO/CEI 10646:2003.

Pour les besoins de la présente norme, les symboles textuels terminaux sont constitués d'une chaîne appropriée de caractères placée entre des apostrophes ou des guillemets appariés. Par exemple, un symbole terminal représenté par la chaîne de caractères ABC peut être représenté par `"ABC"` ou `'ABC'`.

Ceci permet la représentation de chaînes contenant soit des apostrophes, soit des guillemets; par exemple, un symbole terminal constitué d'un guillemet lui-même serait représenté par `'"'`

Un symbole terminal spécial utilisé dans la syntaxe est la «chaîne vide», c'est-à-dire, une chaîne ne contenant aucun caractère. Ceci est représenté par le symbole terminal `NIL`.

Les symboles textuels **non terminaux** sont représentés par des chaînes en lettres minuscules, des nombres et le caractère de soulignement (_), commençant par une lettre minuscule. Par exemple, les chaînes `nonterm1` et `non_term_2` sont des symboles non terminaux valides, alors que les chaînes `3nonterm` et `_nonterm4` ne le sont pas.

Les **règles de production** données dans la présente norme forment une *grammaire étendue* dans laquelle chaque règle a la forme

```
non_terminal_symbol::= extended_structure
```

Cette règle peut être lue comme:

«Un `non_terminal_symbol` peut consister en une `extended_structure`.»

Les structures étendues peuvent être construites selon les règles suivantes:

a) La chaîne vide, `NIL`, est une structure étendue.
b) Un symbole terminal est une structure étendue.
c) Un symbole non terminal est une structure étendue.
d) Si S est une structure étendue, alors les expressions suivantes sont aussi des structures étendues:



61499-1 © CEI:2012 – 189 –

# Annexe B
(normative)

# Syntaxe textuelle

## 指定语法的技术

附件B中的文本结构是根据语法规定的，它规定了可用于定义程序的允许的符号组合；和一组语义，它们指定了语法定义的符号组合的含义。

语法由一组用于程序规范的终端符号定义；一组根据终结符号定义的非终结符号；以及一组指定这些定义的生产规则。

本标准中定义的实体的文本规范的终端符号由表2中给出的字符组中的字符组合组成–与ISOIEC10646:2003中定义的第33条相关联的"基本拉丁文兼容性和CJK表意文字"表的第00行.

就本标准而言，终端文本符号由包含在成对撇号或引号中的适当字符串组成。例如，字符串ABC表示的终结符可以用"ABC"或'ABC'来表示。

这允许表示包含撇号或双引号的字符串；例如，由引号本身组成的终端符号将由'"'表示

语法中使用的特殊终端符号是"空字符串"，即不包含字符的字符串。这由终端符号NIL表示。

非终结符文本符号由小写字母字符串、数字和下划线字符(_)表示，以小写字母开头。例如，字符串nonterm1和non_term_2是有效的非终结符，而字符串3nonterm和_nonterm4不是。

**本标准中给出的产生式规则形成了一个扩展语法，其中每个规则都有以下形式**

```
al_symbol::= extended_structure
```

这条规则可以理解为：

"非终端符号可能由扩展结构组成。"

可以根据以下规则构建扩展结构：

a)空字符串NIL是一个扩展结构。
b)终端符号是一个扩展结构。
c)非终结符号是扩展结构。
d)如果S是扩展结构，则以下表达式也是扩展结构：

- (S), signifiant S lui-même.
- {S}, *fermeture*, signifiant zéro, une ou plusieurs concaténations de S.
- [S], *option*, signifiant zéro ou une occurrence de S.

e) Si S1 et S2 sont des structures étendues, alors les expressions suivantes sont des structures étendues:

- S1 | S2, *union*, signifiant un choix de S1 ou S2.
- S1 S2, *concaténation*, signifiant S1 suivi de S2.

f) La concaténation *précède* l'union, c'est-à-dire que S1 | S2 S3 équivaut à S1 | (S2 S3), et S1 S2 | S3 équivaut à (S1 S2) | S3.

Les **sémantiques** sont définies dans la présente norme par du texte approprié en langage naturel, accompagnant les règles de production, qui référencent les descriptions données dans les articles appropriés. Les options normalisées disponibles pour l'utilisateur et le vendeur sont spécifiées dans ces sémantiques.

Dans certains cas, il est plus commode d'intégrer les informations sémantiques dans une structure étendue. Dans de tels cas, ces informations sont délimitées par des parenthèses en chevron appariées, par exemple <informations sémantiques>.

## B.2 Spécification des types de bloc fonctionnel et de sous-application

### B.2.1 Spécification des types de bloc fonctionnel

La syntaxe définie en B.2.1 peut être utilisée pour la spécification textuelle des *types de bloc fonctionnel* selon les règles données dans les Articles 5 et 6 de la présente norme.

**SYNTAXE:**

```
fb_type_declaration::=
     'FUNCTION_BLOCK' fb_type_name
     fb_interface_list
     [fb_internal_variable_list] <only for basic FB>
     [fb_instance_list]      <only for composite FB>
     [plug_list]
     [socket_list]
     [fb_connection_list]     <only for composite FB>
     [fb_ecc_declaration]     <only for basic FB>
     {fb_algorithm_declaration}  <only for basic FB>
     [fb_service_declaration]
     'END_FUNCTION_BLOCK'


fb_interface_list::=
     [event_input_list]
     [event_output_list]
     [input_variable_list]
     [output_variable_list]


event_input_list::=
     'EVENT_INPUT'
     {event_input_declaration}
     'END_EVENT'


event_output_list::=
     'EVENT_OUTPUT'
     {event_output_declaration}
     'END_EVENT'
```





-

- {S ，闭包，意思是零，一个或多个S的串联。

  [S]，option，表示S出现零次或一次。

e)如果S1和S2是扩展结构，则以下表达式是扩展结构：

- S1S2，并集，意思是选择S1或S2。

f)连接先于联合，即S1S2S3等于S1(S2S3)，并且S1S2S3等于(S1S2)S3。

本标准中的语义由适当的自然语言文本定义，伴随生产规则，这些规则引用了适当条款中给出的描述。在这些语义中指定了用户和供应商可用的标准化选项。

在某些情况下，将语义信息嵌入扩展结构中会更方便。在这种情况下，此信息由成对的尖括号分隔，例如<semanticinformation>。

## B.2 指定功能块和子应用程序类型

### 指定功能块类型

根据本标准第5章和第6章给出的规则，B.2.1中定义的语法可用于功能块类型的文本规范。

**SYN**

```
fb_t
     'FUNCTION_BLOCK'fb_type_namefb_interface_list[fb_internal_variable
     _list]<仅适用于基本FB>[fb_instance_list]<仅适用于复合FB>[plug_list][s
     ocket_list][fb_connection_list]<仅适用于复合FB>[fb_ecc_declaration]<
     仅适用于基本FB>{fb_algorithm_declaration <仅适用于基本FB>[fb_servi
     ce_declaration]'END_FUNCTION_BLOCK'


fb_interface_list::=
     [event_input_list]
     [event_output_list]
     [input_variable_list]
     [output_variable_list]


event_input_list::=
     'EVENT_INPUT'
     {event_input_declaration}
     'END_EVENT'


event_output_list::=
     'EVENT_OUTPUT'
     {event_output_declaration}
     'END_EVENT'
```

```
event_input_declaration::= event_input_name [ ':' event_type ]
     ['WITH' input_variable_name {',' input_variable_name}] ';'

event_output_declaration::= event_output_name [ ':' event_type ]
     ['WITH' output_variable_name {',' output_variable_name}] ';'

input_variable_list::=
     'VAR_INPUT' {input_var_declaration ';'} 'END_VAR'

output_variable_list::=
     'VAR_OUTPUT' {output_var_declaration ';'} 'END_VAR'

fb_internal_variable_list::=
     'VAR' {internal_var_declaration ';'} 'END_VAR'

input_var_declaration::=
     input_variable_name {',' input_variable_name} ':' var_spec_init

output_var_declaration::=
     output_variable_name {',' output_variable_name} ':' var_spec_init

internal_var_declaration::=
     internal_variable_name {',' internal_variable_name}
     ':' var_spec_init

var_spec_init::= located_var_spec_init <as specified in IEC 61131-3>

fb_instance_list::= 'FBS'
     {fb_instance_definition ';'}
     'END_FBS'

fb_instance_definition::= fb_instance_name ':' fb_type_name [parameters]

plug_list::= 'PLUGS'
     {plug_name ':' adapter_type_name [parameters] ';'}
     'END_PLUGS'

socket_list::= 'SOCKETS'
     {socket_name ':' adapter_type_name [parameters] ';'}
     'END_SOCKETS'

fb_connection_list::=   <may be empty, e.g. for basic FB>
     [event_conn_list]
     [data_conn_list]
     [adapter_conn_list]

event_conn_list::=
     'EVENT_CONNECTIONS'
     {event_conn}
     'END_CONNECTIONS'

event_conn::= event_conn_source 'TO' event_conn_destination ';'


event_conn_source::= ([plug_name '.'] event_input_name)
     | ((fb_instance_name | socket_name) '.' event_output_name)

event_conn_destination::= ([plug_name '.'] event_output_name)
     | ((fb_instance_name | socket_name) '.' event_input_name)
```



61499-1 © CEI:2012 – 191 –

```
event_input_declaration::= event_input_name [ ':' event_type ]
     ['WITH' input_variable_name {',' input_variable_name}] ';'

event_output_declaration::= event_output_name [ ':' event_type ]
     ['WITH' output_variable_name {',' output_variable_name}] ';'

input_variable_list::=
     'VAR_INPUT' {input_var_declaration ';'} 'END_VAR'

output_variable_list::=
     'VAR_OUTPUT' {output_var_declaration ';'} 'END_VAR'

fb_internal_variable_list::=
     'VAR' {internal_var_declaration ';'} 'END_VAR'

input_var_declaration::=
     input_variable_name {',' input_variable_name} ':' var_spec_init

output_var_declaration::=
     output_variable_name {',' output_variable_name} ':' var_spec_init

internal_var_declaration::=
     internal_variable_name {',' internal_variable_name}
```

var_spec_init::=located_var_spec_init<按照IEC61131-3的规定>

```
fb_instance_list::= 'FBS'
     {fb_instance_definition ';'}
     'END_FBS'

fb_instance_definition::= fb_instance_name ':' fb_type_name [parameters]

plug_list::= 'PLUGS'
     {plug_name ':' adapter_type_name [parameters] ';'}
     'END_PLUGS'

socket_list::= 'SOCKETS'
     {socket_name ':' adapter_type_name [parameters] ';'}
```

fb_connection_list::=<可能为空，例如对于基本FB>[event_conn_list][data_conn_list]
[adapter_conn_list]

```
event_conn_list::=
     'EVENT_CONNECTIONS'
     {event_conn}
     'END_CONNECTIONS'

event_conn::= event_conn_source 'TO' event_conn_destination ';'


event_conn_source::= ([plug_name '.'] event_input_name)
     | ((fb_instance_name | socket_name) '.' event_output_name)

event_conn_destination::= ([plug_name '.'] event_output_name)
     | ((fb_instance_name | socket_name) '.' event_input_name)
```

```
data_conn_list::=
     'DATA_CONNECTIONS'
     {data_conn}
     'END_CONNECTIONS'

data_conn::= data_conn_source 'TO' data_conn_destination ';'


data_conn_source::= ([plug_name '.'] input_variable_name)
     | ((fb_instance_name | socket_name) '.' output_variable_name)

data_conn_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_instance_name | socket_name) '.' input_variable_name)

adapter_conn_list::=
     'ADAPTER_CONNECTIONS'
     {adapter_conn}
     'END_CONNECTIONS'

adapter_conn::=
     ((fb_instance_name '.' plug_name ) | socket_name)
     'TO' ((fb_instance_name '.' socket_name ) | plug_name) ';'

fb_ecc_declaration::=
     'EC_STATES'
     {ec_state}    <first state is initial state>
     'END_STATES'
     'EC_TRANSITIONS'
     {ec_transition}
     'END_TRANSITIONS'

ec_state::= ec_state_name
     [':' ec_action {',' ec_action}] ';'

ec_action::= algorithm_name | ('->' ec_action_output)
     | (algorithm_name '->' ec_action_output)


ec_action_output:= ([plug_name '.'] event_output_name)
     | (socket_name '.' event_input_name)

ec_transition::=
     ec_state_name
     'TO' ec_state_name
     ':=' ec_transition_condition ';'

ec_transition_condition::= '1'
     | ec_transition_event | '[' guard_condition ']'
     | ec_transition_event '[' guard_condition ']'


ec_transition_event::= ([plug_name '.'] event_input_name)
     | (socket_name '.' event_output_name)

guard_condition::= expression <over ec_expression_operand elements>
     <as defined in IEC 61131-3>
     <Shall evaluate to a BOOL value>

ec_expression_operand::=
     ([(plug_name | socket_name) '.'] input_variable_name)
     | ([(plug_name | socket_name) '.'] output_variable_name)
     | internal_variable_name
     | constant
```



– 192 – 61499-1 © CEI:2012


```
data_conn_list::=
     'DATA_CONNECTIONS'
     {data_conn}
     'END_CONNECTIONS'

data_conn::= data_conn_source 'TO' data_conn_destination ';'


data_conn_source::= ([plug_name '.'] input_variable_name)
     | ((fb_instance_name | socket_name) '.' output_variable_name)

data_conn_destination::= ([plug_name '.'] output_variable_name)
     | ((fb_instance_name | socket_name) '.' input_variable_name)

adapter_conn_list::=
     'ADAPTER_CONNECTIONS'
     {adapter_conn}
     'END_CONNECTIONS'

adapter_conn::=
     ((fb_instance_name '.' plug_name ) | socket_name)
                                                  lug_name) ';'
```


fb_ecc_declaration::='EC_STATES'{ec_state <第一个状态是初始状态>'EN
D_STATES''EC_TRANSITIONS'{ec_transition 'END_TRANSITIONS'


```
ec_state::= ec_state_name
     [':' ec_action {',' ec_action}] ';'

ec_action::= algorithm_name | ('->' ec_action_output)
     | (algorithm_name '->' ec_action_output)


ec_action_output:= ([plug_name '.'] event_output_name)
     | (socket_name '.' event_input_name)

ec_transition::=
     ec_state_name
     'TO' ec_state_name
     ':=' ec_transition_condition ';'

ec_transition_condition::= '1'
     | ec_transition_event | '[' guard_condition ']'
     | ec_transition_event '[' guard_condition ']'


ec_transition_event::= ([plug_name '.'] event_input_name)
```


guard_condition::=expression<overec_expression_operandelements><asdefinedinIEC61131-3><S
hallevaluatetoaBOOLvalue>


```
ec_expression_operand::=
     ([(plug_name | socket_name) '.'] input_variable_name)
     | ([(plug_name | socket_name) '.'] output_variable_name)
     | internal_variable_name
     | constant
```

```
fb_algorithm_declaration::=
     'ALGORITHM' algorithm_name 'IN' language_type ':'
     [temp_var_decls]
     algorithm_body
     'END_ALGORITHM'

temp_var_decls::= <as defined in IEC 61131-3>

algorithm_body::= <as defined in compliant standards>

fb_service_declaration::=
     'SERVICE' service_interface_name '//' service_interface_name
     {service_sequence}
     'END_SERVICE'

service_interface_name::= fb_type_name | 'RESOURCE'

service_sequence::=
     'SEQUENCE' sequence_name
     {service_transaction ';'}
     'END_SEQUENCE'

service_transaction::=
     [input_service_primitive] '->' output_service_primitive
     {'->' output_service_primitive}

input_service_primitive::= service_interface_name '.'
     ([plug_name '.'] event_input_name
      | socket_name '.' event_output_name)
     ['+' | '-']
     '(' [input_variable_name {',' input_variable_name}] ')'

output_service_primitive::= service_interface_name '.' ('NULL' |
     ([plug_name '.'] event_output_name
      | socket_name '.' event_input_name)
     ['+' | '-']
     '(' [output_variable_name {',' output_variable_name}] ')')

algorithm_name::= identifier

ec_state_name::= identifier

event_input_name::= identifier

event_output_name::= identifier

event_type::= identifier

fb_instance_name::= identifier

fb_type_name::= identifier

input_variable_name::= identifier

internal_variable_name::= identifier

language_type::= identifier

output_variable_name::= identifier
```



61499-1 © CEI:2012 – 193 –

```
fb_algorithm_declaration::=
     'ALGORITHM' algorithm_name 'IN' language_type ':'
     [temp_var_decls]
     algorithm_body
```

temp_var_decls::=<如IEC61131-3中定义>

algorithm_body::=<在兼容标准中定义>

```
fb_service_declaration::=
     'SERVICE' service_interface_name '//' service_interface_name
     {service_sequence}
     'END_SERVICE'

service_interface_name::= fb_type_name | 'RESOURCE'

service_sequence::=
     'SEQUENCE' sequence_name
     {service_transaction ';'}
     'END_SEQUENCE'

service_transaction::=
     [input_service_primitive] '->' output_service_primitive
     {'->' output_service_primitive}

input_service_primitive::= service_interface_name '.'
     ([plug_name '.'] event_input_name
      | socket_name '.' event_output_name)
     ['+' | '-']
     '(' [input_variable_name {',' input_variable_name}] ')'

output_service_primitive::= service_interface_name '.' ('NULL' |
     ([plug_name '.'] event_output_name
      | socket_name '.' event_input_name)
     ['+' | '-']
     '(' [output_variable_name {',' output_variable_name}] ')')

algorithm_name::= identifier

ec_state_name::= identifier

event_input_name::= identifier

event_output_name::= identifier

event_type::= identifier

fb_instance_name::= identifier

fb_type_name::= identifier

input_variable_name::= identifier

internal_variable_name::= identifier

language_type::= identifier

output_variable_name::= identifier
```

```
plug_name::= identifier

sequence_name::= identifier

socket_name::= identifier
```

### B.2.2 Spécification des types de sous-application

La syntaxe définie dans ce paragraphe peut être utilisée pour la spécification textuelle des *types* de *sous-application* selon les règles données en 5.4.1.

La production donnée en B.2.1 s'applique aussi au présent paragraphe.

**SYNTAXE:**

```
subapplication_type_declaration::=
     'SUBAPPLICATION' subapp_type_name
          subapp_interface_list
          [fb_instance_list]
          [subapp_instance_list]
          [plug_list]
          [socket_list]
          [subapp_connection_list]
     'END_SUBAPPLICATION'

subapp_interface_list::=
     [subapp_event_input_list]
     [subapp_event_output_list]
     [input_variable_list]
     [output_variable_list]

subapp_event_input_list::=
     'EVENT_INPUT'
     {subapp_event_input_declaration}
     'END_EVENT'

subapp_event_output_list::=
     'EVENT_OUTPUT'
     {subapp_event_output_declaration}
     'END_EVENT'

subapp_event_input_declaration::=
     event_input_name [ ':' event_type ] ';'

subapp_event_output_declaration::=
     event_output_name [ ':' event_type ] ';'

subapp_instance_list::= 'SUBAPPS'
     {subapp_instance_definition ';'}
     'END_SUBAPPS'

subapp_instance_definition::= subapp_instance_name ':' subapp_type_name

subapp_connection_list::=
     [subapp_event_conn_list]
     [subapp_data_conn_list]
     [adapter_conn_list]

subapp_event_conn_list::=
     'EVENT_CONNECTIONS'
     {subapp_event_conn}
     'END_CONNECTIONS'
```



– 194 – 61499-1 © CEI:2012

```
plug_name::= identifier

sequence_name::= identifier

socket_
```

**指定子应用程序类型**

根据5.4.1中给出的规则，本子条款中定义的语法可用于子应用类型的文本规范。

B.2.1中给出的产生式也适用于本段。

**SYNTAXE:**

```
subapplication_type_declaration::=
     'SUBAPPLICATION' subapp_type_name
          subapp_interface_list
          [fb_instance_list]
          [subapp_instance_list]
          [plug_list]
          [socket_list]
          [subapp_connection_list]
     'END_SUBAPPLICATION'

subapp_interface_list::=
     [subapp_event_input_list]
     [subapp_event_output_list]
     [input_variable_list]
     [output_variable_list]

subapp_event_input_list::=
     'EVENT_INPUT'
     {subapp_event_input_declaration}
     'END_EVENT'

subapp_event_output_list::=
     'EVENT_OUTPUT'
     {subapp_event_output_declaration}
     'END_EVENT'

subapp_event_input_declaration::=
     event_input_name [ ':' event_type ] ';'

subapp_event_output_declaration::=
     event_output_name [ ':' event_type ] ';'

subapp_instance_list::= 'SUBAPPS'
     {subapp_instance_definition ';'}
     'END_SUBAPPS'

subapp_instance_definition::= subapp_instance_name ':' subapp_type_name

subapp_connection_list::=
     [subapp_event_conn_list]
     [subapp_data_conn_list]
     [adapter_conn_list]

subapp_event_conn_list::=
     'EVENT_CONNECTIONS'
     {subapp_event_conn}
     'END_CONNECTIONS'
```

```
subapp_event_conn::= subapp_event_source ‘TO’ subapp_event_destination ‘;’

subapp_event_source:: = ([plug_name ‘.’] event_input_name)
     | ((fb_subapp_name | socket_name) ‘.’ event_output_name

subapp_event_destination::= ([plug_name ‘.’] event_output_name)
     | ((fb_subapp_name | socket_name) ‘.’ event_input_name)

fb_subapp_name::= fb_instance_name | subapp_instance_name

subapp_data_conn_list::=
     ‘DATA_CONNECTIONS’
     {subapp_data_conn}
     ‘END_CONNECTIONS’

subapp_data_conn::= subapp_data_source ‘TO’ subapp_data_destination ‘;’

subapp_data_source::= ([plug_name ‘.’] input_variable_name)
     | ((fb_subapp_name | socket_name) ‘.’ output_variable_name)

subapp_data_destination::= ([plug_name ‘.’] output_variable_name)
     | ((fb_subapp_name | socket_name) ‘.’ input_variable_name)

subapp_type_name::= identifier

subapp_instance_name::= identifier
```

## B.3 Eléments de configuration

La syntaxe définie dans cet article peut être utilisée pour la spécification textuelle des *types de ressource*, *types d'équipement*, *types de segment*, *applications*, et *configurations de système* selon les règles données à l'Article 7.

Les productions données à l'Article B.2 s'appliquent aussi à cet article.

**SYNTAXE:**

```
application_configuration::=
     ‘APPLICATION’ application_name
     [fb_instance_list]
     [subapp_instance_list]
     [subapp_connection_list]
     ‘END_APPLICATION’

system_configuration::= ‘SYSTEM’ system_name
      {application_configuration}
      device_configuration
      {device_configuration}
      [mappings]
      [segments]
      [links]
     ‘END_SYSTEM’

segments::= ‘SEGMENTS’
      segment
      {segment}
     ‘END_SEGMENTS’

segment::= segment_name ‘:’ segment_type_name [parameters] ‘;’
```



61499-1 © CEI:2012 – 195 –

```
subapp_event_conn::= subapp_event_source ‘TO’ subapp_event_destination ‘;’

subapp_event_source:: = ([plug_name ‘.’] event_input_name)
     | ((fb_subapp_name | socket_name) ‘.’ event_output_name

subapp_event_destination::= ([plug_name ‘.’] event_output_name)
     | ((fb_subapp_name | socket_name) ‘.’ event_input_name)

fb_subapp_name::= fb_instance_name | subapp_instance_name

subapp_data_conn_list::=
     ‘DATA_CONNECTIONS’
     {subapp_data_conn}
     ‘END_CONNECTIONS’

subapp_data_conn::= subapp_data_source ‘TO’ subapp_data_destination ‘;’

subapp_data_source::= ([plug_name ‘.’] input_variable_name)
     | ((fb_subapp_name | socket_name) ‘.’ output_variable_name)

subapp_data_destination::= ([plug_name ‘.’] output_variable_name)
     | ((fb_subapp_name | socket_name) ‘.’ input_variable_name)

subapp_type_name::= identifier

subapp                          er
```

## 配置项

根据第7章中给出的规则，本节中定义的语法可用于资源类型、设备类型、段类型、应用程序和系统配置的文本规范。

B.2条中给出的产生式也适用于本条。

**SYNTAXE:**

```
application_configuration::=
     ‘APPLICATION’ application_name
     [fb_instance_list]
     [subapp_instance_list]
     [subapp_connection_list]
     ‘END_APPLICATION’

system_configuration::= ‘SYSTEM’ system_name
      {application_configuration}
      device_configuration
      {device_configuration}
      [mappings]
      [segments]
      [links]
     ‘END_SYSTEM’

segments::= ‘SEGMENTS’
      segment
      {segment}
     ‘END_SEGMENTS’

segment::= segment_name ‘:’ segment_type_name [parameters] ‘;’
```

```
links::= 'LINKS'
      link
      {link}
     'END_LINKS'

link::= resource_hierarchy '=>' segment_name [parameters] ';'

parameters::= '(' parameter {',' parameter} ')'

parameter::= parameter_name ':='
     (constant | enumerated_value | array_initialization |
     structure_initialization) ';'
     <as defined in IEC 61131-3>

device_configuration::=
     'DEVICE' device_name ':' device_type_name [parameters]
     [resource_type_list]
     {resource_configuration}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE'

resource_type_list::= 'RESOURCE_TYPES'
     {resource_type_name ';'}
     'END_RESOURCE_TYPES'

resource_configuration::=
     'RESOURCE' resource_instance_name ':' resource_type_name [parameters]
     [fb_type_list]
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

fb_type_list::= 'FB_TYPES' {fb_type_name ';'} 'END_FB_TYPES'

config_connection_list::=
     [config_event_conn_list]
     [config_data_conn_list]
     [config_adapter_conn_list]

config_event_conn_list::= 'EVENT_CONNECTIONS'
     {config_event_conn}
     'END_CONNECTIONS'

config_event_conn::= fb_instance_name '.' event_output_name
     'TO' fb_instance_name '.' event_input_name ';'

config_data_conn_list::= 'DATA_CONNECTIONS'
     {config_data_conn}
     'END_CONNECTIONS'

config_data_conn::=
     (fb_instance_name '.' output_variable_name | input_variable_name)
     'TO'
     (fb_instance_name | resource_instance_name)'.' Input_variable_name ';'
     <resource_instance_name only applies to connections within device_type or
     device_configuration declarations>

config_adapter_conn_list::= 'ADAPTER_CONNECTIONS'
     {config_adapter_conn}
     'END_CONNECTIONS'
```



– 196 – 61499-1 © CEI:2012

```
links::= 'LINKS'
      link
      {link}
     'END_LINKS'

link::= resource_hierarchy '=>' segment_name [parameters] ';'

parameters::= '(' parameter {',' parameter} ')'

para
     (constantenumerated_valuearray_initializationstructure_initialization)';'<在IEC
     61131-3中定义>

device_configuration::=
     'DEVICE' device_name ':' device_type_name [parameters]
     [resource_type_list]
     {resource_configuration}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE'

resource_type_list::= 'RESOURCE_TYPES'
     {resource_type_name ';'}
     'END_RESOURCE_TYPES'

resource_configuration::=
     'RESOURCE' resource_instance_name ':' resource_type_name [parameters]
     [fb_type_list]
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

fb_type_list::= 'FB_TYPES' {fb_type_name ';'} 'END_FB_TYPES'

config_connection_list::=
     [config_event_conn_list]
     [config_data_conn_list]
     [config_adapter_conn_list]

config_event_conn_list::= 'EVENT_CONNECTIONS'
     {config_event_conn}
     'END_CONNECTIONS'

config_event_conn::= fb_instance_name '.' event_output_name
     'TO' fb_instance_name '.' event_input_name ';'

config_data_conn_list::= 'DATA_CONNECTIONS'
     {config_data_conn}
     'END_CONNECTIONS'

conf
     (fb_instance_name'.'output_variable_nameinput_variable_name)'TO'(fb_instance_nameresource_instance_
     name)'.'Input_variable_name';'<resource_instance_name仅适用于device_type或device_configuration声明中
     的连接>

config_adapter_conn_list::= 'ADAPTER_CONNECTIONS'
     {config_adapter_conn}
     'END_CONNECTIONS'
```

```
config_adapter_conn::= fb_instance_name '.' plug_name
     'TO' fb_instance_name '.' socket_name ';'

fb_instance_reference::= [app_hierarchy_name] fb_instance_name

app_hierarchy_name:= application_name '.'{subapp_instance_name '.'}

device_type_specification::=
     'DEVICE_TYPE' device_type_name
     [input_variable_list]
     [resource_type_list] <if not given, defined by resource instances>
     {resource_instance}
     [fb_instance_list]
     [config_connection_list]
     'END_DEVICE_TYPE'

resource_instance::=
     'RESOURCE' resource_instance_name ':' resource_type_name
     [fb_instance_list]
     [config_connection_list]
     'END_RESOURCE'

resource_type_specification::= 'RESOURCE_TYPE' resource_type_name
     [input_variable_list]
     [fb_type_list] <if not given, defined by function block instances>
     [fb_instance_list]
     config_connection_list
     'END_RESOURCE_TYPE'

segment_type_specification::= 'SEGMENT_TYPE' segment_type_name
     {parameter_declaration}
     'END_SEGMENT_TYPE'

parameter_declaration:= parameter_name ':' var_spec_init ';'

mappings::= 'MAPPINGS' mapping {mapping} 'END_MAPPINGS'

mapping::= fb_instance_reference 'ON' fb_resource_reference ';'

fb_resource_reference::= resource_hierarchy ['.' Fb_instance_name]
     <Lorsque l'élément facultatif ['.' Fb_instance_name] n'est pas donné, le
     nom d'instance du FB dans la ressource est le même que son nom d'instance
     dans le fb_instance_reference correspondant du mapping.>

resource_hierarchy::= device_name ['.' Resource_instance_name]

segment_name::= identifier

segment_type_name::= identifier

parameter_name::= identifier

system_name::= identifier

device_name::= identifier

device_type_name::= identifier

application_name::= identifier
```



61499-1 © CEI:2012 – 197 –

```
config_adapter_conn::= fb_instance_name '.' plug_name
     'TO' fb_instance_name '.' socket_name ';'

fb_instance_reference::= [app_hierarchy_name] fb_instance_name

app_hierarchy_name:= application_name '.'{subapp_instance_name '.'}

devi
     'DEVICE_TYPE'device_type_name[input_variable_list][resource_type_list]<如果没有给出，由资源
     实例定义>{resource_instance [fb_instance_list][config_connection_list]'END_DEVICE_TYPE'

resource_instance::=
     'RESOURCE' resource_instance_name ':' resource_type_name
     [fb_instance_list]
     [config_connection_list]

resource_type_specification::='RESOURCE_TYPE'resource_type_name[input_variable_list][fb_type_list]<
如果没有给出，由功能块实例定义>[fb_instance_list]config_connection_list'END_RESOURCE_TYPE'

segment_type_specification::= 'SEGMENT_TYPE' segment_type_name
     {parameter_declaration}
     'END_SEGMENT_TYPE'

parameter_declaration:= parameter_name ':' var_spec_init ';'

mappings::= 'MAPPINGS' mapping {mapping} 'END_MAPPINGS'

mapping::= fb_instance_reference 'ON' fb_resource_reference ';'

fb_r
     <Lorsquel'élémentfacultatif['.'Fb_instance_name]n'estpasdonné lenomd'instanceduFBdanslaressourceestle
     memequesonnomd'instancedanslefb_instance_referenceassociateantdumapping.>

resource_hierarchy::= device_name ['.' Resource_instance_name]

segment_name::= identifier

segment_type_name::= identifier

parameter_name::= identifier

system_name::= identifier

device_name::= identifier

device_type_name::= identifier

application_name::= identifier
```

```
resource_instance_name::= identifier

resource_type_name::= identifier
```

## B.4 Eléments communs

Lorsque des productions syntaxiques ne sont pas données pour les symboles non terminaux dans l'Annexe B, les productions syntaxiques et la sémantique correspondante données dans l'Annexe B de la CEI 61131-3:2003 doivent s'appliquer.

## B.5 Représentations prises en charge pour les commandes de gestion

La syntaxe définie dans cet article est référencée dans le Tableau 8.

**SYNTAXE:**

```
data_type_list::= 'DATA_TYPES' {data_type_name ';'} 'END_DATA_TYPES'

connection_definition::=
     connection_start_point ' ' connection_end_point

connection_start_point::= fb_instance_reference '.' attachment_point

connection_end_points::=
     connection_end_point {',' connection_end_point}

connection_end_point::= fb_instance_reference '.' attachment_point

attachment_point::= identifier

referenced_parameter::=
     [(resource_instance_name | fb_instance_name)'.'] parameter
     <resource_instance_name se réfère à une ressource localisée dans le même
     équipement que le bloc MANAGER défini en 6.3.2>
     <fb_instance_name se réfère à un FB contenu dans le même équipement ou la
     même ressource que le bloc <MANAGER> >
     <si aucun nom d'instance de ressource ou de FB n'est donné, le paramètre
     se réfère à un paramètre du équipement ou de la ressource contenant le
     bloc MANAGER>

parameter_reference::=
     [(resource_instance_name | fb_instance_name)'.'] parameter_name
     <see above for semantics>

all_data_types::= 'ALL_DATA_TYPES'

all_fb_types::= 'ALL_FB_TYPES'

fb_status::= 'IDLE' | 'RUNNING' | 'STOPPED' | 'KILLED'
```

## B.6 Types de données étiquetés

La syntaxe définie ci-dessous doit être utilisée pour l'attribution d'étiquettes telles que définies dans l'ISO/CEI 8824-1 à des types de données dérivés comme spécifié dans l'Annexe B et l'Annexe E. Comme défini dans l'ISO/CEI 8824-1, les étiquettes de classe `APPLICATION` et `PRIVATE` doivent être utilisées, sauf pour les types devant être utilisés seulement dans l'étiquetage spécifique à un contexte.



– 198 – 61499-1 © CEI:2012

resource_instance_name::= identifier

resource_type_name::= identifier

如果附录B中的非终结符号没有给出句法产生式，则应适用IEC61131-3:2003附录B中给出的句法产生式和相应的语义。

### 管理命令支持的表示

本文中定义的语法在表8中引用。

**SYNTAXE:**

data_type_list::= 'DATA_TYPES' {data_type_name ';'} 'END_DATA_TYPES'

connection_definition::=
     connection_start_point ' ' connection_end_point

connection_start_point::= fb_instance_reference '.' attachment_point

connection_end_points::=
     connection_end_point {',' connection_end_point}

connection_end_point::= fb_instance_reference '.' attachment_point

attachment_point::= identifier

refe
     [(resource_instance_namefb_instance_name)'.']参数<resource_instance_name指与6.3.2中定义的MANAGER块位于同一设备中的资源><fb_instance_name指与<MANAGER包含在同一设备或同一资源中的FB>块><如果没有给出资源或FB实例名称，则该参数是指包含MANAGER块的设备或资源的参数>

para
     [(resource_instance_namefb_instance_name)'.']parameter_name<语义见上文>

all_data_types::= 'ALL_DATA_TYPES'

all_fb_types::= 'ALL_FB_TYPES'

fb_sta 'STOPPED' | 'KILLED'

### 带标签的数据类型

下面定义的语法应用于将ISOIEC8824-1中定义的标签分配给附录B和附录E中指定的派生数据类型。按照ISOIEC8824-1中的定义，应使用APPLICATION和PRIVATE类标签，除了对于仅在特定于上下文的标记中使用的类型。

**SYNTAXE:**

```
tagged_type_declaration::=
     'TYPE'
     asn1_tag type_declaration ';'
     {asn1_tag type_declaration ';'}
     'END_TYPE'

asn1_tag::= '[' ['APPLICATION' | 'PRIVATE'] (integer | hex_integer) ']'
```

## B.7 Types d'adaptateurs d'interfaces

Voir 5.5 pour la sémantique associée à la syntaxe suivante.

**SYNTAXE:**

```
adapter_type_declaration::=
     'ADAPTER' adapter_type_name
          fb_interface_list
          [fb_service_declaration]
     'END_ADAPTER'

adapter_type_name::= identifier
```



61499-1 © CEI:2012 – 199 –

**SYNTAXE:**


```
tagged_type_declaration::=
     'TYPE'
     asn1_tag type_declaration ';'
     {asn1_tag type_declaration ';'}
     'END_TYPE'

asn1_tag::= '[' ['APPLICATION' | 'PRIVATE'] (integer | hex_integer) ']'
```


有关与以下语法相关的语义，请参见5.5。

**SYNTAXE:**


```
adapter_type_declaration::=
     'ADAPTER' adapter_type_name
          fb_interface_list
          [fb_service_declaration]
     'END_ADAPTER'

adapter_type_name::= identifier
```

# Annexe C
(informative)

# Modèles d'objets

## C.1 Notation du modèle

L'Annexe C donne des modèles d'objets pour certaines des classes qui peuvent être utilisées dans les systèmes Engineering Support Systems (ESS) pour prendre en charge la conception, la mise en œuvre, la mise en service et le fonctionnement des systèmes de mesure et commande dans les processus industriels (les IPMCS) construits selon l'architecture définie dans la présente norme.

La notation utilisée dans l'Annexe C est le langage de modélisation unifié (UML). Des références à une documentation extensive de cette notation peuvent être consultées sur internet à l'URL (localisateur uniforme de ressource «Uniform Resource Locator») **http://www.omg.org/uml/**.

## C.2 Modèle des systèmes ESS

### C.2.1 Vue d'ensemble du système ESS

La Figure C.1 présente une vue d'ensemble des classes principales dans le système ESS (Engineering Support System «système de support d'ingénierie») pour un système de mesure et commande dans les processus industriels (IPMCS) et leur correspondance avec les classes d'objets dans l'IPMCS. Les descriptions des classes dans la Figure C.1 sont données dans le Tableau C.1.

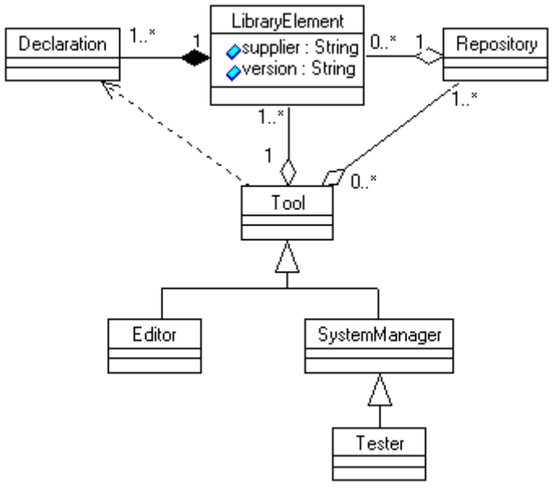



**Figure C.1 – Vue d'ensemble du système ESS**



– 200 – 61499-1 © CEI:2012

# Annexe C
(informative)

# Modèles d'objets

**型号符号**

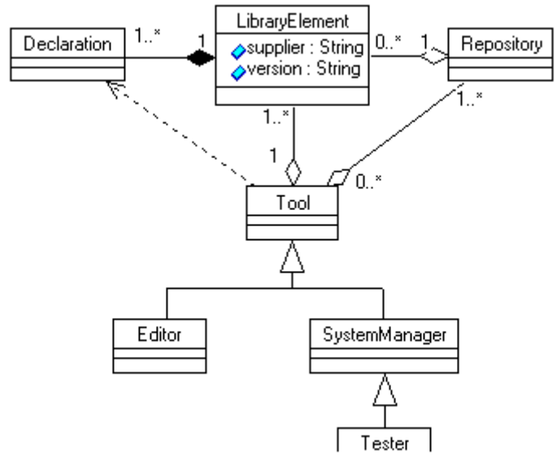
附录C给出了一些可用于工程支持系统(ESS)的类的对象模型，以支持测量系统的设计、实施、调试和操作，以及根据定义的架构构建的工业过程控制(IPMCS)在本标准中。

附录C中使用的符号是统一建模语言(UML)。可以在Internet上的URL（统一资源定位器）http：www.omg.oruml上找到对该表示法的大量文档的引用。

## C.2 ESS系统模型

**ESS系统概述**

图C.1概述了工业过程测量和控制系统(IPMCS)的工程支持系统(ESS)中的主要类别及其与IPMCS中对象类别的对应关系。图C.1中的类描述在表C.1中给出。




**图C.1 ESS系统概述**

**Tableau C.1 – Descriptions des classes ESS**

| | |
|---|---|
| **Declaration** | Il s'agit d'une superclasse abstraite pour les *déclarations*. |
| **Editor** | Les instances de cette classe fournissent les fonctions d'édition sur les *déclarations* nécessaires pour prendre en charge le cas d'utilisation EDIT. |
| **LibraryElement** | Il s'agit d'une superclasse abstraite qui peut être stockée dans des zones de stockage et qui peut être importée et exportée avec la syntaxe textuelle définie dans l'Annexe B ou avec la syntaxe XML définie dans la CEI 61499-2. De tels objets ont des attributs fournisseur (vendeur, programmeur, etc.) et version (numéro de version, date, etc.) pour aider à la gestion, en plus d'un nom (hérité de **NamedDeclaration –** voir C.2.2) comme attribut-clé. |
| **Repository** | Les instances de cette classe assurent le stockage permanent et la récupération d'éléments de la bibliothèque. Elles peuvent aussi fournir des services de contrôle de version. |
| **SystemManager** | Les instances de cette classe fournissent les fonctions nécessaires pour prendre en charge les cas d'utilisation INSTALL et OPERATE. |
| **Tester** | Cette classe étend les capacités de la classe SystemManager pour prendre en charge les opérations du cas d'utilisation TEST. |
| **Tool** | Cette classe modélise les comportements génériques des *outils logiciels* pour le support d'ingénierie des IPMCS. |

### C.2.2 Eléments bibliothèques

Les sous-classes de **LibraryElement** sont montrées à la Figure C.2. La production syntaxique dans l'Annexe B correspondant à chaque sous-classe est énumérée dans le Tableau C.2.

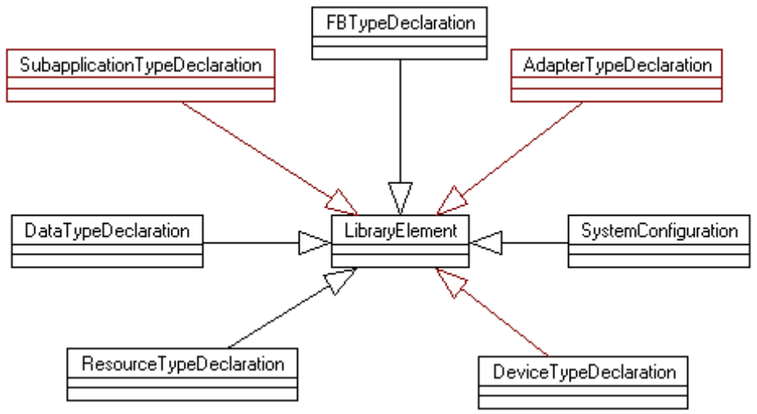



**Figure C.2 – Eléments bibliothèques**

**Tableau C.2 – Productions syntaxiques pour les éléments des bibliothèques**

| Classe | Production syntaxique |
|---|---|
| **DataTypeDeclaration** | type_declaration |
| **FBTypeDeclaration** | fb_type_declaration |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **SubapplicationTypeDeclaration** | subapplication_type_declaration |
| **ResourceTypeDeclaration** | resource_type_specification |
| **DeviceTypeDeclaration** | device_type_specification |
| **SystemConfiguration** | system_configuration |

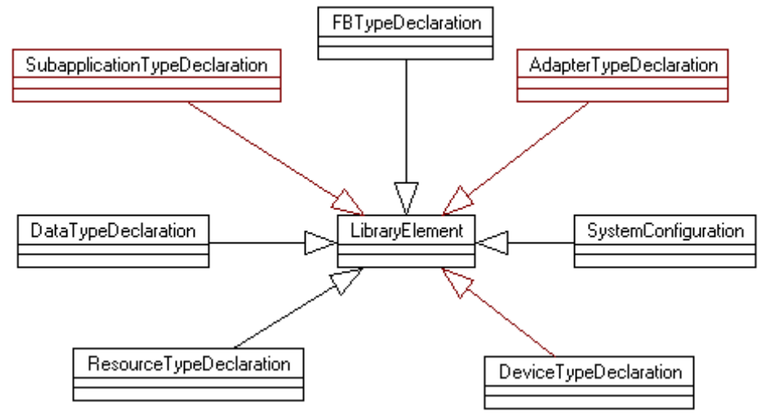




**表C.1 ESS类描述**

| | |
|---|---|
| **Declaration** | 这是声明的抽象超类。 |
| **Editor** | 此类的实例提供了对支持EDIT用例所必需的声明的编辑功能。 |
| **LibraryElement** | 它是一个抽象超类，可以存储在逆止器中，并且可以使用附件B中定义的文本语法或IEC61499-2中定义的XML语法进行导入和导出。此类对象具有供应商（供应商、程序员等）和版本（版本号、日期等）属性以帮助管理，此外还有名称（继承自NamedDeclaration 参见C.2.2），例如key属性。 |
| **Repository** | 此类的实例提供库项目的持久存储和检索。他们还可能提供版本控制服务。 |
| **SystemManager** | 此类的实例提供支持INSTALL和OPERATE用例所需的功能。 |
| **Tester** | 此类扩展了SystemManager类的功能以支持TEST用例的操作。 |
| **Tool** | 此类为IPMCS工程支持的软件工具的一般行为建模。 |

LibraryElement子类如图C.2所示。附录B中对应于每个子类的句法产生列于表C.2中。

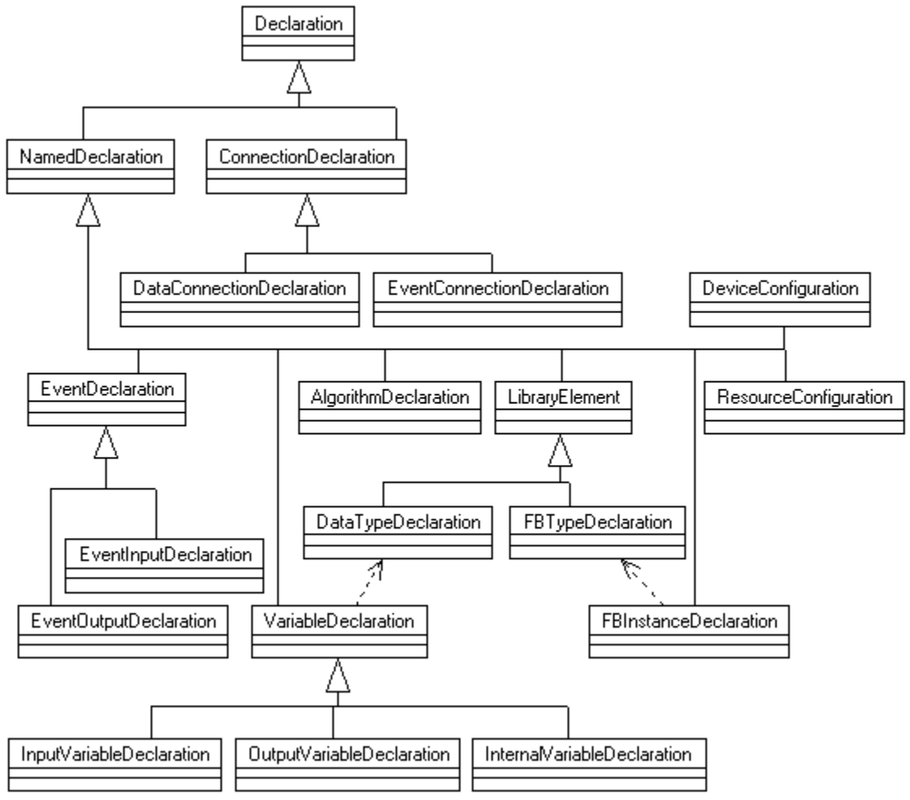



**表C.2 库元素的句法产生式**

| Classe | Production syntaxique |
|---|---|
| **DataTypeDeclaration** | type_declaration |
| **FBTypeDeclaration** | fb_type_declaration |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **SubapplicationTypeDeclaration** | subapplication_type_declaration |
| **ResourceTypeDeclaration** | resource_type_specification |
| **DeviceTypeDeclaration** | device_type_specification |
| **SystemConfiguration** | system_configuration |

### C.2.3 Déclarations

La Figure C.3 montre la hiérarchie de classe des *déclarations* qui peuvent être manipulées par des *outils logiciels*. Les productions syntaxiques dans l'Annexe B correspondant à chacune de ces sous-classes sont énumérées dans le Tableau C.3.



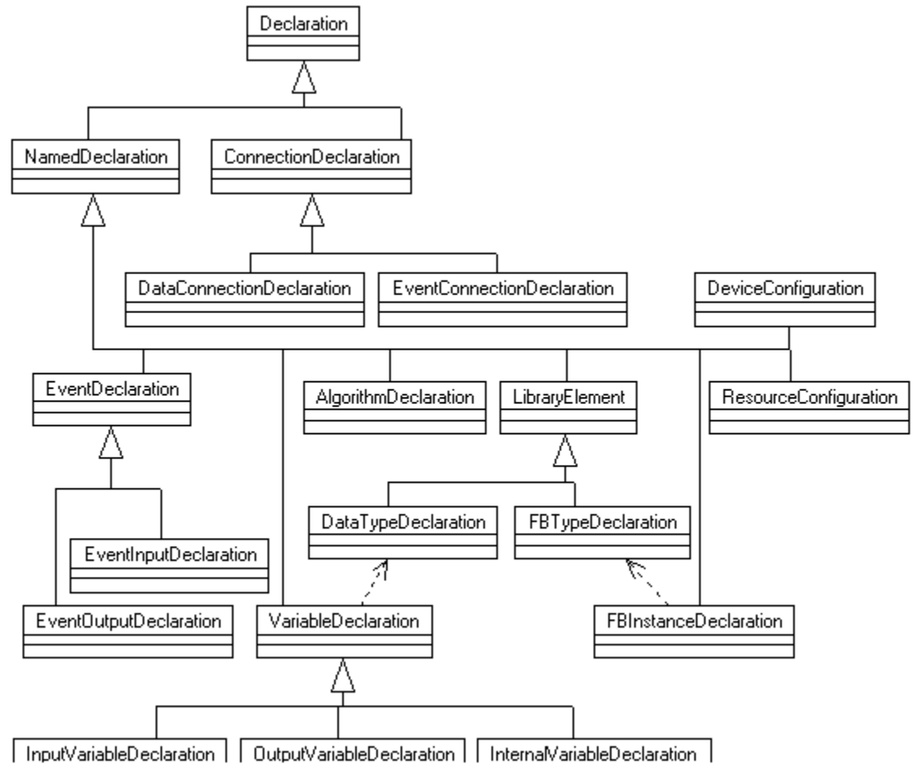

NOTE Pour éviter la confusion, les classes relatives aux *types, instances* et *connexions* d'adaptateurs ne sont pas montrées dans cette Figure; elles sont toutefois énumérées dans le Tableau C.3 pour référence.

**Figure C.3 – Déclarations**





图C.3显示了可由软件工具操作的语句的类层次结构。附录B中对应于这些子类的句法产生式列于表C.3中。




注：为避免混淆，与适配器类型、实例和连接相关的类未在此图中显示；但是，它们在表C.3中列出以供参考。

**Figure C.3 – Déclarations**

**Tableau C.3 – Productions syntaxiques pour les déclarations**

| Classe | Production syntaxique |
|---|---|
| **AdapterConnectionDeclaration** | adapter_conn |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **AlgorithmDeclaration** | fb_algorithm_declaration |
| **DataConnectionDeclaration** | data_conn |
| **DeviceConfiguration** | device_configuration |
| **EventConnectionDeclaration** | event_conn |
| **EventInputDeclaration** | event_input_declaration |
| **EventOutputDeclaration** | event_output_declaration |
| **FBInstanceDeclaration** | fb_instance_definition |
| **InputVariableDeclaration** | input_var_declaration |
| **InternalVariableDeclaration** | internal_var_declaration |
| **OutputVariableDeclaration** | output_var_declaration |
| **PlugDeclaration** | Part of plug_list |
| **ResourceConfiguration** | resource_instance |
| **SocketDeclaration** | Part of socket_list |

### C.2.4 Déclarations des réseaux de blocs fonctionnels

La Figure C.4 montre les relations entre les éléments de *déclarations des réseaux de blocs fonctionnels*. Voir C.2.2 pour les définitions des classes agrégées dans ce diagramme.

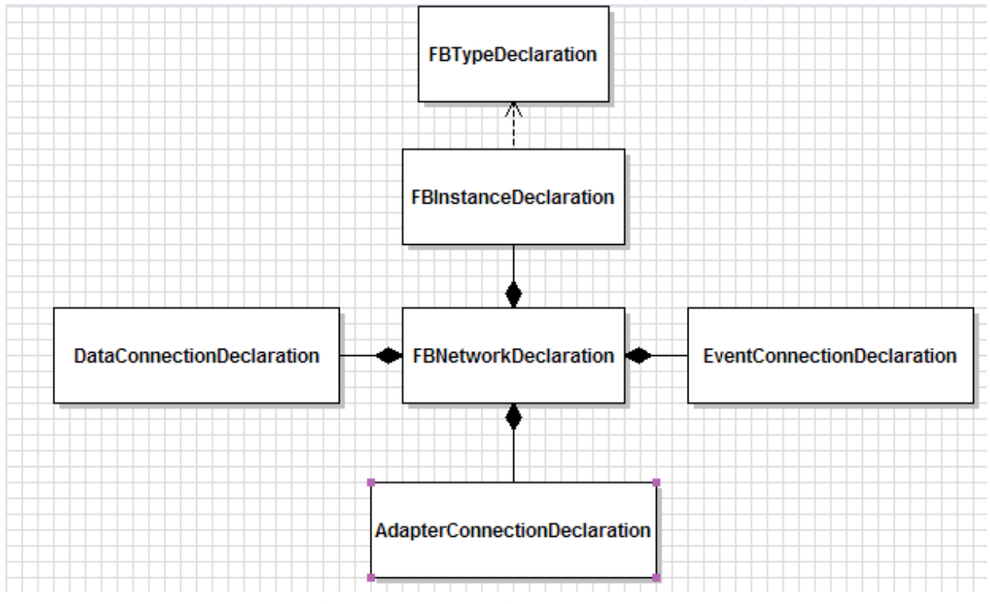



**Figure C.4 – Déclarations des réseaux de blocs fonctionnels**



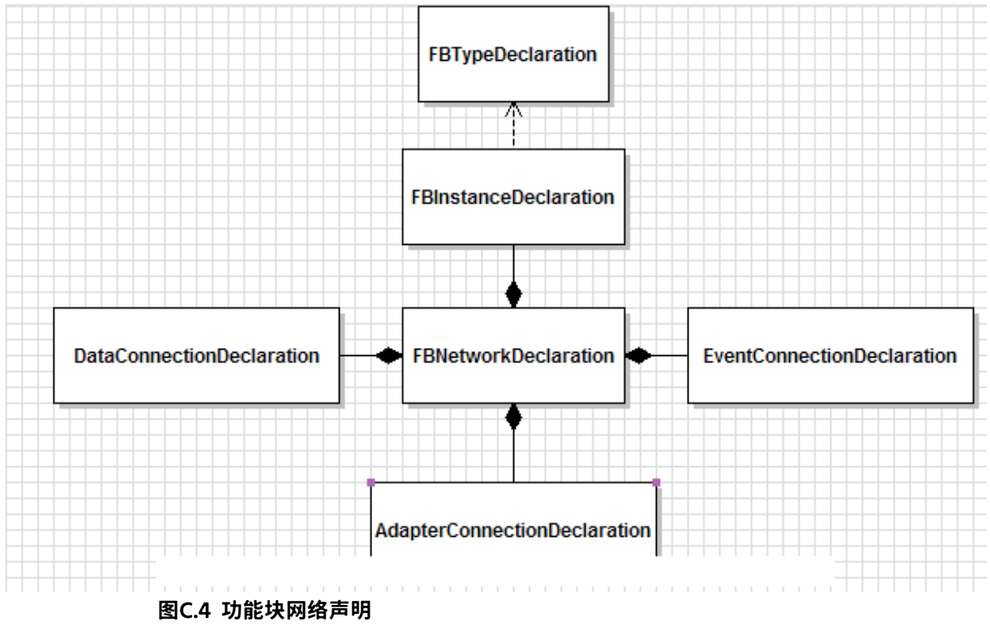


**表C.3 声明的语法产生式**

| Classe | Production syntaxique |
|---|---|
| **AdapterConnectionDeclaration** | adapter_conn |
| **AdapterTypeDeclaration** | adapter_type_declaration |
| **AlgorithmDeclaration** | fb_algorithm_declaration |
| **DataConnectionDeclaration** | data_conn |
| **DeviceConfiguration** | device_configuration |
| **EventConnectionDeclaration** | event_conn |
| **EventInputDeclaration** | event_input_declaration |
| **EventOutputDeclaration** | event_output_declaration |
| **FBInstanceDeclaration** | fb_instance_definition |
| **InputVariableDeclaration** | input_var_declaration |
| **InternalVariableDeclaration** | internal_var_declaration |
| **OutputVariableDeclaration** | ration |
| **PlugDeclaration** | plug_list的一部分 |
| **ResourceConfiguration** | |
| **SocketDeclaration** | socket_list的一部分 |

**功能块网络声明**

图C.4显示了功能块数组声明的元素之间的关系。有关此图中聚合类的定义，请参见C.2.2。




**图C.4 功能块网络声明**

### C.2.5 Déclarations des types de blocs fonctionnels

La Figure C.5 montre les relations entre les éléments de *déclarations des types de blocs fonctionnels*. Les productions syntaxiques pour les classes **EventInputDeclaration, EventOutputDeclaration, InputVariableDeclaration, OutputVariableDeclaration, InternalVariableDeclaration,** et les classes constitutives de **FBNetworkDeclaration** sont données dans le Tableau C.3. Les productions syntaxiques `fb_ecc_declaration` et `fb_service_declaration` de l'Article B.2 correspondent respectivement aux classes **ECCDeclaration** et **ServiceDeclaration**.

NOTE 1 Les déclarations des *sous-applications* sont représentées par des instances de la classe **CompositeFBTypeDeclaration** qui ne contiennent aucune association de données événement `WITH`.

NOTE 2 **NamedDeclaration** est la superclasse abstraite de déclarations qui ont des noms, par exemple, *noms de type* ou *noms d'instance*.

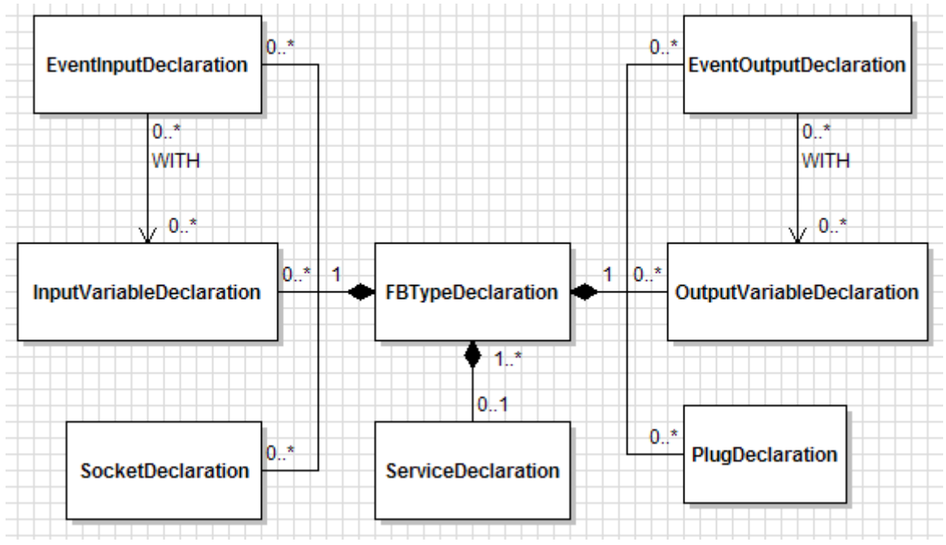



**Figure C.5a – Composition**





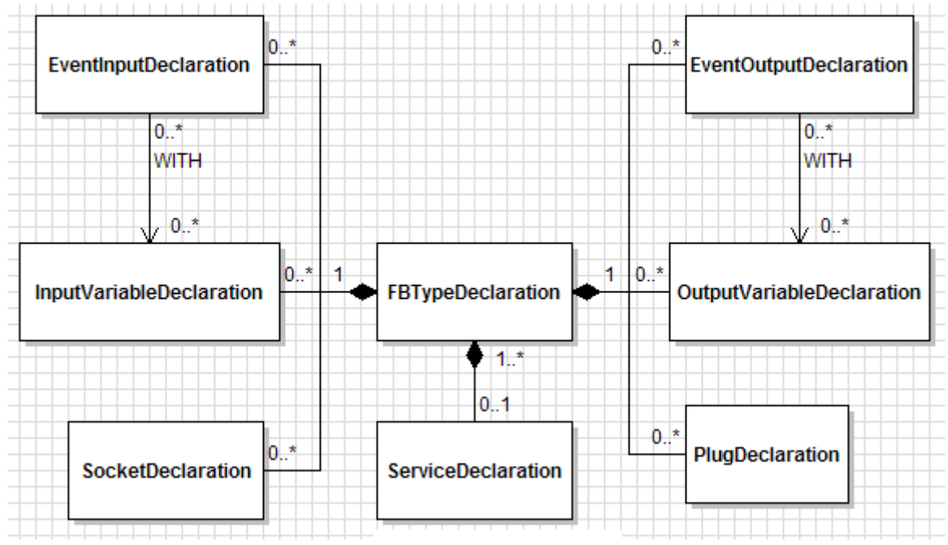
### 功能块类型的声明

图C.5显示了功能块类型声明的元素之间的关系。EventInputDeclaration、EventOutputDeclaration、

**InternalVariableDeclaration和FBNetworkDeclaration的组成类在表C.3中给出。条款B.2中的语法产生fb_ecc_declaration和fb_service_declaration分别对应于ECCDeclaration和ServiceDeclaration类。**

注1子应用声明由类的实例表示
**CompositeFBTypeDeclaration不包含任何WITH事件数据关联。**

注2NamedDeclaration是具有名称的声明的抽象超类，例如，类型名称或实例名称。




**图C.5a 成员**

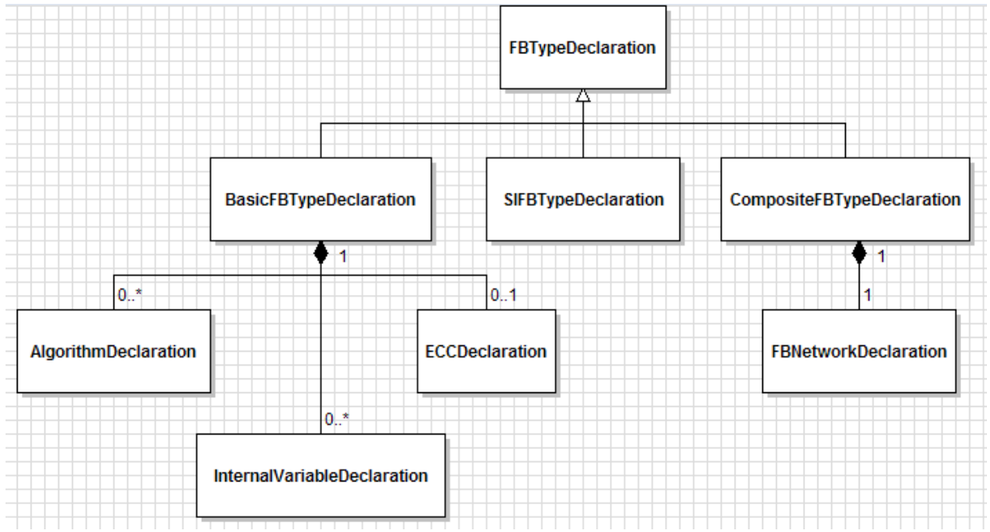



**Figure C.5b – Hiérarchie de classes**

**Figure C.5 – Déclarations des types de blocs fonctionnels**

## C.3 Modèles IPMCS

La Figure C.6 présente une vue d'ensemble des classes principales dans le système de mesure et commande dans les processus industriels (IPMCS). Les descriptions des classes dans la Figure C.6 et leurs objets correspondants dans le système de support d'ingénierie (ESS) sont donnés dans le Tableau C.4.

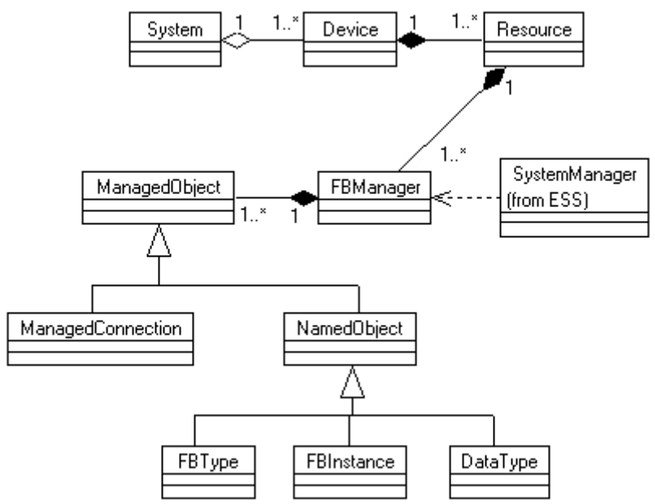
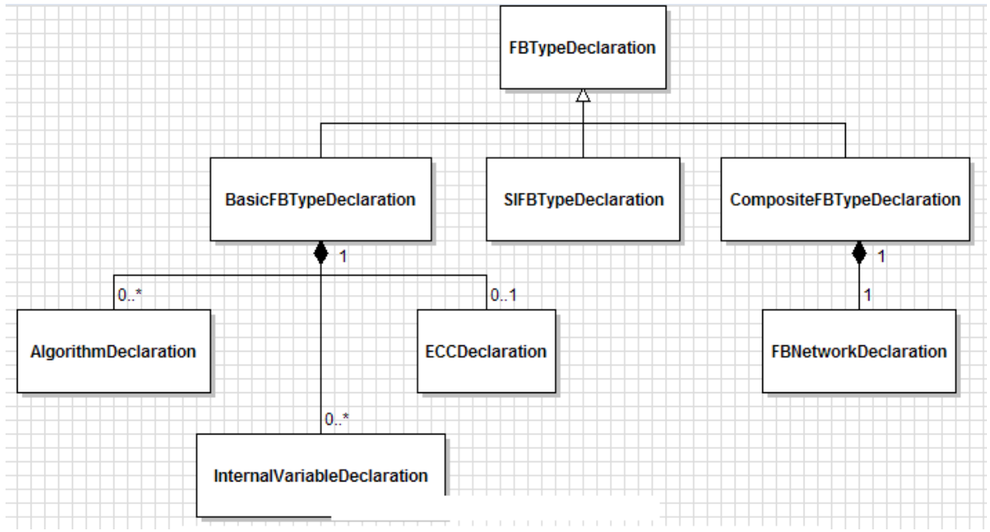



**Figure C.6 – Vue d'ensemble du système IPMCS**



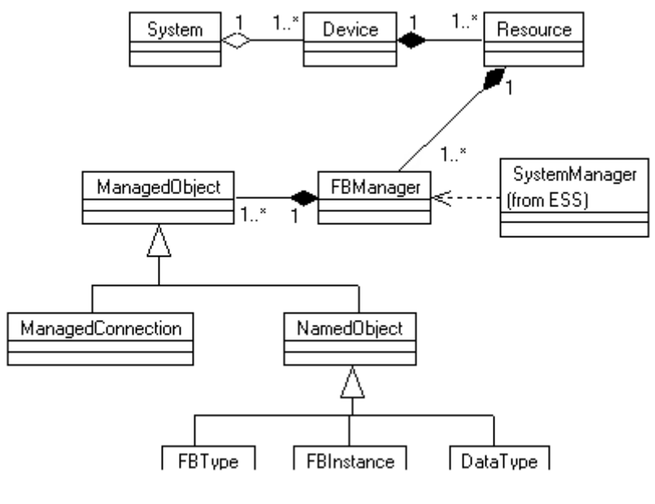





**图C.5b 类层次结构**

**图C.5 功能块类型声明**

图C.6概述了工业过程测量和控制系统(IPMCS)中的主要类别。图C.6中的类的描述及其在工程支持系统（ESS）中的相应对象在表C.4中给出。




**图C.6 IPMCS系统概述**

**Tableau C.4 – Classes des IPMCS**

| Classe des IPMCS | Description | Classe ESS correspondante |
|---|---|---|
| **DataType** | Une instance de cette classe est un *type de données*. | **DataTypeDeclaration** |
| **Device** | Une instance de cette classe représente un *équipement*. | **DeviceConfiguration** |
| **FBInstance** | Une instance de cette classe est une *instance de bloc fonctionnel*. | **FBInstanceDeclaration** |
| **FBManager** | Une instance de cette classe fournit les services de gestion définis à l'Article 6. | **SystemManager** |
| **FBType** | Une instance de cette classe est un *type de bloc fonctionnel*. | **FBTypeDeclaration** |
| **ManagedConnection** | Des instances de cette classe peuvent être accessibles à une instance de la classe **FBManager** utilisant la combinaison de la source et de la destination comme clé unique. | **ConnectionDeclaration** |
| **ManagedObject** | Il s'agit de la superclasse abstraite d'objets qui sont gérés par une instance de la classe **FBManager**. De tels objets peuvent avoir des attributs fournisseur (vendeur, programmateur, etc.) et version (numéro de version, date, etc.) pour aider à la gestion. | None |
| **NamedObject** | Il s'agit de la superclasse abstraite d'objets qui peuvent être accessibles par le nom par une instance de la classe **FBManager**. | **NamedDeclaration** |
| **Resource** | Une instance de cette classe représente une *ressource*. | **ResourceConfiguration** |
| **System** | Une instance de cette classe représente un *Système* de mesure et commande dans les processus industriels (IPMCS). | **SystemConfiguration** |

La Figure C.7 montre les relations entre les éléments d'une *instance de bloc fonctionnel* et son *type de bloc fonctionnel* associé.





**表C.4 IPMCS的类别**

| Classe des IPMCS | | Classe ESS correspondante |
|---|---|---|
| **DataType** | 此类的一个实例是一种数据类型。 | **DataTypeDeclaration** |
| **Device** | 此类的一个实例代表一件设备。 | **DeviceConfiguration** |
| **FBInstance** | 此类的一个实例是一个功能块实例。 | **FBInstanceDeclaration** |
| **FBManager** | 此类的一个实例提供第6章中定义的管理服务。 | **SystemManager** |
| **FBType** | 此类的实例是一种功能块。 | **FBTypeDeclaration** |
| **ManagedConnection** | FBManager类的实例可以使用源和目标的组合作为唯一键来访问此类的实例。 | **ConnectionDeclaration** |
| **ManagedObject** | 它是由FBManager类的实例管理的对象的抽象超类。此类对象可能具有供应商（供应商、开发人员等）和版本（版本号、日期等）属性以帮助管理。 | None |
| **NamedObject** | 它是对象的抽象超类，可以通过FBManager类的实例按名称访问。 | **NamedDeclaration** |
| **Resource** | 此类的一个实例代表一个资源。 | **ResourceConfiguration** |
| **System** | 此类的一个实例表示工业过程测量和控制系统(IPMCS)。 | **SystemConfiguration** |

图C.7显示了功能块实例的元素与其相关的功能块类型之间的关系。

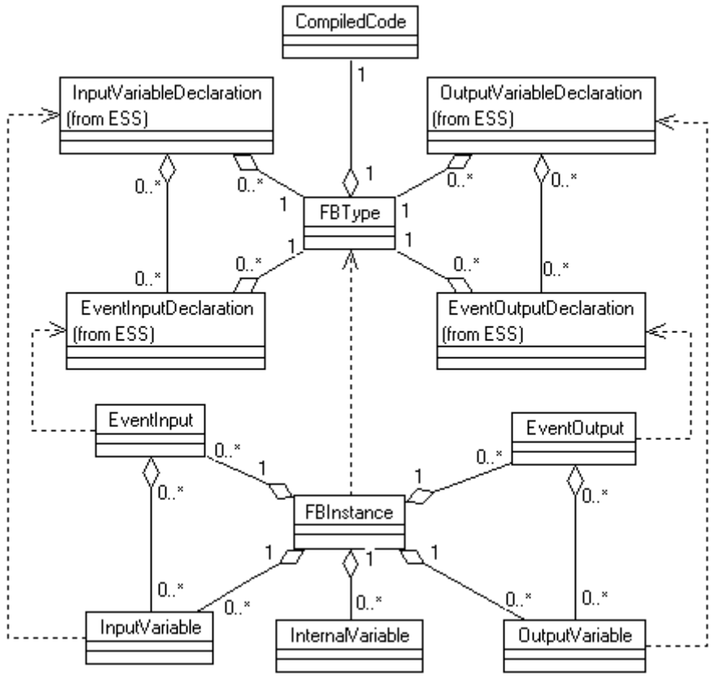
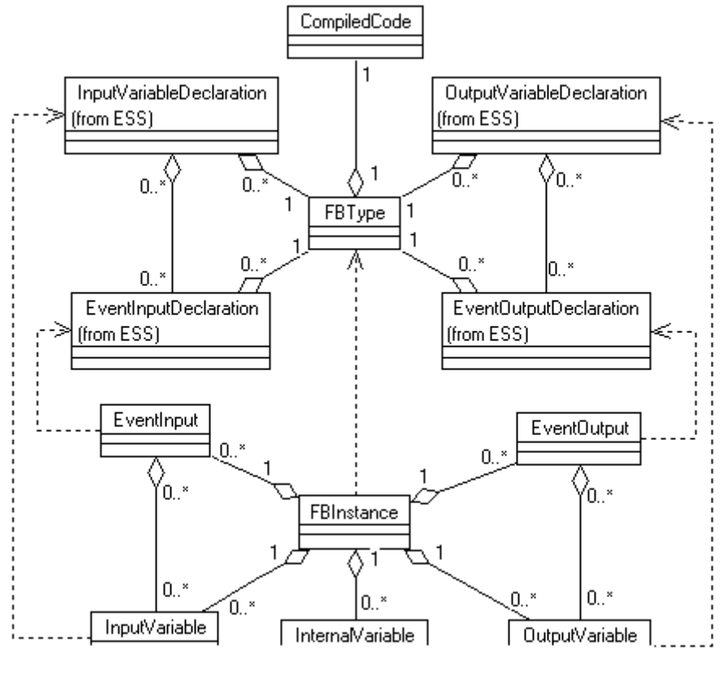



**Figure C.7 – Types et instances de bloc fonctionnel**



61499-1 © CEI:2012 – 207 –




**图C.7 类型和实例deblocfonctionnel**

# Annexe D
(informative)

# Relation à la CEI 61131-3

## D.1 Généralités

Les *fonctions* et les *blocs fonctionnels* tels que définis dans la CEI 61131-3 peuvent être utilisés pour la *déclaration* d'*algorithmes* pour les *types de bloc fonctionnel de base* tels que spécifiés en 5.2.1. L'Article D.2 définit des règles pour la conversion des *fonctions* et des *types de bloc fonctionnel* de la CEI 61131-3 en *bloc fonctionnel de type simples* afin qu'ils puissent être utilisés dans la spécification d'*applications* et de *types de ressource*. L'Article D.3 définit des versions événementielles des fonctions et des blocs fonctionnels de la CEI 61131-3 pour les mêmes utilisations.

## D.2 Blocs fonctionnels "simples"

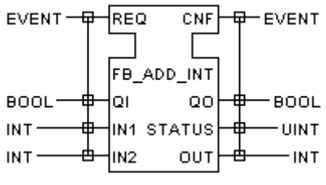
Comme illustré à la Figure D.1, les fonctions et les blocs fonctionnels de la CEI 61131-3 peuvent être convertis en blocs fonctionnels "simples" conformément aux règles suivantes:

a) Les blocs fonctionnels simples sont représentés comme des *blocs fonctionnels interface de service* pour des applications déclenchées par une application telles que montrées à la Figure 21 a).

b) Le *nom de type* du type de bloc fonctionnel simple est le nom du type de fonction ou du type de bloc fonctionnel CEI 61131-3 converti avec le préfixe `FB_` (par exemple, `FB_ADD_INT` à la Figure D.1). Le préfixe `F_` à la place de `FB_` peut être facultativement utilisé pour des types de bloc fonctionnel simple qui sont le résultat des conversions de *fonctions* de la CEI 61131-3.

c) Les variables d'*entrée* et de *sortie* et leurs *types de données* correspondants sont les mêmes que les variables d'entrée et de sortie correspondantes du type de fonction ou du type de bloc fonctionnel de la CEI 61131-3 qui a été converti.

d) L'entrée d'événements `INIT` et la sortie d'événements `INITO` sont utilisées avec des types de bloc fonctionnel simple qui ont été convertis à partir de *types de bloc fonctionnel* de la CEI 61131-3, mais elles ne sont pas utilisées avec des types de bloc fonctionnel simple qui ont été convertis à partir de *fonctions* de la CEI 61131-3.




NOTE Une déclaration textuelle complète de ce type de bloc fonctionnel est donnée dans l'Annexe F.

**Figure D.1 – Exemple de type de bloc fonctionnel "simple"**

Le comportement des *instances* de *types* bloc fonctionnel simple est conforme aux règles suivantes:

e) L'initialisation est comme spécifié en 2.4.2 de la CEI 61131-3:2003 pour les *variables* et comme spécifié en 2.6 de la CEI 61131-3:2003 pour les éléments graphes séquentiels de fonction (SFC).





# 与IEC61131-3的关系

IEC61131-3中定义的功能和功能块可用于声明5.2.1中规定的基本功能块类型的算法。条款D.2定义了将功能和功能块类型从IEC61131-3转换为简单功能块类型的规则，以便它们可以在应用程序和资源类型的规范中使用。条款D.3定义了事件版本的功能和功能块

IEC61131-3用于相同用途。

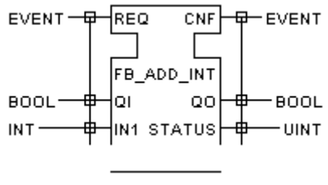
如图D.1所示，IEC61131-3的功能和功能块可以按照以下规则转换为"简单"功能块：

a)简单功能块表示为应用程序触发应用程序的服务接口功能块，如图21a)所示。

b)简单功能块类型的类型名称是功能类型或IEC61131-3功能块类型转换的名称加上前缀FB_（例如图D.1中的FB_ADD_INT）。F_前缀而不是FB_可以选择用于作为IEC61131-3功能转换结果的简单功能块类型。

c)输入和输出变量及其对应的数据类型与已转换的IEC61131-3功能类型或功能块类型的相应输入和输出变量相同。

d)INIT事件输入和INITO事件输出用于从IEC61131-3功能块类型转换而来的简单功能块类型，但不用于从IEC61131-3功能转换而来的简单功能块类型。




注：此类功能块的完整文本声明在附录F中给出。

**图D.1 "简单"功能块类型示例**

简单功能块类型实例的行为符合以下规则：

e)初始化如IEC61131-3:2003的2.4.2中对变量的规定和IEC61131-3:2003的2.6中对功能序列图(SFC)元素的规定。

f) L'occurrence d'une primitive du service INIT+ équivaut à une initialisation en «redémarrage à froid» telle que définie dans les paragraphes susmentionnés de la CEI 61131-3:2003, suivie d'une primitive de service INITO+ avec une valeur de STATUS de zéro (0).

g) L'occurrence d'une primitive de service INIT- ou REQ- n'a aucun effet, sauf d'induire une primitive de service INITO- ou CNF- avec une valeur de STATUS égale à un (1).

h) L'occurrence d'une primitive de service REQ+ entraîne l'*exécution* de l'*algorithme* spécifié dans le corps du bloc fonctionnel, conformément aux règles données dans la CEI 61131-3 pour le langage dans lequel l'algorithme est programmé.

i) L'exécution réussie de l'algorithme en réponse à une primitive REQ+ donne lieu à une primitive CNF+ avec une valeur de STATUS égale à zéro (0).

j) S'il se produit une erreur au cours de l'exécution de l'algorithme, le résultat est une primitive CNF- avec une valeur de STATUS déterminée conformément au Tableau D.1.

**Tableau D.1 – Sémantique des valeurs de STATUS**

| Valeur | Sémantique |
|---|---|
| **0** | Fonctionnement normal |
| **1** | Propagation de INIT- ou de REQ- |
| **2** | Erreur de conversion de type |
| **3** | Le résultat numérique dépasse la plage pour le type de données |
| **4** | Division par zéro |
| **5** | Sélecteur (K) hors plage pour la fonction MUX |
| **6** | Position du caractère spécifiée non valide |
| **7** | Le résultat dépasse la longueur maximale de la chaîne |
| **8** | Simultanément vraies, transactions non classées par ordre de priorité, dans une divergence de sélection |
| **9** | Erreur de conflit du contrôle d'action |
| **10** | Retour issu d'une fonction sans valeur assignée |
| **11** | L'itération ne se termine pas |
| **12** | Valeur d'indice non valide |
| **13** | Erreur de taille de matrice |

## D.3 Fonctions et blocs fonctionnels événementiels

Des *fonctions* de la CEI 61131-3 peuvent être converties en blocs fonctionnels pour un usage efficace dans des systèmes événementiels conformément aux règles données à l'Article D.2 avec les modifications suivantes:

a) le *nom de type* du type de bloc fonctionnel événementiel est le même que le nom de la fonction CEI 61131-3 convertie avec le préfixe complémentaire E_, par exemple, E_ADD_INT;

b) une primitive CNF+ ou CNF- ne suit pas l'exécution de l'algorithme, à moins qu'une telle exécution donne lieu à une valeur modifiée de la sortie de fonction.

NOTE Si le «chaînage en cascade» des sorties CNF aux entrées REQ est utilisé pour mettre en œuvre une séquence de calculs, la séquence s'arrêtera au premier point où une valeur de sortie ne change pas.

En général, étant donné que les *blocs fonctionnels* de la CEI 61131-3 ont des informations d'états internes, de tels blocs doivent être spécialement convertis pour être utilisables dans des systèmes événementiels. Par exemple, le bloc fonctionnel E_DELAY montré dans le Tableau A.1 peut être utilisé par un grand nombre des fonctions de retard fournies par les





f) INIT+服务原语的出现相当于IEC61131-3:2003的上述条款中定义的"冷重启"初始化，然后是状态值为零(0)的INITO+服务原语。

g)INIT或REQ服务原语的出现没有任何影响，除了诱导一个STATUS值为一(1)的INITO或CNF服务原语。

h)根据IEC61131-3中给出的算法编程语言规则，REQ+服务原语的出现导致功能块主体中指定的算法的执行。

i) 响应REQ+原语的算法的成功执行导致CNF+原语的STATUS值为零(0)。

j) 如果在算法执行期间发生错误，则结果是一个CNF原语，其STATUS值根据表D.1确定。

**表D.1 STATUS值的语义**

| Valeur | Sémantique |
|---|---|
| **0** | |
| **1** | INIT或REQ传播 |
| **2** | 类型转换错误 |
| **3** | 数值结果超出数据类型的范围 |
| **4** | 被零除 |
| **5** | 选择器(K)超出MUX功能范围 |
| **6** | 指定的字符位置无效 |
| **7** | 结果超过了字符串的最大长度 |
| **8** | 在选择分歧中同时为真、无优先级的交易 |
| **9** | 动作控制冲突错误 |
| **10** | 从没有赋值的函数返回 |
| **11** | 迭代没有结束 |
| **12** | 无效的索引值 |
| **13** | Erreur de taille de matrice |

IEC61131-3的功能可以根据D.2中给出的规则转换为功能块，以便在事件系统中有效使用，并进行以下修改：

a)事件功能块类型的类型名称与转换后的IEC61131-3功能的名称相同，带有附加前缀E_，例如E_ADD_INT；

b)CNF+或CNF原语不跟随算法的执行，除非这样的执行导致函数输出的修改值。

注意如果使用CNF输出到REQ输入的"级联"来执行一系列计算，则该序列将在输出值不改变的第一个点停止。

一般来说，由于IEC61131-3的功能块具有内部状态信息，因此必须对这些块进行特殊转换才能在事件驱动系统中使用。例如，表A.1中所示的E_DELAY功能块可以被许多由

blocs fonctionnels temporisateurs de la CEI 61131-3. Un exemple d'une conversion du bloc fonctionnel CTU de la norme CEI 61131-3 est donné comme Caractéristique n°18 dans le Tableau A.1.

## D.4 Conformité à la CEI 61131-3

Les mises en œuvre de la présente norme doivent se conformer aux exigences des paragraphes 1.5.1, 2.1, 2.2, 2.3 et 2.4 de la CEI 61131-3:2003 et aux éléments associés de l'Annexe B de la CEI 61131-3:2003 pour la syntaxe et la sémantique de représentation textuelle d'éléments communs, avec les exceptions et les extensions notées à l'Article D.5.

Lorsque des productions syntaxiques ne sont pas données pour les symboles non terminaux dans l'Annexe B, les productions syntaxiques correspondantes données dans l'Annexe B de la CEI 61131-3:2003 doivent s'appliquer.

## D.5 Exceptions

Les mises en œuvre de la présente norme **ne doivent pas** utiliser la notation de *variable directement représentée* définie en 2.4.1.1 de la CEI 61131-3:2003 et les caractéristiques associées dans d'autres paragraphes. Cependant, un *libellé* de type STRING ou WSTRING, contenant une chaîne dont la syntaxe et la sémantique correspondent à la notation de variable directement représentée, peut être utilisé comme *paramètre* d'un *bloc fonctionnel interface de service* qui fournit un accès à la variable correspondante.

## D.6 Interfonctionnement avec des dispositifs de commande programmables

### D.6.1 Vue d'ensemble

Un dispositif de commande programmable peut agir comme un *serveur* tel que défini dans la CEI 61131-5, par rapport à un *équipement* tel que défini dans la présente norme, agissant comme *client* tel que défini dans la CEI 61131-5. Ces services sont fournis en utilisant les moyens définis dans la CEI 61131-5 et sont accessibles à partir d'un équipement CEI 61499 utilisant les instances des *types de bloc fonctionnel* spécifiés dans l'Annexe D. Ces types de blocs fonctionnels sont modélisés comme des *types bloc fonctionnel de communication* tels que définis dans la présente norme.

L'équipement client CEI 61499 peut exister sur un réseau de communication avec le dispositif de commande programmable agissant comme serveur ou peut être un sous-système spécifique au réalisateur au sein du «bloc de traitement principal» du dispositif de commande programmable, tel qu'illustré à la Figure 4 de la CEI 61131-5:2000. Dans un cas comme dans l'autre, l'interaction entre l'équipement client CEI 61499 et le bloc de traitement principal est modélisée comme se produisant sur une ou plusieurs *connexions de communication* telles que définies dans la CEI 61499-1, en utilisant les instances des types de bloc fonctionnel définis dans l'Annexe D.

### D.6.2 Conventions de service

À l'exception des extensions définies dans l'Annexe D, les conventions pour nommer des variables et des événements d'entrée et de sortie et pour décrire les *services* (tels que définis dans la présente norme) fournis par des instances des types de bloc fonctionnel décrites dans l'Annexe D, sont telles que définies dans la CEI 61499-1 pour les descriptions des *types bloc fonctionnel interface de service* et des *types bloc fonctionnel de communication*.

Pour les besoins de l'Annexe D, l'entrée PARAMS de type ANY défini dans la présente norme est remplacée par une entrée ID de type WSTRING. Le contenu de cette chaîne spécifie une représentation **dépendante de la mise en œuvre** du chemin allant vers la *variable* concernée du serveur.



IEC61131-3定时器功能块。从IEC61131-3转换CTU功能块的示例在表A.1中作为特征#18给出。

本标准的实施应符合IEC61131-3:2003的1.5.1、2.1、2.2、2.3和2.4小节以及IEC61131-3:2003附件B的相关部分对文本的语法和语义的要求。通用元素的表示，但第D.5条中注明的例外和扩展除外。

如果附录B中的非终结符号没有给出句法产生式，则应适用IEC61131-3:2003附录B中给出的相应句法产生式。

本标准的实施不应使用IEC61131-3:2003的2.4.1.1中定义的直接表示的变量符号以及其他子条款中的相关特征。但是，类型为STRING或WSTRING的标签，包含其语法和语义对应于直接表示的变量表示法的字符串，可以用作提供对相应变量的访问的服务接口功能块的参数。

## D.6 与可编程控制设备互通

对于本标准中定义的设备，可编程控制器可以充当IEC61131-5中定义的服务器，充当IEC61131-5中定义的客户端。这些服务是使用IEC61131-5中定义的方式提供的，并且可以使用附录D中指定的功能块类型的实例从IEC61499设备访问。这些功能块类型被建模为本标准中定义的通信功能块类型。

IEC61499客户端设备可能存在于通信网络上，可编程控制器充当服务器，也可能是可编程控制器"主处理块"内的特定于实现者的子系统，如IEC61131-5的图4所示:2000。在任何一种情况下，IEC61499客户端设备和主处理块之间的交互都被建模为发生在IEC61499-1中定义的一个或多个通信连接上，使用附件D中定义的功能块类型的实例。

### 服务协议

除附录D中定义的扩展外，命名输入和输出变量和事件以及描述由附录D中描述的块类型实例提供的服务（如本标准中定义）的约定在IEC61499-1中定义用于服务接口功能块类型和通信功能块类型的描述。

就附录D而言，本标准中定义的ANY类型的PARAMS条目由WSTRING类型的ID条目代替。此字符串的内容指定相关服务器变量的路径的实现相关表示。

EXEMPLE 1 Dans le cas où l'équipement client CEI 61499 se situe dans la proximité logique du serveur CEI 61131, il peut suffire de nommer simplement le *chemin d'accès* CEI 61131-3 par la variable souhaitée dans l'entrée `ID`, par exemple `"CELL_1.CHARLIE"` dans l'exemple montré à la Figure 19a de la CEI 61131-3:2003.

EXEMPLE 2 Dans le cas où l'équipement client CEI 61499 est relié à distance au serveur 61131-3 par un réseau de communication, il peut être possible d'utiliser l'entrée `ID` pour encapsuler un identificateur de ressource universel (URI) pour spécifier le chemin d'accès souhaité, par exemple, `"http://192.168.0.1:61131/CELL_1.CHARLIE"`.

NOTE Lorsque cela est pris en charge par une mise en œuvre, l'entrée `ID` peut spécifier un chemin d'accès vers une variable de statut, tel que les chemins d'accès prédéfinis `P_PCSTATUS` et `P_PCSTATE` spécifiés dans la CEI 61131-5.

Lorsque cela est utilisé, le contenu de l'entrée TYPE d'un type de bloc fonctionnel défini dans l'Annexe D spécifie le nom du *type de données* des données (SD ou RD) transférées. Il peut s'agir du nom d'un type de données élémentaire tel que `"BOOL"` ou un type de données dérivé tel que `"ANALOG_ 16_INPUT_DATA"`.

Lorsque cela est utilisé, le contenu de l'entrée TASK d'un type de bloc fonctionnel défini dans l'Annexe D spécifie une représentation **dépendante de la mise en œuvre** du chemin allant à la *tâche* concernée dans le serveur.

EXEMPLE 3 Dans le cas où l'équipement client CEI 61499 se situe dans la proximité logique d'un serveur CEI 61131-3 configuré comme montré à la Figure 19 a) de la CEI 61131-3:2003, un chemin allant à la tâche nommée `SLOW_1` dans la ressource `STATION_1` pourrait être représenté comme `"CELL_1.STATION_1.SLOW_1"`.

Les valeurs de la sortie STATUS des types de bloc fonctionnel définis en D.6.3 sont telles que données dans le Tableau 24 de la CEI 61131-5:2000.

### D.6.3 Types de bloc fonctionnel

#### D.6.3.1 READ

Une instance du type de bloc fonctionnel READ montré sous forme graphique à la Figure D.2 et sous forme textuelle dans le Tableau D.2 peut être utilisée par un équipement client CEI 61499 pour lire les valeurs de variables programme ou le statut dans le serveur CEI 61131-3.





示例1在IEC61499客户端设备在逻辑上接近IEC61131服务器的情况下，通过ID条目中的所需变量简单地命名IEC61131-3路径就足够了，例如"CELL_1.CHARLIE"在IEC61131-3:2003的图19a所示的示例。

示例2在IEC61499客户端设备通过通信网络远程链接到61131-3服务器的情况下，可以使用ID条目来封装通用资源标识符

注：当实现支持时，ID条目可以指定状态变量的路径，例如IEC61131-5中指定的预定义路径P_PCSTATUS和P_PCSTATE。

使用时，附件D中定义的功能块类型的TYPE条目的内容指定正在传输的数据（SD或RD）的数据类型名称。它可以是基本数据类型的名称，例如"BOOL"，也可以是派生数据类型的名称，例如"ANALOG_16_INPUT_DATA"。

使用时，附件D中定义的功能块类型的TASK条目的内容指定了服务器中相关任务的路径的实现相关表示。

示例3在IEC61499客户端设备与IEC61131-3服务器逻辑接近的情况下，如IEC61131-3:2003的图19a)所示配置，到STATION_1资源中名为SLOW_1的任务的路径可以是表示为"CELL_1.STATION_1.SLOW_1"。

D.6.3中定义的功能块类型的STATUS输出值如IEC61131-5:2000的表24给出。

### D.6.3 功能块类型

IEC61499客户端设备可以使用图D.2中以图形方式显示和表D.2中以文本形式显示的READ功能块类型的实例，以从服务器读取程序变量值或状态

CEI 61131-3.

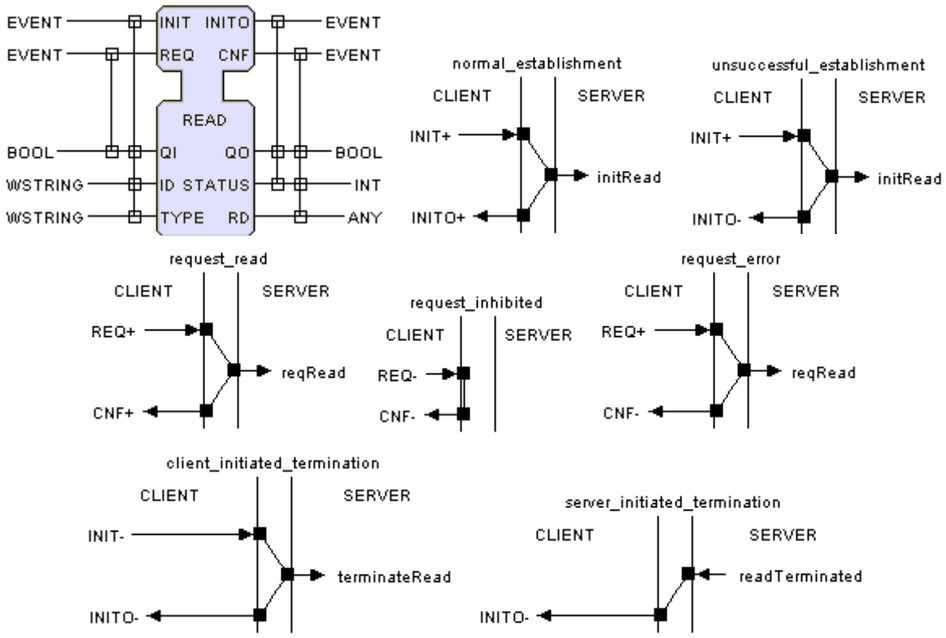
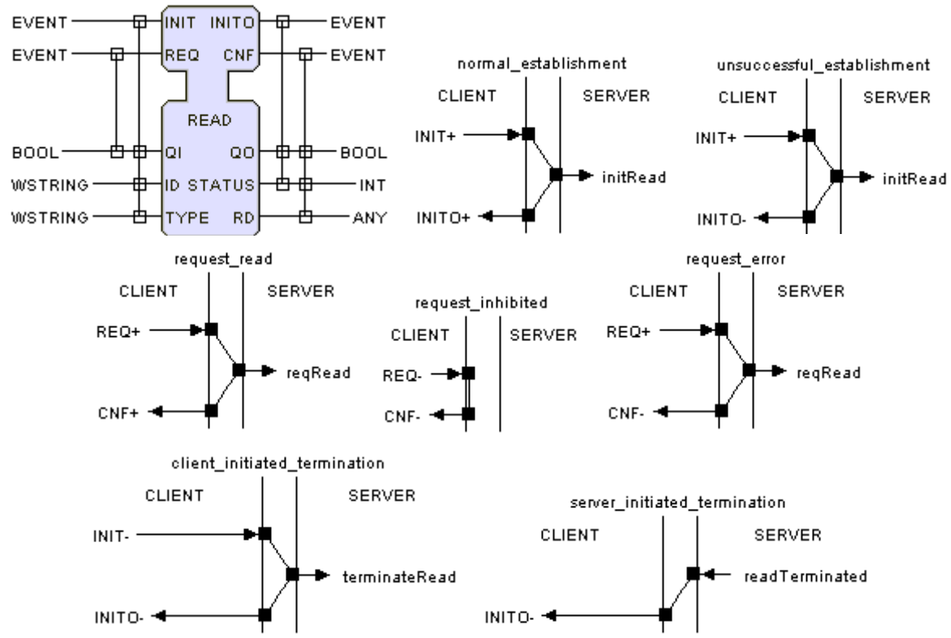



**Figure D.2 – Type de bloc fonctionnel READ**



– 212 – 61499-1 © CEI:2012




**图D.2 类型debloc函数READ**

**Tableau D.2 – Code source du type de bloc fonctionnel READ**

```
FUNCTION_BLOCK READ (* Lire la variable statut ou programme du serveur *)
EVENT_INPUT
  INIT WITH QI,ID,TYPE; (* Initialiser/Arrêter le service *)
  REQ WITH QI; (* Demande de service *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
  CNF WITH QO,STATUS,RD; (* Confirmation du service demandée *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Qualificateur d'entrée d'événements *)
  ID: WSTRING; (* Chemin allant vers la variable devant être lue *)
  TYPE: WSTRING; (* Type de données de variable RD *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Fonctionnement normal, 0= Fonctionnement anormal *)
  STATUS: INT;
  RD: ANY; (* Données variables issues de l'équipement CEI 61131 *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_read
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF+(RD);
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateRead(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.readTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.2 UREAD

Une instance du type de bloc fonctionnel UREAD montré sous forme graphique à la Figure D.3 et sous forme textuelle dans le Tableau D.3 peut être utilisée par un équipement client CEI 61499 pour demander une notification asynchrone d'un changement de valeur de variable programme ou de statut auprès d'un serveur CEI 61131-3. La notification est reçue par la sortie d'événements IND du bloc à la fin de l'exécution de la tâche spécifiée lorsqu'un changement de la valeur de la variable spécifiée (par rapport à sa valeur à la suite du lancement de l'exécution de la tâche) est détecté.

Une instance de ce type de bloc fonctionnel peut aussi être utilisée pour recevoir une notification de fin de chaque exécution de la tâche spécifiée en laissant non spécifiées les entrées ID et TYPE du bloc.



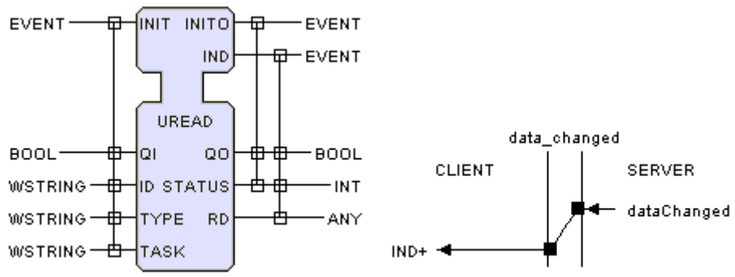


**表D.2 READ功能块类型源代码**

```
FUNCTION_BLOCKREAD(*读取服务器状态或程序变量*)

用QI ID TYPE初始化；(*初始化停止服务*)
要求QI；（*服务请求*）
END_EVENT


  INITO WITH QO.STATUS; (* Initialiser/Arrêter «Confirm» *)
具有QO、状态、RD的CNF；(*请求服务确认*)



问：布尔值；(*事件输入限定符*)ID:WSTRING;(*要读取的变量的路径*)

类型：WSTRING；(*RD变量数据类型*)



QO：布尔值；（*1=正常运行，0=异常运行*）
  STATUS：INT;
RD：任何；(*来自IEC61131设备的可变数据*)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initRead(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_read
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF+(RD);
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+() -> SERVER.reqRead(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateRead(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.readTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

IEC61499客户端设备可以使用图D.3中以图形形式和表D.3中以文本形式显示的UREAD功能块类型的实例，以通过IEC请求变量值更改程序或状态的异步通知61131-3服务器。当检测到指定变量的值（与启动执行作业后的值相比）发生变化时，块的IND事件输出在指定任务的执行结束时收到通知。

这种类型的功能块的实例也可用于接收指定任务的每次执行完成的通知，方法是不指定块的ID和TYPE输入。




NOTE La représentation graphique d'autres séquences de service énumérées dans le Tableau D.3 est similaire à la Figure D.2.

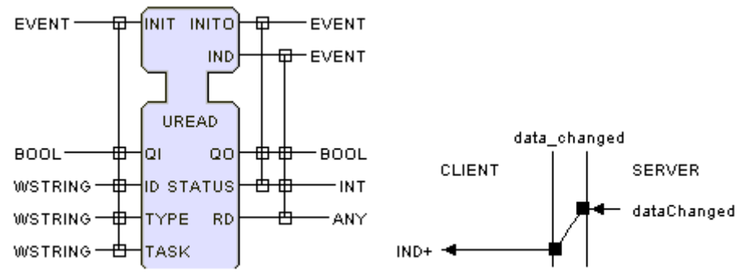
**Figure D.3 – Type de bloc fonctionnel UREAD**

**Tableau D.3 – Code source du type de bloc fonctionnel UREAD**

```
FUNCTION_BLOCK UREAD (* Lecture non sollicitée de la variable programme ou statut de
la CEI 61131 *)

EVENT_INPUT
  INIT WITH QI,ID,TASK,TYPE; (* Initialiser/Arrêter le service *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
  IND WITH QO,STATUS,RD; (* Indication de la valeur RD modifiée *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Qualificateur d'entrée d'événements *)
  ID: WSTRING; (* Chemin allant vers la variable devant être lue *)
  TYPE: WSTRING; (* Type de données de variable RD *)
  TASK: WSTRING; (* Chemin vers TASK CEI 61131 déclenchant la lecture sur valeur
modifiée *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Fonctionnement normal, 0= Fonctionnement anormal *)
  STATUS: INT;
  RD: ANY; (* Données d'entrée issues de ressource *)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK)
    -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE data_changed
  SERVER.dataChanged() -> CLIENT.IND+(RD);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateURead() -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.UreadTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```








注：表D.3中列出的其他服务序列的图形表示类似于图D.2。

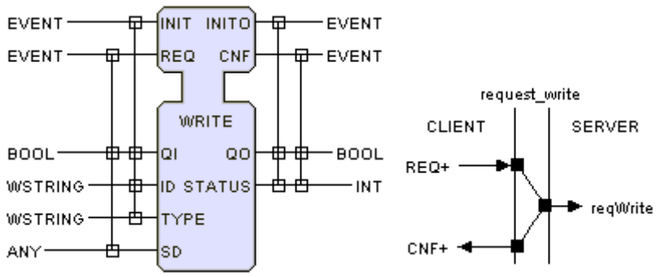
**图D.3 UREAD功能块类型**

**表D.3 UREAD功能块类型源代码**

```
FUNCTION_BLOCKUREAD(*主动读取IEC61131程序或状态变量*)


EVENT INPUT
用QI ID TASK TYPE初始化；(*初始化停止服务*)
END_EVENT


  INITO WITH OO.STATUS; (* Initialiser/Arrêter «Confirm» *)
IND与QO、状态、RD；(*表示修改后的RD值*)



问：布尔值；(*事件输入限定符*)ID:WSTRING;(*要读取的变量的路径*)

类型：WSTRING；(*RD变量数据类型*)
任务：WSTRING；（*IEC61131TASK触发读取修改值的路径*）END_VAR



QO：布尔值；（*1=正常运行，0=异常运行*)
  STATUS: INT;
RD：任何；(*从资源输入数据*)
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE,TASK) -> SERVER.initURead(ID,TYPE,TASK)
    -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE data_changed
  SERVER.dataChanged() -> CLIENT.IND+(RD);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateURead() -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.UreadTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.3 WRITE

Une instance du type de bloc fonctionnel WRITE montré sous forme graphique à la Figure D.4 et sous forme textuelle dans le Tableau D.4 peut être utilisée par un équipement client CEI 61499 pour écrire des valeurs de données variables dans un serveur CEI 61131-3.



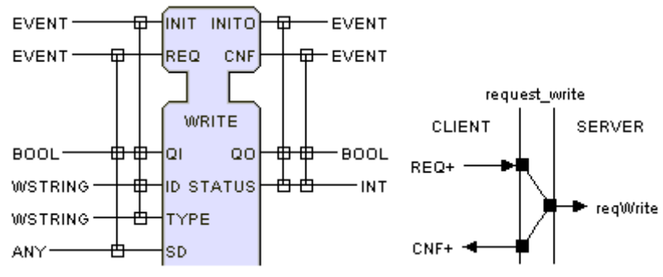

NOTE La représentation graphique d'autres séquences de service énumérées dans le Tableau D.4 est similaire à la Figure D.2.

**Figure D.4 – Type de bloc fonctionnel WRITE**



61499-1 © CEI:2012 – 215 –

IEC61499客户端设备可以使用图D.4中以图形方式显示和表D.4中以文本形式显示的WRITE功能块类型的实例将可变数据值写入IEC61131-3。




注：表D.4中列出的其他服务序列的图形表示类似于图D.2。

**图D.4 WRITE功能块的类型**

**Tableau D.4 – Code source du type de bloc fonctionnel WRITE**

```
FUNCTION_BLOCK WRITE (* Écrire une valeur de variable dans un serveur CEI 61131 *)
EVENT_INPUT
  INIT WITH QI,ID,TYPE; (* Initialiser/Arrêter le service *)
  REQ WITH QI,SD; (* Demande de service *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
  CNF WITH QO,STATUS; (* Confirmation du service demandée *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Qualificateur d'entrée d'événements *)
  ID: WSTRING; (* Chemin allant vers la variable devant être lue *)
  TYPE: WSTRING; (* Type de données de variable SD *)
  SD: ANY; (* Valeur de variable à écrire *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Fonctionnement normal, 0= Fonctionnement anormal *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_write
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateWrite(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.writeTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.3.4 TASK

Une instance du type de bloc fonctionnel TASK montré sous forme graphique à la Figure D.5 et sous forme textuelle dans le Tableau D.5 peut être utilisée par un équipement client CEI 61499 pour demander l'exécution d'une tâche sur un serveur CEI 61131-3.

Lorsqu'une mise en œuvre prend en charge cette caractéristique, aucune valeur n'est configurée pour l'entrée `SINGLE` ou `INTERVAL` du bloc TASK correspondant tel que défini dans le Tableau 50 de la CEI 61131-3:2003; plus exactement, l'exécution de la tâche correspondante est déclenchée comme illustré dans la séquence de service `request_task` montrée à la Figure D.5.



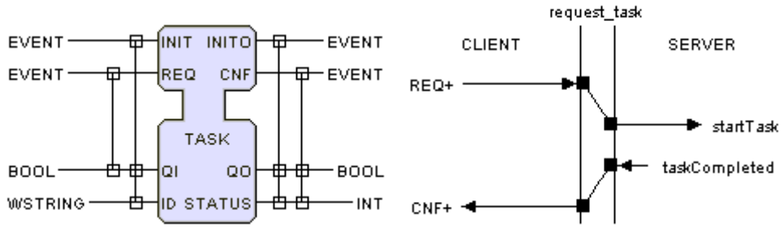


**表D.4 WRITE功能块类型源代码**

```
FUNCTION_BLOCKWRITE（*在IEC61131服务器中写入变量值*）

用QI ID TYPE初始化；(*初始化停止服务*)
请求QI SD；(*服务请求*)
END_EVENT


  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
带QO、状态的CNF；(*请求服务确认*)



问：布尔值；(*事件输入限定符*)ID:WSTRING;(*要读取的变量的路径*)

类型：WSTRING；(*SD变量数据类型*)
标清：任何；(*要写入的变量值*)


VAR_OUTPUT
QO：布尔值；（*1=正常运行，0=异常运行*）
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID,TYPE) -> SERVER.initWrite(ID,TYPE) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_write
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID,SD) -> SERVER.reqWrite(ID,SD) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateWrite(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.writeTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

IEC61499客户端设备可以使用图D.5中以图形形式和表D.5中以文本形式显示的TASK功能块类型的实例来请求在IEC服务器61131-3上执行任务。

当实现支持此功能时，不会为IEC61131-3:2003的表50中定义的相应TASK块的SINGLE或INTERVAL输入配置值；更准确地说，相应任务的执行被触发，如图D.5所示的request_task服务序列所示。




NOTE La représentation graphique d'autres séquences de service énumérées dans le Tableau D.5 est similaire à la Figure D.2.

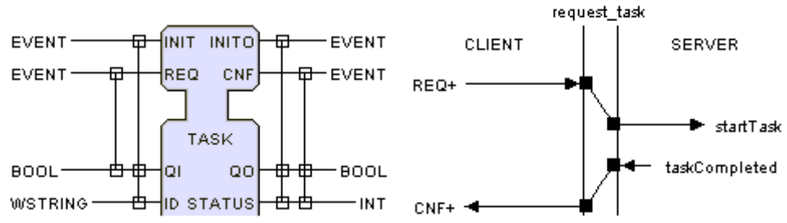
**Figure D.5 – Type de bloc fonctionnel TASK**

**Tableau D.5 – Code source du type de bloc fonctionnel TASK**

```
FUNCTION_BLOCK TASK (* Déclencher une tâche CEI 61131 *)
EVENT_INPUT
  INIT WITH QI,ID; (* Initialiser/Arrêter le service *)
  REQ WITH QI; (* Demande de service *)
END_EVENT

EVENT_OUTPUT
  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
  CNF WITH QO,STATUS; (* Confirmation du service demandée *)
END_EVENT

VAR_INPUT
  QI: BOOL; (* Qualificateur d'entrée d'événements *)
  ID: WSTRING; (* Chemin allant à la tâche devant être déclenchée *)
END_VAR

VAR_OUTPUT
  QO: BOOL; (* 1=Fonctionnement normal, 0= Fonctionnement anormal *)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID) -> SERVER.initTask(ID) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID) -> SERVER.init(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_task
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateTask(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.taskTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```



61499-1 © CEI:2012 – 217 –




注：表D.5中列出的其他服务序列的图形表示类似于图D.2。

**图D.5 TASK功能块的类型**

**表D.5 TASK功能块类型源代码**

```
FUNCTION_BLOCKTASK(*触发IEC61131任务*)

用QI ID初始化；(*初始化停止服务*)
要求QI； (*服务请求*)
END_EVENT


  INITO WITH QO,STATUS; (* Initialiser/Arrêter «Confirm» *)
带QO、状态的CNF；(*请求服务确认*)
END_EVENT


  QI: BOOL; (* Qualificateur d'entrée d'événements *)
编号：WSTRING；(*要触发的任务路径*)


VAR_OUTPUT
QO：布尔值； (*1=正常运行，0=异常运行*)
  STATUS: INT;
END_VAR

SERVICE CLIENT/SERVER
SEQUENCE normal_establishment
  CLIENT.INIT+(ID) -> SERVER.initTask(ID) -> CLIENT.INITO+();
END_SEQUENCE

SEQUENCE unsuccessful_establishment
  CLIENT.INIT+(ID) -> SERVER.init(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE request_task
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF+();
END_SEQUENCE

SEQUENCE request_inhibited
  CLIENT.REQ-() -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE request_error
  CLIENT.REQ+(ID) -> SERVER.reqTask(ID) -> CLIENT.CNF-(STATUS);
END_SEQUENCE

SEQUENCE client_initiated_termination
  CLIENT.INIT-() -> SERVER.terminateTask(ID) -> CLIENT.INITO-(STATUS);
END_SEQUENCE

SEQUENCE server_initiated_termination
  SERVER.taskTerminated(ID,STATUS) -> CLIENT.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
```

### D.6.4 Conformité

Les spécifications données dans l'Annexe D peuvent être référencées dans les profils de conformité selon les règles données dans la CEI 61499-4.

Lorsqu'un système de dispositif de commande programmable conforme à la CEI 61131-3 prend en charge l'interopérabilité avec un ou plusieurs types de bloc fonctionnel de la CEI 61499 définis dans l'Annexe D, il convient qu'il inclue dans sa liste de caractéristiques prises en charge une référence aux caractéristiques prises en charge issues du Tableau D.6, et il convient qu'il inclue des spécifications des valeurs pour les caractéristiques et paramètres spécifiques à une mise en œuvre tels que définis respectivement en 8.1 et en 8.2 de la CEI 61131-5:2000.

**Tableau D.6 – Caractéristiques d'interopérabilité de la CEI 61499**

| No. | Description |
|---|---|
| 1 | Type de bloc fonctionnel READ |
| 2 | Type de bloc fonctionnel UREAD |
| 3 | Type de bloc fonctionnel WRITE |
| 4 | Type de bloc fonctionnel TASK |





根据IEC61499-4中给出的规则，可以在符合性配置文件中引用附录D中给出的规范。

当符合IEC61131-3的可编程控制器系统支持与附录D中定义的一种或多种IEC61499功能块类型的互操作性时，它应在其支持的功能列表中包括对表D.6中支持的功能的引用，并应包括规范分别在IEC61131-5:2000的8.1和8.2中定义的实现特定功能和参数的值。

**表D.6 IEC61499互操作特性**

| No. | |
|---|---|
| 1 | 功能块类型READ |
| 2 | 功能块类型UREAD |
| 3 | 功能块类型WRITE |
| 4 | TASK功能块类型 |

# Annexe E
(informative)

# Echange d'informations

## E.1 Utilisation des moyens de la couche application

Le paragraphe 7.1.3.2 de l'ISO/CEI 7498-1:1994 identifie un certain nombre de moyens fournis par des *entités d'application* (c'est-à-dire: *entités* de la *couche application*) pour permettre à des *processus d'application* d'échanger des informations. Pour fournir ces moyens, les entités d'application utilisent des *protocoles d'application* et des *services de présentation.* Les blocs fonctionnels de communication définis à l'Article E.2 peuvent utiliser ces moyens, lorsqu'ils sont fournis par des entités d'application appropriées, de la façon suivante:

a) Les blocs fonctionnels de communication utilisent les moyens de *transfert d'informations* fournis par des entités d'application pour assurer la *synchronisation d'applications qui coopèrent*, représentées par les événements `REQ`, `CNF`, `IND` et `RSP` et transférer les données représentées par des entrées `SD` et des sorties `RD`.

b) Les moyens suivants peuvent être utilisés au cours de l'initialisation du service par les événements `INIT` et `INITO`, en utilisant les éléments de la structure de données `PARAMS` selon les besoins:

- identification des partenaires de communications attendus;
- détermination de la qualité acceptable du service;
- accord sur la responsabilité de la reprise sur erreur;
- accord sur des aspects de sécurité;
- identification de la syntaxe abstraite.

c) Les moyens servant à la *sélection du mode de dialogue* peuvent être utilisés par les types spécifiques de bloc fonctionnel, par exemple, par un `SUBSCRIBER` (c'est-à-dire: abonné) pour s'assurer qu'il interagit correctement avec un `PUBLISHER` (c'est-à-dire: éditeur).

Beaucoup des moyens énumérés ci-dessous ne sont pas fournis par des entités d'application des systèmes de mesure et commande dans les processus industriels (les IPMCS). Dans ce cas, les blocs fonctionnels de communication doivent mettre en œuvre des moyens équivalents pour fournir les services requis.

En particulier, les services de présentation ne sont parfois pas fournis par les entités d'application IMPCS. Par conséquent, afin de faciliter la mise en œuvre de ces services par des blocs fonctionnels de communication, les syntaxes de transfert tant pour le transfert d'informations que pour la gestion d'application sont définies à l'Article E.3.

NOTE 1 Voir l'ISO/CEI 7498-1 pour des définitions de termes utilisés dans cette annexe, mais pas définis dans la présente norme.

NOTE 2 Une *ressource* est un «processus d'application» tel que défini dans l'ISO/CEI 7498-1.

NOTE 3 Le contenu de l'Annexe E pourrait être considéré normatif par le fait que les profils de conformité tels que définis dans la CEI 61499-4, dans d'autres normes et spécifications sont susceptibles de spécifier un contexte dans lequel l'ensemble ou une partie de ses dispositions sont employés.



61499-1 © CEI:2012 – 219 –

Annexe E
(informative)

Echange d'informations

## 使用应用层手段

ISOIEC7498-1:1994的子条款7.1.3.2确定了应用实体（即：应用层实体）提供的许多方法，以使应用程序能够交换信息。为了提供这些手段，应用实体使用应用协议和表示服务。当由适当的应用实体提供时，条款E.2中定义的通信功能块可以使用这些方法，如下所示：

a)通信功能块使用应用实体提供的信息传输手段来确保协作应用程序的同步，由事件REQ、CNF、IND和RSP表示，并传输由SD输入和RD输出表示的数据。

b)在INIT和INITO事件的服务初始化期间，可以使用以下方法，根据需要使用PARAMS数据结构的元素：

- 确定预期的沟通伙伴；
- 确定可接受的服务质量；
- 错误恢复责任协议；
- 安全方面的协议；

  抽象语法的识别。

c)特定类型的功能块可以使用对话模式选择的手段，例如，订阅者（即：订阅者）以确保它与发布者（即：发布者）正确交互。

下面列出的许多设施不是由工业过程测量和控制系统应用实体(IPMCS)提供的。在这种情况下，通信功能块必须实现等效的方法来提供所需的服务。

特别是，IMPCS应用实体有时不提供表示服务。因此，为了便于通过通信功能块实现这些服务，在第E.3节中定义了信息传输和应用程序管理的传输语法。

注1：参见ISOIEC7498-1，了解本附录中使用但未在本标准中定义的术语的定义。

注2资源是ISOIEC7498-1中定义的"应用程序过程"。

注3：附录E的内容可以被认为是规范性的，因为IEC61499-4中定义的一致性配置文件，其他标准和规范可能会指定使用组件或其部分条款的上下文。

## E.2 Types de bloc fonctionnel de communication

### E.2.1 Généralités

Ce Paragraphe définit des *types de bloc fonctionnel de communication* génériques pour les transactions *unidirectionnelles* et *bidirectionnelles*. Il convient que les personnalisations **dépendantes de la mise en œuvre** de ces types respectent les règles suivantes:

a) la mise en œuvre doit spécifier les types de données et la sémantique des valeurs des entrées de données et des sorties de données de chaque type de bloc fonctionnel de ce genre;

b) la mise en œuvre doit spécifier le traitement du transfert anormal de données;

c) la mise en œuvre doit spécifier toutes les éventuelles différences entre le comportement d'instances de ces types de bloc fonctionnel et les comportements spécifiés à l'Article E.2.

### E.2.2 Blocs fonctionnels pour les transactions unidirectionnelles

Les Figures E.1 à E.4 donnent des déclarations de types et des séquences de primitives de service typiques des blocs fonctionnels qui assurent des *transactions unidirectionnelles* sur une *connexion de communication*. Une telle connexion consiste en une *instance* `PUBLISH` et une ou plusieurs instances du type `SUBSCRIBE`.

NOTE 1 Des spécifications textuelles complètes de ces types de bloc fonctionnel ne sont pas données dans l'Annexe F.

NOTE 2 Les types de données et la sémantique de l'entrée `PARAMS` et de la sortie `STATUS` sont **dépendants de la mise en œuvre**.

NOTE 3 Le nombre (`m`) et les types des données reçues `RD_1,…,RD_m` correspondent au nombre et aux types des données émises `SD_1,…,SD_m`.

NOTE 4 Les moyens par lesquels les connexions de communications sont établies ne relèvent pas du domaine d'application de la présente norme.

NOTE 5 Le transfert de données pourrait être requis afin de déterminer si, oui ou non, `RD_1,…,RD_m` respectent les contraintes exprimées dans la Note 3.

NOTE 6 Les syntaxes de transfert définies à l'Article E.3 sont susceptibles d'être utilisées pour effectuer la détermination décrite dans la Note 5.

NOTE 7 Le traitement du transfert anormal de données est **dépendant de la mise en œuvre.**

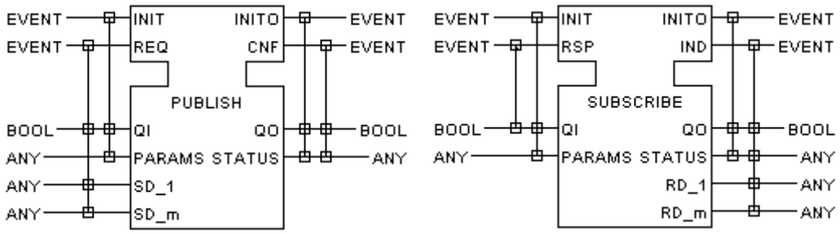



**Figure E.1 – Spécifications du type pour les transactions unidirectionnelles**





## E.2 通讯功能块类型

本节定义了单向和双向事务的通用通信功能块类型。这些类型的依赖于实现的定制应遵守以下规则：

a)实现必须指定此类功能块的每种类型的数据输入和数据输出的数据类型和值语义；

b)实现必须指定异常数据传输的处理；

c)实现必须指定这些功能块类型实例的行为与第E.2条中规定的行为之间的任何差异。

**单向交易的功能块**

图E.1到E.4为支持通过通信连接的单向事务的功能块提供了典型的类型声明和服务原语序列。这样的连接由一个PUBLISH实例和一个或多个SUBSCRIBE类型的实例组成。

注1这些功能块类型的全文规范未在附录F中给出。

**注2：PARAMS输入和STATUS输出的数据类型和语义取决于实现。**

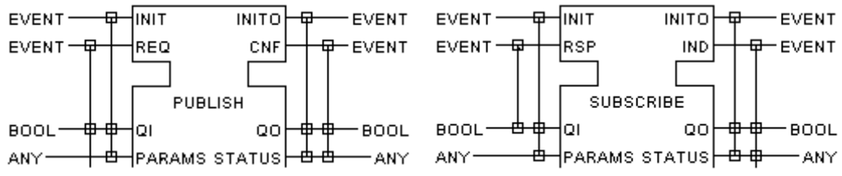
注3接收数据RD_1 … RD_m的数量(m)和类型对应于传输数据SD_1 … SD_m的数量和类型。

注4：建立通信连接的方法不在本标准的范围内。

注5可能需要数据传输来确定RD_1 … RD_m是否满足注3中表达的约束。

注6条款E.3中定义的传输语法可用于执行注5中描述的确定。

**注7异常数据传输的处理取决于实现。**




**图E.1 单向交易的类型规范**

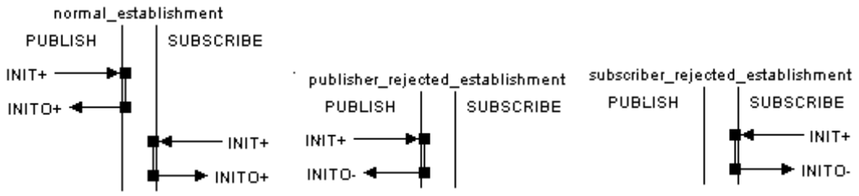



**Figure E.2 – Établissement de connexion pour les transactions unidirectionnelles**

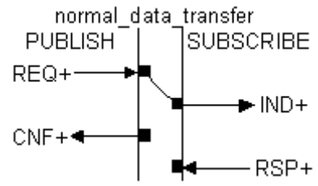



**Figure E.3 – Transfert unidirectionnel normal de données**

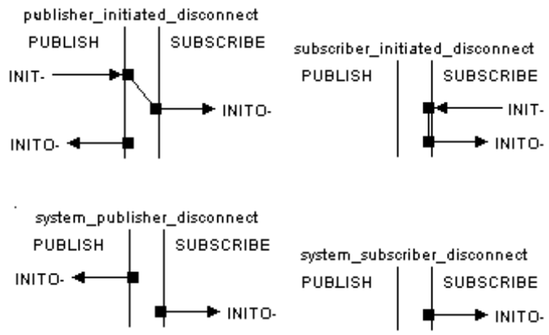



**Figure E.4 – Libération de connexion pour le transfert unidirectionnel de données**

### E.2.3 Blocs fonctionnels pour les transactions bidirectionnelles

Les Figures E.5 à E.8 donnent des déclarations de types et des séquences de primitives de service des blocs fonctionnels qui assurent des *transactions bidirectionnelles* sur une *connexion de communication*. Une telle connexion comprend une instance de type CLIENT et une instance de type SERVER.

NOTE 1 Des spécifications textuelles complètes de ces types de bloc fonctionnel ne sont pas données dans l'Annexe F.

NOTE 2 Les types de données et la sémantique de l'entrée PARAMS et de la sortie STATUS sont **dépendants de la mise en œuvre**.

NOTE 3 Le nombre (m) et les types des données reçues `RD_1,…,RD_m` correspondent au nombre et aux types des données émises `SD_1,…,SD_m`.

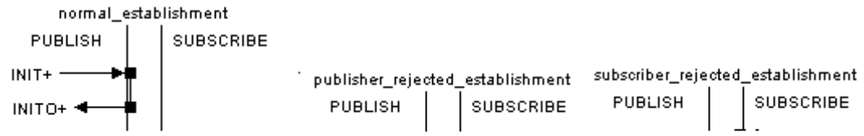
NOTE 4 Le nombre (n) et les types des données reçues `RD_1,…,RD_n` correspondent au nombre et aux types des données émises `SD_1,…,SD_n`.

NOTE 5 Le transfert de données peut être requis afin de déterminer si, oui ou non, `RD_1,…,RD_m` et `RD_1,…,RD_n` respectent les contraintes exprimées dans les Notes 3 et 4.

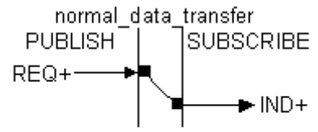
NOTE 6 Les syntaxes de transfert définies à l'Article E.3 peuvent être utilisées pour effectuer la détermination décrite dans la Note 5.



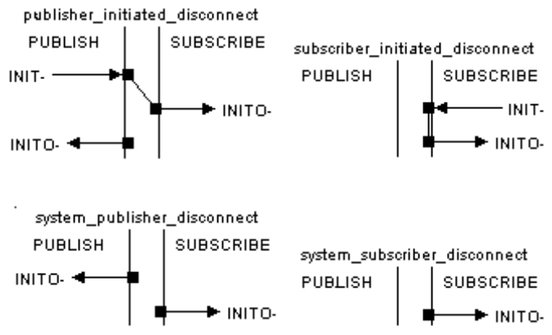
61499-1 © CEI:2012 – 221 –




图E.2 单向事务的连接建立




**图E.3 正常的单向数据传输**




**图E.4 单向数据传输的连接释放**

**双向交易功能块**

图E.5到E.8提供了支持通信连接上的双向事务的功能块的类型声明和服务原语序列。这样的连接包括一个CLIENT类型的实例和一个SERVER类型的实例。

注1这些功能块类型的全文规范未在附录F中给出。

**注2：PARAMS输入和STATUS输出的数据类型和语义取决于实现。**

注3接收数据RD_1 … RD_m的数量(m)和类型对应于传输数据SD_1 … SD_m的数量和类型。

注4接收数据RD_1 … RD_n的数量(n)和类型对应于传输数据SD_1 … SD_n的数量和类型。

注5可能需要数据传输来确定是否RD_1、…、RD_m和
RD_1 … RD_n遵守注释3和4中表达的约束。

注6条款E.3中定义的传输语法可用于执行注5中描述的确定。

NOTE 7 Le traitement du transfert anormal de données est **dépendant de la mise en œuvre.**

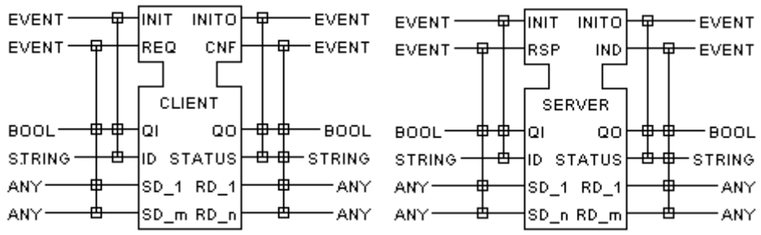



**Figure E.5 – Spécifications du type pour les transactions bidirectionnelles**

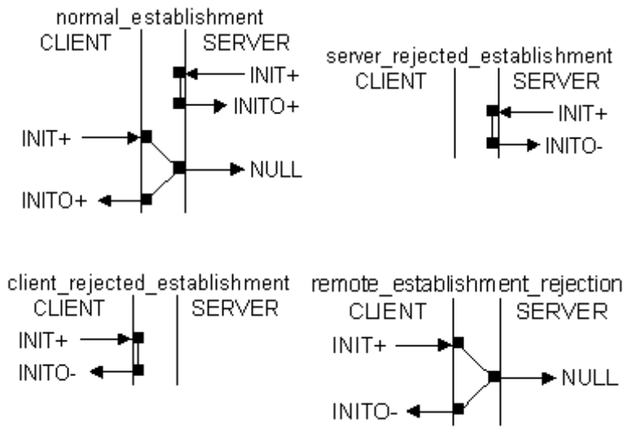



**Figure E.6 – Établissement de connexion pour une transaction bidirectionnelle**

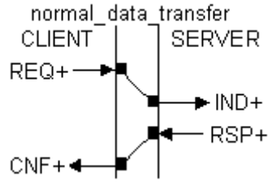



**Figure E.7 – Transfert de données bidirectionnel**

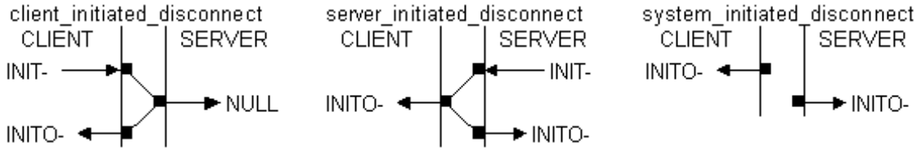



**Figure E.8a – Déclenchée par le client**

**Figure E.8b – Déclenchée par le serveur**

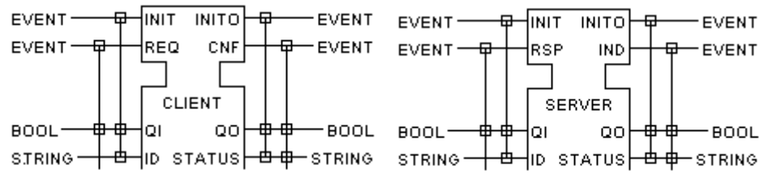
**Figure E.8c – Déclenchée par le système**

**Figure E.8 – Libération de connexion dans le transfert bidirectionnel de données**

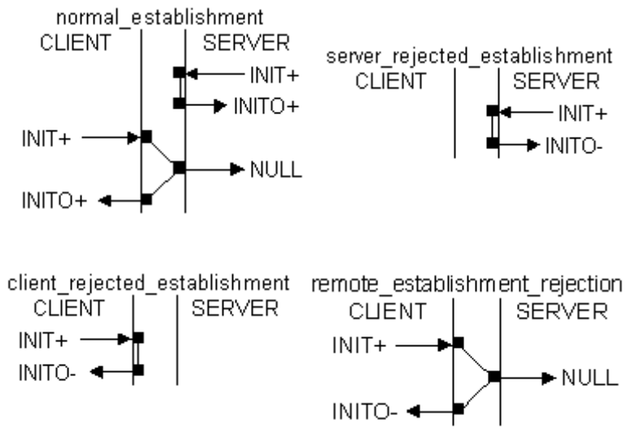




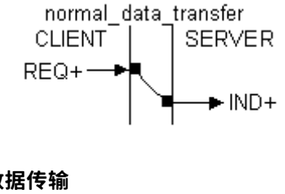
注7异常数据传输的处理取决于实现。



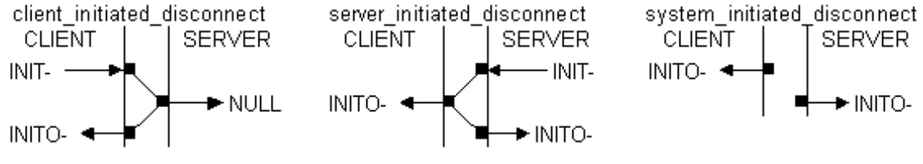

**图E.5 双向交易的类型规范**




**图E.6 双向事务的连接建立**




**图E.7 双向数据传输**




**图E.8a 客户触发**

**图E.8b 服务器触发**

**图E.8c 系统触发**

**图E.8 双向数据传输中的连接释放**

## E.3 Syntaxe de transfert

### E.3.1 Contexte

Une syntaxe de transfert est définie en termes de *syntaxe abstraite* décrivant les types des données devant être transférées et un jeu de *règles de codage* pour la représentation codée d'instances des types de données ainsi définis.Le paragraphe E.3.2 utilise la Notation de syntaxe abstraite numéro un (ASN.1), comme défini dans l'ISO/CEI 8824-1, pour définir la syntaxe IEC61499-FBDATA pour le transfert de données.

Deux jeux de règles de codage sont donnés dans l'Annexe E:

a) Le paragraphe E.3.3.1 définit des règles de codage BASIC, en utilisant les règles définies dans l'ISO/CEI 8825-1.

b) Le paragraphe E.3.3.2 utilise les caractéristiques spéciales des types de données dans la syntaxe IEC61499-FBDATA pour obtenir un jeu de règles de codage COMPACT selon les principes suivants:

- Lorsque le nombre des «octets de contenu» est fixe, les «octets de longueur» ne sont pas utilisés dans le codage.
- Des codages spéciaux sont utilisés pour réduire au maximum le nombre d'octets et l'effort de codage/décodage requis pour les types de longueur fixe.
- Les «octets d'identificateur» ne sont pas utilisés pour les éléments individuels des types de données STRUCT et ARRAY, car le type de chaque élément est fixé dans la *déclaration de types* correspondante.

### E.3.2 Syntaxe abstraite IEC61499-FBDATA

La syntaxe de transfert obtenue en appliquant les règles de codage COMPACT en E.3.3.2 à la syntaxe abstraite en E.3.2 est recommandée pour:

- transférer des valeurs des entrées SD d'un *bloc fonctionnel de communication* jusqu'aux sorties RD du/des bloc(s) fonctionnel(s) de communication à l'extrémité opposée d'une *connexion de communication*;
- déterminer si, oui ou non, les contraintes sur le nombre et le type correspondants de variables entre les entrées SD et les sorties RD sont satisfaites comme noté dans les Figures E.1 et E.5.

L'utilisation de la syntaxe abstraite définie en E.3.2 pour le transfert de données exprimées sous forme de *libellés* et de valeurs de *variables* assujettie aux règles sémantiques suivantes (RULES):

a) Lorsque le nom d'un type de données dans ce module (par exemple, BOOL) correspond au nom d'un type de données défini dans la CEI 61131-3, la définition de type donnée est destinée au transfert de données du type de données correspondant de la CEI 61131-3.

b) Les valeurs de «VisibleString» pour les types de données DATE et TIME_OF_DAY sont restreintes à la syntaxe textuelle pour ces types de données tels que définis dans la CEI 61131-3.

c) La notation [typeID] implique que l'étiquette des données consiste en la valeur de l'étiquette ASN.1 du type de données dérivé correspondant, établi comme spécifié dans l'Annexe A de la CEI 61499-2:2005 par un autre moyen ne relevant pas du domaine d'application de la présente norme.

d) La valeur d'un élément EnumeratedData consiste en la position cardinale (débutant à zéro) de l'identificateur correspondant dans la séquence d'identificateurs définis pour le type de données «enumerated» (énuméré) correspondant, établi comme spécifié dans la CEI 61131-3.

e) Le type spécifique d'un élément SubrangeData est comme pour son *type de données «subrange»* particulier, déclaré comme spécifié dans la CEI 61131-3.



61499-1 © CEI:2012 – 223 –

## E.3 Syntaxe de transfert

传输语法是根据描述要传输的数据类型的抽象语法和一组用于如此定义的数据类型实例的编码表示的编码规则来定义的。第E.3.2节使用语法符号抽象号一（ASN.1），如ISOIEC8824-1中所定义，定义用于数据传输的IEC61499-FBDATA语法。

附录E给出了两组编码规则：

a)使用ISOIEC8825-1中定义的规则，子条款E.3.3.1定义了BASIC编码规则。

b)子条款E.3.3.2使用IEC61499-FBDATA语法中数据类型的特殊特性，根据以下原则获得COMPACT编码规则集：

- 当"内容字节"的数量固定时，编码中不使用"长度字节"。
- 特殊编码用于最小化字节数和固定长度类型所需的编码/解码工作。
- "标识符字节"不用于STRUCT和ARRAY数据类型的单个元素，因为每个元素的类型在相应的类型声明中是固定的。

通过将E.3.3.2中的COMPACT编码规则应用于E.3.2中的抽象语法获得的传输语法推荐用于：

- 将值从通信功能块的SD输入传输到通信连接另一端的通信功能块的RD输出。
- 确定是否满足SD输入和RD输出之间变量的相应数量和类型的约束，如图E.1和E.5所示。

使用E.3.2中定义的抽象语法传输以标签和变量值形式表示的数据，需遵守以下语义规则（RULES）：

a)当此模块中的数据类型名称（例如BOOL）与IEC61131-3中定义的数据类型名称匹配时，给定类型定义用于传输IEC61131-3中对应数据类型的数据.

b)DATE和TIME_OF_DAY数据类型的"VisibleString"值仅限于IEC61131-3中定义的这些数据类型的文本语法。

c)[typeID]符号表示数据标签由相应派生数据类型的ASN.1标签值组成，按照IEC61499-2:2005的附录A中的规定通过不属于这个标准。

d)EnumeratedData元素的值由为相应"枚举"数据类型定义的标识符序列中相应标识符的基数位置（从零开始）组成，按照IEC61131-3的规定建立。

e)SubrangeData元素的特定类型与其特定的"子范围"数据类型相同，声明为IEC61131-3中的规定。

f) Le type des éléments d'un élément de données ARRAY est établi comme spécifié pour les *types de données «array»* dans la CEI 61131-3.

g) Les types des éléments d'un élément de données STRUCT sont établis comme spécifié pour les *types de données structurées* dans la CEI 61131-3.

**ASN.1 MODULE**

```
IEC61499-FBDATA DEFINITIONS::=

BEGIN

EXPORTS FBDataSequence, FBData, ElementaryData, BOOL, FixedLengthInteger,
     FixedLengthReal, TIME, AnyDate, AnyString, FixedLengthBitString,
     SignedInteger, UnsignedInteger, REAL, LREAL, DATE, TIME_OF_DAY,
     DATE_AND_TIME, STRING, WSTRING, BYTE, WORD, DWORD, LWORD,
     DirectlyDerivedData, EnumeratedData, SubrangeData, ARRAY, STRUCT;

FBDataSequence::= [APPLICATION 23] IMPLICIT SEQUENCE OF FBData

FBData::= CHOICE{ElementaryData, DerivedData}

ElementaryData::= CHOICE{
     BOOL,
     FixedLengthInteger,
     FixedLengthReal,
     TIME,
     AnyDate,
     AnyString,
     FixedLengthBitString}

FixedLengthInteger::= CHOICE{SignedInteger, UnsignedInteger}

SignedInteger::= CHOICE{SINT, INT, DINT, LINT}

UnsignedInteger::= CHOICE{USINT, UINT, UDINT, ULINT}

FixedLengthReal::= CHOICE{REAL, LREAL}

AnyDate::= CHOICE{DATE, TIME_OF_DAY, DATE_AND_TIME}

AnyString::= CHOICE{STRING, WSTRING}

FixedLengthBitString::= CHOICE{BYTE, WORD, DWORD, LWORD}

BOOL::= CHOICE{BOOL0, BOOL1}

BOOL0::= [APPLICATION 0] IMPLICIT NULL

BOOL1::= [APPLICATION 1] IMPLICIT NULL

SINT::= [APPLICATION 2] IMPLICIT INTEGER(-128..127)

INT::= [APPLICATION 3] IMPLICIT INTEGER(-32768..32767)

DINT::= [APPLICATION 4] IMPLICIT INTEGER(-2147483648..2147483647)

LINT::= [APPLICATION 5]
     IMPLICIT INTEGER(-9223372036854775808..9223372036854775807)

USINT::= [APPLICATION 6] IMPLICIT INTEGER(0..255)

UINT::= [APPLICATION 7] IMPLICIT INTEGER(0..65535)

UDINT::= [APPLICATION 8] IMPLICIT INTEGER(0..4294967295)

ULINT::= [APPLICATION 9] IMPLICIT INTEGER(0..18446744073709551615)

REAL::= [APPLICATION 10] IMPLICIT OCTET STRING (SIZE(4))

LREAL::= [APPLICATION 11] IMPLICIT OCTET STRING (SIZE(8))

TIME::= [APPLICATION 12] IMPLICIT LINT – Durée en unités de 1µs

DATE::= [APPLICATION 13] IMPLICIT ULINT – Voir Tableau E.1.
```



ARRAY数据元素的元素类型按照IEC61131-3中对数组数据类型的规定进行设置。

g)STRUCT数据元素的元素类型按照IEC61131-3中结构化数据类型的规定建立。

**ASN.1 MODULE**

```
IEC61499-FBDATA DEFINITIONS::=

BEGIN

EXPO                                                              ger,
```

有符号整数、无符号整数、实数、实数、日期、TIME_OF_DAY、
DATE_AND_TIME STRING WSTRING BYTE WORD DWORD LWORD

```
                                                                  TRUCT;
```

FBDataSequence::=[应用23]FBData的隐式序列

```
FBDataSequence::= [APPLICATION 23] IMPLICIT SEQUENCE OF FBData

FBData::= CHOICE{ElementaryData, DerivedData}

ElementaryData::= CHOICE{
     BOOL,
     FixedLengthInteger,
     FixedLengthReal,
     TIME,
     AnyDate,
     AnyString,
     FixedLengthBitString}

                                                        ignedInteger}
```

SignedInteger::=CHOICE{SINT INT DINT LINT

```
UnsignedInteger::= CHOICE{USINT, UINT, UDINT, ULINT}

FixedLengthReal::= CHOICE{REAL, LREAL}

AnyDate::= CHOICE{DATE, TIME_OF_DAY, DATE_AND_TIME}

AnyString::= CHOICE{STRING, WSTRING}

FixedLengthBitString::= CHOICE{BYTE, WORD, DWORD, LWORD}
```

BOOL0::=[应用程序0]隐式NULL

BOOL1::=[应用程序1]隐式NULL

```
SINT::= [APPLICATION 2] IMPLICIT INTEGER(-128..127)

INT::= [APPLICATION 3] IMPLICIT INTEGER(-32768..32767)

DINT::= [APPLICATION 4] IMPLICIT INTEGER(-2147483648..2147483647)

LINT::= [APPLICATION 5]
     IMPLICIT INTEGER(-9223372036854775808..9223372036854775807)

USINT::= [APPLICATION 6] IMPLICIT INTEGER(0..255)

UINT::= [APPLICATION 7] IMPLICIT INTEGER(0..65535)

UDINT::= [APPLICATION 8] IMPLICIT INTEGER(0..4294967295)

                                                              09551615)
```

REAL::=[应用程序10]隐式八位字节字符串（大小（4））

LREAL::=[应用程序11]隐式八位字节字符串（大小（8））

TIME::=[应用12]隐式LINT 时间，单位为1µs

DATE::=[应用13]隐式ULINT 见表E.1。

```
TIME_OF_DAY::= [APPLICATION 14] IMPLICIT ULINT – Voir Tableau E.1.

DATE_AND_TIME::= [APPLICATION 15] IMPLICIT ULINT – Voir Tableau E.1.

STRING::= [APPLICATION 16] IMPLICIT OCTET STRING – 1 octet par caractère

BYTE::= [APPLICATION 17] IMPLICIT BIT STRING (SIZE(8))

WORD::= [APPLICATION 18] IMPLICIT BIT STRING (SIZE(16))

DWORD::= [APPLICATION 19] IMPLICIT BIT STRING (SIZE(32))

LWORD::= [APPLICATION 20] IMPLICIT BIT STRING (SIZE(64))

WSTRING::= [APPLICATION 21] IMPLICIT OCTET STRING – 2 octets par caractère

DerivedData::= CHOICE{
     DirectlyDerivedData,
     EnumeratedData,
     SubrangeData,
     ARRAY,
     STRUCT}

DirectlyDerivedData::= [typeID] IMPLICIT ElementaryData

EnumeratedData::= [typeID] IMPLICIT UINT

SubrangeData::= [typeID] IMPLICIT FixedLengthInteger

ARRAY::= CHOICE {ArrayVariable, TypedArray}

ArrayVariable::= [APPLICATION 22] IMPLICIT FBDataSequence – même type

TypedArray::= [typeID] IMPLICIT FBDataSequence – même type

STRUCT::= [typeID] IMPLICIT SEQUENCE – types différents

END
```

### E.3.3 Règles de codage

#### E.3.3.1 Codage BASIC

Ce codage doit être le résultat obtenu en appliquant les règles de codage de base selon l'ISO/CEI 8825-1 à des variables des types définis en E.3.2.

#### E.3.3.2 Codage COMPACT

Ce codage doit être le résultat obtenu en modifiant les règles pour le codage BASIC données en E.3.3.1 comme suit:

a) les «octets de longueur» ne doivent pas être inclus dans le codage des valeurs des types de données montrés au Tableau E.1;

b) la longueur (en octets) et le codage des «octets de contenu» décrits dans l'ISO/CEI 8825-1 doivent être tels que définis dans le Tableau E.1 pour les valeurs des types de données qui y sont montrés;

c) le codage des variables de type `TIME`, `DirectlyDerivedData`, `EnumeratedData` ou `SubrangeData` doit suivre les mêmes règles de codage que le type de base.

d) les «octets de typ» ne doivent pas être inclus dans le codage d'éléments individuels de type `STRUCT`, à l'exception du codage d'éléments de type `BOOL`, qui doivent être codés suivant la règle (1) du Tableau E.1;

e) le codage des valeurs des types `STRING` et `WSTRING` doit être primitif;

f) le codage des éléments `ARRAY` doit être *construit* dans le sens de l'ISO/CEI 8825-1, avec les dispositions suivantes pour le codage COMPACT:



TIME_OF_DAY::=[应用程序14]隐式ULINT 见表E.1。

DATE_AND_TIME::=[应用程序15]隐式ULINT 见表E.1。

STRING::=[应用程序16]隐式八位字节串——每个字符1个字节

BYTE::=[应用程序17]隐式位串（大小（8））

WORD::=[应用程序18]隐式位串（大小（16））

DWORD::=[应用程序19]隐式位串（大小（32））

LWORD::=[应用程序20]隐式位串（大小（64））

WSTRING::=[应用21]隐式八位字节串——每个字符2个字节

```
DerivedData::= CHOICE{
     DirectlyDerivedData,
     EnumeratedData,
     SubrangeData,
     ARRAY,
     STRUCT}

DirectlyDerivedData::= [typeID] IMPLICIT ElementaryData

EnumeratedData::= [typeID] IMPLICIT UINT

SubrangeData::= [typeID] IMPLICIT FixedLengthInteger
```

ArrayVariable::=[应用22]隐式FBDataSequence–相同类型

TypedArray::=[typeID]IMPLICITFBDataSequence 相同类型

STRUCT::=[typeID]隐式序列–不同的类型

```
END
```

### E.3.3 Règles de codage

该编码应是通过将根据ISOIEC8825–1的基本编码规则应用于E.3.2中定义的类型的变量而获得的结果。

该编码应是通过修改E.3.3.1中给出的BASIC编码规则获得的结果，如下所示：

a)“长度八位字节”不应包含在表E.1所示数据类型的值的编码中；

b)ISOIEC8825–1中描述的“内容八位字节”的长度（以八位字节为单位）和编码应如表E.1中针对其中显示的数据类型的值所定义；

c)TIME、DirectlyDerivedData、EnumeratedData或SubrangeData类型的变量的编码必须遵循与基本类型相同的编码规则。

d)“典型字节”不应包含在STRUCT类型的单个元素的编码中，但BOOL类型的元素的编码除外，它应根据表E.1的规则(1)进行编码；

e)STRING和WSTRING类型的值的编码应该是原始的；

f) ARRAY元件的编码必须按照ISOIEC8825–1的意义构建，并针对COMPACT编码提供以下规定：

1) Le sous-champ «longueur» de l'élément ARRAY doit être codé comme une valeur du type UINT sans octets d'identificateur ou de longueur, c'est-à-dire comme un nombre entier non signé («unsigned») de 16 bits.

   NOTE 1 Cela semble réduire le nombre maximal d'éléments d'un ARRAY à 65535. Cependant, la longueur réelle peut être réduite encore davantage par le nombre maximal d'octets pouvant être transféré par le protocole de transport sous-jacent.

   EXEMPLE Pour les messages UDP possédant un nombre maximal de 65508 octets, la longueur maximale transmissible d'un ARRAY d'éléments BYTE serait égale à: (nombre maximal d'octets – octets d'étiquette – octets de longueur – octets de type d'élément)/(longueur d'élément) = (65508-1-2-1)/1 = 65504 éléments.

2) Le codage COMPACT doit être utilisé pour le premier élément du champ «valeurs».

3) Les éléments suivants, le cas échéant, doivent être codés en utilisant la syntaxe COMPACT sans sous-champ «identificateur», sauf pour les éléments de type BOOL, qui doivent être codés suivant la règle (1) du Tableau E.1.

4) Si la longueur spécifiée de l'élément ARRAY reçu est inférieure à l'espace alloué localement, les éléments restants de la matrice locale sont inchangés; si la longueur de l'élément ARRAY reçu est supérieure à l'espace alloué localement, les éléments reçus restants sont ignorés.

NOTE 2 Etant donné que ARRAY est une sous-classe de FBData, un élément ARRAY multidimensionnel est susceptible d'être codé de façon récurrente en tant qu'ARRAY dont les éléments sont des éléments ARRAY.



1)ARRAY元素的"长度"子字段必须编码为不带标识符或长度字节的UINT类型的值，即16位的无符号整数。

注1：这似乎将ARRAY中的最大元素数减少到65535。但是，实际长度可能会进一步减少底层传输协议可以传输的最大字节数。

示例对于最大数量为65508字节的UDP消息，BYTE元素数组的最大可传输长度将等于：（最大字节数-标签字节-长度字节-数据类型字节元素）（元素长度）=(65508-1-2-1)1=65504个元素。

2)COMPACT编码应该用于"values"字段的第一个元素。

3)以下元素（如果有）应使用不带"标识符"子字段的COMPACT语法进行编码，但类型为BOOL的元素除外，其应根据表E.1的规则(1)进行编码。

4）如果接收到的ARRAY元素的指定长度小于本地分配的空间，则本地数组的剩余元素不变；如果接收到的ARRAY元素的长度大于本地分配的空间，则丢弃剩余的接收到的元素。

注2由于ARRAY是FBData的子类，因此多维ARRAY元素很可能被递归地编码为元素是ARRAY元素的ARRAY。

**Tableau E.1 – Codage COMPACT des types de données de longueur fixe**

| Type de données | Octets de contenu | |
|---|---|---|
| | Longueur | Règle de codage |
| BOOL | 0 | (1) |
| SINT | 1 | (2) |
| INT | 2 | (2) |
| DINT | 4 | (2) |
| LINT | 8 | (2) |
| USINT | 1 | (3) |
| UINT | 2 | (3) |
| UDINT | 4 | (3) |
| ULINT | 8 | (3) |
| REAL | 4 | (4) |
| LREAL | 8 | (4) |
| DATE | 8 | (5) |
| TIME | 8 | (7) |
| TIME_OF_DAY | 12 | (5) |
| DATE_AND_TIME | 20 | (5) |
| BYTE | 1 | (6) |
| WORD | 2 | (6) |
| DWORD | 4 | (6) |
| LWORD | 8 | (6) |

RÈGLES DE CODAGE POUR LE TABLEAU E.1

(1) Les valeurs de ce type de données doivent être codées en un seul octet d'identificateur contenant le codage d'étiquette pour la classe `BOOL0` ou `BOOL1`, défini en E.3.2, correspondant respectivement aux valeurs de `FALSE (0)` ou `TRUE (1)`.

(2) Les valeurs de ces types de données `SignedInteger` doivent être codées de la même manière qu'un `UnsignedInteger` de même longueur que le type `SignedInteger` avec une valeur $N - N_{min}$, où N est la valeur de la variable `SignedInteger` à coder et $N_{min}$ est l'extrémité inférieure de la plage de valeurs du sous-type `SignedInteger` tel que défini en E.3.2.

(3) Les valeurs de ces types de données `UnsignedInteger` doivent être codées en numérotant les bits dans les octets de contenu, en commençant par le bit 1 du dernier octet comme bit zéro et en terminant la numérotation par le bit 8 du premier octet. Chaque bit se voit attribuer une valeur de $2^N$, où N est sa position dans la séquence de numérotation ci-dessus. La valeur du nombre entier non signé («unsigned») est obtenue en additionnant les valeurs numériques attribuées à chaque bit pour les bits qui sont mis à un.

(4) Les valeurs de ces types de données doivent être codées comme des nombres au format simple 32 bits et au format double 64 bits comme défini dans la CEI 60559, où «lsb» défini dans la CEI 60559 correspond au «bit zéro» tel que défini dans la Règle(3).

(5) Les valeurs de ces types doivent être codées comme dans le cas du type `ULINT`, représentant le nombre de millisecondes écoulées depuis minuit pour `TIME_OF_DAY`, le nombre de millisecondes écoulées depuis le `1970-01-01-00:00:00.000` pour `DATE_AND_TIME`, le nombre de millisecondes écoulées à partir du `1970-01-01-00:00:00.000` jusqu'au `YYYY-MM-DD-00:00:00.000` pour `DATE`, où `YYYY-MM-DD` est la date courante.

(6) Le codage des valeurs de ces types de données `FixedLengthBitString` doit être primitif et doit être obtenu en plaçant les bits dans la chaîne de bits, en commençant par le premier bit et en poursuivant jusqu'au bit de queue, dans les bits 8 à 1 du premier octet de contenu, suivi tour à tour des bits 8 à 1 de chacun des octets suivants, la notation «premier bit» et «bit de queue» étant spécifiée dans l'ISO/CEI 8824-1.

(7) Le codage des valeurs de ce type de données doit être le même que celui des valeurs du type `LINT`, représentant un intervalle de temps en unités de 1 μs.





**表E.1 固定长度数据类型的COMPACT编码**

| 数据类型 | Octets de contenu | |
|---|---|---|
| | Longueur | Règle de codage |
| BOOL | 0 | (1) |
| SINT | 1 | (2) |
| INT | 2 | (2) |
| DINT | 4 | (2) |
| LINT | 8 | (2) |
| USINT | 1 | (3) |
| UINT | 2 | (3) |
| UDINT | 4 | (3) |
| ULINT | 8 | (3) |
| REAL | 4 | (4) |
| LREAL | 8 | (4) |
| DATE | 8 | (5) |
| TIME | 8 | (7) |
| TIME_OF_DAY | 12 | (5) |
| DATE_AND_TIME | 20 | (5) |
| BYTE | 1 | (6) |
| WORD | 2 | (6) |
| DWORD | 4 | (6) |
| | 8 | (6) |

表E.1的编码规则

(1)此数据类型的值应编码为单个标识符八位字节，其中包含
BOOL0或BOOL1类的标签编码，在E.3.2中定义，分别对应于FALSE(0)或TRUE(1)的值。

(2)这些SignedInteger数据类型的值必须以与a相同的方式编码
UnsignedInteger与SignedInteger类型长度相同，取值为N Nmin，其中N是要编码的SignedInteger变量的值，Nmin是E中定义的SignedInteger子类型的取值范围的下限。3.2.

（3）这些UnsignedInteger数据类型的值必须通过对位编号进行编码
在内容字节中，从最后一个字节的第1位开始作为第0位，并以第一个字节的第8位结束编号。每个位被分配一个值2N，其中N是它在上述编号序列中的位置。无符号整数的值是通过将分配给每个位的数值与设置为1的位相加得到的。

(4)这些数据类型的值应编码为IEC60559定义的32位单格式和64位双格式的数字，其中IEC60559定义的"lsb"对应定义的"零位"在规则（3）中。

(5)这些类型的值应该像ULINT类型的情况一样进行编码，表示
TIME_OF_DAY自午夜以来经过的毫秒数，自1970-01-01-00:00:00.000以来经过的毫秒数
对于DATE_AND_TIME，数字
对于DATE，从1970-01-01-00:00:00.000到YYYY-MM-DD00:00:00.000所经过的毫秒数，其中YYYY-MM_DD是当前日期。

(6)这些FixedLengthBitString数据类型的值的编码必须是原始的，必须是
通过将位放入位串中获得，从第一位开始并继续到尾随位，在第一个内容字节的位8到1中，然后依次是以下每个八位字节的位8到1，表示法ISOIEC8824-1中规定了"第一位"和"尾随位"。

(7)该数据类型的值的编码必须与LINT类型的值的编码相同，
表示以1μs为单位的时间间隔。

# Annexe F
(normative)

## Spécifications textuelles

L'Annexe F fournit des spécifications textuelles, dans la syntaxe définie dans l'Annexe B, pour tous les types des blocs fonctionnels et d'adaptateurs illustrés dans la présente norme. Le contenu de l'Annexe F est normatif dans la mesure définie par la description de chaque type de bloc fonctionnel ou d'adaptateur de ce genre dans la présente norme.

NOTE Les spécifications sont énumérées alphabétiquement par nom de type.

```
 ================================================================
 FUNCTION_BLOCK E_CTU (* Compteur progressif événementiel *)
 EVENT_INPUT
   CU WITH PV; (* Comptage progressif *)
   R; (* Réinitialisation *)
 END_EVENT
 EVENT_OUTPUT
   CUO WITH Q,CV; (* Compter progressivement l'événement de sortie *)
   RO WITH Q,CV; (* Réinitialiser l'événement de sortie *)
 END_EVENT
 VAR_INPUT
   PV: UINT; (* Valeur préétablie *)
 END_VAR
 VAR_OUTPUT
   Q: BOOL; (* CV>=PV *)
   CV: UINT;
 END_VAR
 EC_STATES
   START;
   CU: CU -> CUO;
   R: R -> RO;
 END_STATES
 EC_TRANSITIONS
   START TO CU:= CU [CV<65535];
   CU TO START:= 1;
   START TO R:= R;
   R TO START:= 1;
 END_TRANSITIONS
 ALGORITHM CU IN ST: (* Comptage progressif *)
 CV:= CV + 1;
 Q:= (CV >= PV);
 END_ALGORITHM
 ALGORITHM R IN ST: (* Réinitialisation *)
 CV:= 0;
 Q:= FALSE;
 END_ALGORITHM
 END_FUNCTION_BLOCK
 ================================================================
 FUNCTION_BLOCK E_CYCLE (* Génération (cyclique) périodique d'un Event *)
 EVENT_INPUT
   START WITH DT;
   STOP;
 END_EVENT
 EVENT_OUTPUT
  EO;   (* Événement périodique à la période DT, commençant à DT après GO *)
 END_EVENT
 VAR_INPUT
   DT: TIME; (* Période entre événements *)
 END_VAR
 FBS
  DLY: E_DELAY;
 END_FBS
 EVENT_CONNECTIONS
  START TO DLY.START;
```



– 228 – 61499-1 © CEI:2012

# Annexe F
(normative)

## Spécifications textuelles

附录F以附录B中定义的语法提供了本标准中显示的所有类型的功能块和适配器的文本规范。附录F的内容在本标准中对每种类型的功能块或此类适配器的描述所定义的范围内是规范性的。

注意规格按类型名称的字母顺序列出。

```
                                                 ============================
                                                 sif événementiel *)
```

铜带光伏； （*计数*）

CUOWITHQ CV;(*逐渐统计退出事件*)
RO带Q CV;(*重置退出事件*)

```
END_EVENT
VAR_INPUT
  PV: UINT; (* Valeur préétablie *)
END_VAR
VAR_OUTPUT
  Q: BOOL; (* CV>=PV *)
  CV: UINT;
END_VAR
EC_STATES
  START;
  CU: CU -> CUO;
```

开始到CU:=CU[CV<65535];
CU开始：=1;
开始R:=R;R开始：=1;

ST中的算法CU： （*向上计数*）

ST中的算法R： （*重置*）

END_FUNCTION_BLOCK==========
===FUNCTION_BLOCKE_CYCLE（*周期性（循环）生成事件*）

从DT开始；

EO;（*周期DT的周期性事件，从GO后的DT开始*）

```
END_EVENT
VAR_INPUT
  DT: TIME; (* Période entre événements *)
END_VAR
FBS
```

开始DLY.START;

```
 STOP TO DLY.STOP;
 DLY.EO TO DLY.START;
 DLY.EO TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=============================================================*)
FUNCTION_BLOCK E_D_FF (* Verrou de données (D) guidé par des événements *)
EVENT_INPUT
  CLK WITH D; (* Horloge de données *)
END_EVENT
EVENT_OUTPUT
  EO WITH Q; (* Événement de sortie lorsque la sortie Q change *)
END_EVENT
VAR_INPUT
  D: BOOL; (* Entrée de données *)
END_VAR
VAR_OUTPUT
  Q: BOOL; (* Données verrouillées *)
END_VAR
EC_STATES
  Q0; (* Q est initialement FALSE *)
  RESET: LATCH -> EO; (* Réinitialiser Q et émettre EO *)
  SET: LATCH -> EO; (* Verrouiller et émettre EO *)
END_STATES
EC_TRANSITIONS
  Q0 TO SET:= CLK [D];
  SET TO RESET:= CLK [NOT D];
  RESET TO SET:= CLK [D];
END_TRANSITIONS
ALGORITHM LATCH IN ST:
Q:=D;
END_ALGORITHM
END_FUNCTION_BLOCK
(*=============================================================*)
FUNCTION_BLOCK E_DELAY
 (* Propagation retardée d'un événement - Annulable *)
EVENT_INPUT
 START WITH DT; (* Commencer le retard *)
 STOP;      (* Annuler le retard *)
END_EVENT
EVENT_OUTPUT
 EO;     (* Événement retardé *)
END_EVENT
VAR_INPUT
 DT: TIME; (* Temps de retard *)
END_VAR
SERVICE E_DELAY/RESOURCE
SEQUENCE event_delay
 E_DELAY.START(DT) ->E_DELAY.EO();
END_SEQUENCE
SEQUENCE delay_canceled
 E_DELAY.START(DT);
 E_DELAY.STOP();
END_SEQUENCE
SEQUENCE no_multiple_delay
 E_DELAY.START(DT);
 E_DELAY.START(DT);
 ->E_DELAY.EO();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
(*=============================================================*)
FUNCTION_BLOCK E_DEMUX (* Démultiplexeur d'événements *)
EVENT_INPUT
 EI WITH K;  (* Événement à démultiplexer *)
END_EVENT
```





```
停下来DLY.STOP;
 DLY.EO TO DLY.START;
 DLY.EO TO EO;
END_CONNECTIONS



END_FUNCTION_BLOCK=============================================================
=FUNCTION_BLOCKE_D_FF（*事件驱动数据（D）锁定*）


带D的时钟；(*数据时钟*)


EO与Q；(*输出Q改变时的输出事件*)
END_EVENT
VAR_INPUT
  D: BOOL; (* Entrée de données *)
END_VAR
VAR_OUTPUT



Q0;(*Q最初为FALSE*)
复位：锁存器->EO；（*重置Q并发出EO*）
设置：锁存器->EO；(*锁定并签发EO*)



设置为复位：=CLK[NOTD];
重置为设置：=CLK[D];

ST中的算法锁存器：


END_FUNCTION_BLOCK=============================================================
========FUNCTION_BLOCKE_DELAY(*可取消事件的延迟传播*)EVENT_INPUT



从DT开始；(*开始延迟*)
停止;(*取消延迟*)
END_EVENT
EVENT_OUTPUT



DT：时间；（*延迟时间*）
END_VAR
SERVICE E_DELAY/RESOURCE
SEQUENCE event_delay
 E_DELAY.START(DT) ->E_DELAY.EO();
END_SEQUENCE
SEQUENCE delay_canceled
 E_DELAY.START(DT);
 E_DELAY.STOP();
END_SEQUENCE
SEQUENCE no_multiple_delay
 E_DELAY.START(DT);
 E_DELAY.START(DT);
 ->E_DELAY.EO();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
                                                    ======================
                                                    énements *)

EI与K；(*解复用的事件*)
END_EVENT
```

```
EVENT_OUTPUT
 EO0;
 EO1;
 EO2;
 EO3;     (* Le nombre de sorties dépend de la mise en œuvre *)
END_EVENT
VAR_INPUT
 K: UINT;  (* Indice d'événement, le maximum dépend de la mise en œuvre *)
END_VAR
EC_STATES
 START; (* État initial *)
 TRIGGERED; (* État intermédiaire après l'arrivée de EI *)
 EO0: -> EO0;
 EO1: -> EO1;
 EO2: -> EO2;
 EO3: -> EO3;
END_STATES
EC_TRANSITIONS
 START TO TRIGGERED:= EI;
 TRIGGERED TO EO0:= [K=0];
 TRIGGERED TO EO1:= [K=1];
 TRIGGERED TO EO2:= [K=2];
 TRIGGERED TO EO3:= [K=3];
 TRIGGERED TO START:= [K>3];
 EO0 TO START:= 1;
 EO1 TO START:= 1;
 EO2 TO START:= 1;
 EO3 TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_F_TRIG (* Détection booléenne de front descendant *)
EVENT_INPUT
 EI WITH QI;   (* Entrée d'événements *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Sortie d'événements *)
END_EVENT
VAR_INPUT
 QI: BOOL;  (* Entrée booléenne pour détection de front descendant *)
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;
END_FBS
EVENT_CONNECTIONS
  EI TO D.CLK;
  D.EO TO SW.EI;
  SW.EO0 TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
  QI TO D.D;
  D.Q TO SW.G;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_MERGE (* Fusion (OR) de plusieurs événements *)
EVENT_INPUT
  EI1; (* Premier événement d'entrée *)
  EI2; (* Deuxième événement d'entrée *)
  END_EVENT
EVENT_OUTPUT EO; (* Événement de sortie *)
END_EVENT
EC_STATES
  START;  (* État initial *)
  EO:    (* Émettre l'événement EO *)
    ->EO;
END_STATES
EC_TRANSITIONS
  START TO EO:= EI1;
```





```
EVENT_OUTPUT
```

EO3;(*输出的数量取决于实现*)

K：单位；（*事件索引，最大值取决于实现*）

触发；(*EI到达后的中间状态*)

```
 EO0: -> EO0;
 EO1: -> EO1;
 EO2: -> EO2;
```

开始触发：=EI;
触发到EO0:=[K=0];触发到EO1:=[K=1];
触发到EO2:=[K=2];触发到EO3:=[K=3];

触发开始：=[K>3];

```
 EO0 TO START:= 1;
 EO1 TO START:= 1;
```

END_FUNCTION_BLOCK==========
=======FUNCTION_BLOCKE_F_TRIG(*布尔检测下降沿*)

EI与气；(*事件条目*)

```
END_EVENT
EVENT_OUTPUT
```

问：布尔值；(*用于下降沿检测的布尔输入*)

```
END_VAR
FBS
 D: E_D_FF;
```

EI转D.CLK;
D.EO到SW.EI;

```
              O;
              NS
              ONS
```

QITOD.D;

END_FUNCTION_BLOCK==========
========FUNCTION_BLOCKE_MERGE（*合并（或）多个事件*）

```
EVENT_INPUT
```

EVENT_OUTPUTEO；(*退出事件*)

```
END_EVENT
EC_STATES
  START;  (* État initial *)
  EO:    (* Émettre l'événement EO *)
```

开始到EO:=EI1;

```
  START TO EO:= EI2;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_N_TABLE (* Génération d'un train fini d'événements
distincts, guidée par table *)
EVENT_INPUT
  START WITH DT, N;
  STOP;
END_EVENT
EVENT_OUTPUT
 EO0;   (* N événements à des périodes DT, en commençant à DT[0] après START
*)
 EO1;
 EO2;
 EO3;   (* Extensible *)
END_EVENT
VAR_INPUT
  DT: TIME[3]; (* Périodes entre événements *)
  N: UINT;  (* Nombre d'événements à générer (=3 dans cet exemple) *)
END_VAR
SERVICE E_N_TABLE/RESOURCE
SEQUENCE typical_operation
  E_N_TABLE.START(DT,N) -> E_N_TABLE.EO0() -> E_N_TABLE.EO1() ->
E_N_TABLE.EO2() -> E_N_TABLE.EO3();
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_PERMIT (* Propagation permissive d'un événement *)
EVENT_INPUT EI WITH PERMIT; (* Entrée d'événements *)
END_EVENT
EVENT_OUTPUT EO; (* Sortie d'événements *)
END_EVENT
VAR_INPUT PERMIT: BOOL; END_VAR
EC_STATES
  START;  (* État initial *)
  EO:    (* Émettre l'événement EO *)
    ->EO;
END_STATES
EC_TRANSITIONS
  START TO EO:= EI [PERMIT];
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
(*=================================================================*)
FUNCTION_BLOCK E_R_TRIG (* Détection booléenne de front montant *)
EVENT_INPUT
 EI WITH QI;   (* Entrée d'événements *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Sortie d'événements *)
END_EVENT
VAR_INPUT
 QI: BOOL;  (* Entrée booléenne pour détection de front montant *)
END_VAR
FBS
 D: E_D_FF;
 SW: E_SWITCH;
END_FBS
EVENT_CONNECTIONS
  EI TO D.CLK;
  D.EO TO SW.EI;
  SW.EO1 TO EO;
END_CONNECTIONS
DATA_CONNECTIONS
  QI TO D.D;
  D.Q TO SW.G;
```





开始到EO:=EI2;
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK(*=================================================================
========FUNCTION_BLOCKE_N_TABLE(*生成有限序列的不同事件，由表引导*)EVENT_INPUT

以DT、N开头；

EO0;(*DT周期的N个事件，从START之后的DT[0]开始*)EO1;EO2;

```
 EO3;   (* Extensible *)
```

N：单位；（*要生成的事件数（在本例中=3）*）

SERVICEE_N_TABLERESOURCESEQUEN
CE典型操作

```
  E_N_TABLE.START(DT,N) -> E_N_TABLE.EO0() -> E_N_TABLE.EO1() ->
```

END_FUNCTION_BLOCK(*=================================================================
========FUNCTION_BLOCKE_PERMIT(*允许传播事件*)

EVENT_INPUTEIWITHPERMIT；(*事件条目*)

```
END_EVENT
EVENT_OUTPUT EO; (* Sortie d'événements *)
END_EVENT
VAR_INPUT PERMIT: BOOL; END_VAR
EC_STATES
  START;  (* État initial *)
  EO:    (* Émettre l'événement EO *)
```

开始到EO:=EI[许可];
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK(*=================================================================
========FUNCTION_BLOCKE_R_TRIG（*上升沿的布尔检测*）

EI与气；(*事件条目*)

```
END_EVENT
EVENT_OUTPUT
```

问：布尔值；(*上升沿检测的布尔输入*)

```
END_VAR
FBS
 D: E_D_FF;
```

EI转D.CLK；
D.EO到SW.EI；

```
            O;
            NS
            ONS
```

QITOD.D;

```
  D.Q TO SW.G;
```

```
END_CONNECTIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_REND (* Rendez-vous de deux éléments *)
EVENT_INPUT
  EI1; (* Premier événement d'entrée *)
  EI2; (* Deuxième événement d'entrée *)
  R;  (* Réinitialiser l'événement *)
END_EVENT
EVENT_OUTPUT
 EO;  (* Rendez-vous d'événement de sortie *)
END_EVENT
EC_STATES
  START;  (* État initial *)
  EI1;   (* EI1 est arrivé, attendre EI2 ou R *)
  EO:    (* Émettre l'événement de rendez-vous*)
    ->EO;
  EI2;   (* EI2 est arrivé, attendre EI1 ou R *)
END_STATES
EC_TRANSITIONS
  START TO EI1:= EI1;
  EI1 TO START:= R;
  START TO EI2:= EI2;
  EI2 TO START:= R;
  EI1 TO EO:= EI2;
  EI2 TO EO:= EI1;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_RESTART (* Génération d'événements de redémarrage *)
EVENT_OUTPUT
 COLD;     (* Redémarrage à froid *)
 WARM;     (* Redémarrage à chaud *)
END_EVENT
SERVICE RESOURCE/E_RESTART
SEQUENCE cold_restart ->E_RESTART.COLD(); END_SEQUENCE
SEQUENCE warm_restart ->E_RESTART.WARM(); END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_RS (* bistable piloté par des événements *)
EVENT_INPUT
  S; (* Établir l'événement *)
  R; (* Réinitialiser l'événement *)
END_EVENT
EVENT_OUTPUT
  EO WITH Q; (* Produire l'événement *)
END_EVENT
VAR_OUTPUT
  Q; BOOL; (* État de sortie courant *)
END_VAR
EC_STATES
  Q0; (* Q est initialement FALSE *)
  RESET; RESET -> EO; (* Réinitialiser Q et émettre EO *)
  SET; SET -> EO; (* Positionner Q et émettre EO *)
END_STATES
EC_TRANSITIONS
  Q0 TO SET:= S;
  SET TO RESET:= R;
  RESET TO SET:= S;
END_TRANSITIONS
ALGORITHM SET IN ST: (* Établir Q*)
Q:=TRUE;
END_ALGORITHM
ALGORITHM RESET IN ST: (* Réinitialiser Q *)
Q:=FALSE;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
```



– 232 – 61499-1 © CEI:2012

```
END_FUNCTION_BLOCK================================================================
========FUNCTION_BLOCKE_REND(*渲染两个元素*)

EVENT_INPUT
  EI1; (* Premier événement d'entrée *)
  EI2; (* Deuxième événement d'entrée *)


EO;(*退出活动预约*)


EI1;(*EI1已到达，等待EI2或R*)EO:(*Emitrendezvousevent*)->EO;EI2;(*E
I2已到，等待EI1或R*)



开始到EI1：=EI1；EI1开始：=R
；开始到EI2：=EI2；


EI1到EO:=EI2;EI2到EO:=EI1;
EO开始：=1；END_TRANSI
TIONS

END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_RESTART (* Génération d'événements de redémarrage *)
EVENT_OUTPUT
 COLD;     (* Redémarrage à froid *)
 WARM;     (* Redémarrage à chaud *)
END_EVENT
SERVICE RESOURCE/E_RESTART



END_FUNCTION_BLOCK================================================================
========FUNCTION_BLOCKE_RS（*事件驱动双稳态*）

EVENT_INPUT
  S; (* Établir l'événement *)


EO与Q；(*生产活动*)


问；布尔;(*当前输出状态*)


Q0;(*Q最初为FALSE*)
重置;重置->EO；（*重置Q并发出EO*）
放;设置->EO；（*位置Q和发射EO*）



设置为复位：=R；重置为设
置：=S;

ST中的算法集：（*设置Q*）


ST中的算法重置：（*重置Q*）
Q:=FALSE;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
```

```
FUNCTION_BLOCK E_SELECT (* Sélection entre deux événements *)
EVENT_INPUT
  EI0 WITH G; (* Événement d'entrée, sélectionné lorsque G=0 *)
  EI1 WITH G; (* Événement d'entrée, sélectionné lorsque G=1 *)
END_EVENT
EVENT_OUTPUT EO; (* Événement de sortie *)
END_EVENT
VAR_INPUT G: BOOL; (* Séletctionner EI0 lorsque G=0, EI1 lorsque G=1 *)
END_VAR
EC_STATES
  START;   (* État initial *)
  EO: -> EO; (* Émettre l'événement de sortie*)
END_STATES
EC_TRANSITIONS
  START TO EO:= EI0 [NOT G];
  START TO EO:= EI1 [G];
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SPLIT (* Diviser un événement *)
EVENT_INPUT
  EI; (* Événement d'entrée *)
END_EVENT
EVENT_OUTPUT
 EO1;  (* Premier événement de sortie *)
 EO2;  (* Deuxième événement de sortie, etc. *)
END_EVENT
EC_STATES
  START; (* État initial *)
  EO: (* Extensible *)
    ->EO1, (* Produire un premier événement *)
    ->EO2; (* Produire un deuxième événement, etc. *)
END_STATES
EC_TRANSITIONS
  START TO EO:= EI;
  EO TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SWITCH (* Commuter (démultiplexer) un événement *)
EVENT_INPUT EI WITH G; (* Entrée d'événements *)
END_EVENT
EVENT_OUTPUT
  EO0; (* Sortie, commutée de EI lorsque G=0 *)
  EO1; (* Sortie, commutée de EI lorsque G=1 *)
END_EVENT
VAR_INPUT G: BOOL; (* Commuter EI à EI0 lorsque G=0, à EI1 lorsque G=1 *)
END_VAR
EC_STATES
  START; (* État initial *)
  G0:  (* Émettre EO0 lorsque EI arrive avec G=0 *)
    ->EO0;
  G1: (* Émettre EO1 lorsque EI arrive avec G=1 *)
    ->EO1;
END_STATES
EC_TRANSITIONS
  START TO G0:= EI [NOT G];
  G0 TO START:= 1;
  START TO G1:= EI [G];
  G1 TO START:= 1;
END_TRANSITIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_TABLE (* Génération d'un train fini d'événements, guidée
par table *)
EVENT_INPUT
 START WITH DT, N;
 STOP; (* Annuler*)
```





```
FUNCTION_BLOCKE_SELECT(*两个事件之间的选择*)

EI0与G；(*输入事件，当G=0时选择*)EI1WITHG;(*输入事件，当G=1时选择*)

END_EVENTEVENT_OUTPUTEO；(*退出事件*)

END_EVENT
VAR_INPUT G: BOOL; (* Séletctionner EI0 lorsque G=0, EI1 lorsque G=1 *)
END_VAR
EC_STATES
  START;   (* État initial *)
                              nement de sortie*)


开始到EO:=EI0[不是G];
开始到EO:=EI1[G];
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_SPLIT (* Diviser un événement *)
EVENT_INPUT



EO1;(*第一次退出事件*)
EO2;(*第二次退出事件等*)



EO:(*Expandable*)->EO1 (*Producefirstevent*)->EO2;(*产生第二个事件，等等
*)END_STATES



开始到EO:=EI;
EO开始：=1；END_TRANSI
TIONS
END_FUNCTION_BLOCK================================================================
========FUNCTION_BLOCKE_SWITCH（*切换（解复用）事件*）

EVENT_INPUTEI与G；(*事件条目*)


EO0:(*输出，G=0时从EI切换*)EO1;(*输出，当G=1时从EI切换*)


VAR_INPUTG：布尔；(*当G=0时将EI切换到EI0，当G=1时切换到EI1*)



G0:(*EmitEO0whenEI到达G=0*)->EO0;G1:(*EmitEO1whenEI到达G=1*)->EO
1;END_STATES



开始到G0:=EI[NOTG];
G0开始：=1；开始到G1:=EI[G];



END_FUNCTION_BLOCK================================================================
==FUNCTION_BLOCKE_TABLE(*生成有限的事件序列，由表引导*)EVENT_INPUT



以DT、N开头；
 STOP; (* Annuler*)
```

```
END_EVENT
EVENT_OUTPUT
 EO WITH CV; (* N événements à des périodes DT, en commençant à DT[0] après
START *)
END_EVENT
VAR_INPUT
  DT: TIME[4]; (* Périodes entre événements *)
  N: UINT; (* Nombre d'événements à générer *)
END_VAR
VAR_OUTPUT
  CV; UINT; (* Indice d'événement courant, 0..N-1 *)
END_VAR
FBS
  CTRL: E_TABLE_CTRL;
  DLY: E_DELAY;
END_FBS
EVENT_CONNECTIONS
  START TO CTRL.INIT;
  CTRL.CLKO TO DLY.START;
  DLY.EO TO EO;
  DLY.EO TO CTRL.CLK;
  STOP TO DLY.STOP;
END_CONNECTIONS
DATA_CONNECTIONS
  DT TO CTRL.DT;
  N TO CTRL.N;
  CTRL.DTO TO DLY.DT;
  CTRL.CV TO CV;
END_CONNECTIONS
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK E_TABLE_CTRL (* Commande pour E_TABLE *)
EVENT_INPUT
 INIT WITH DT, N;
 CLK;
END_EVENT
EVENT_OUTPUT
 CLKO WITH DTO, CV;
END_EVENT
VAR_INPUT
 DT; TIME[4]; (* La longueur de Array dépend de la mise en œuvre *)
 N; UINT;   (* Nombre réel d'échelons temporels *)
END_VAR
VAR_OUTPUT
 DTO; TIME;  (* Intervalle de retard courant *)
 CV; UINT;   (* Indice d'événement courant, 0..N-1 *)
END_VAR
EC_STATES
 START;
 INIT0: INIT;
 INIT1: -> CLKO;
 STEP: STEP -> CLKO;
END_STATES
EC_TRANSITIONS
 START TO INIT0:= INIT;
 INIT0 TO INIT1:= [N>0];
 INIT0 TO START:= [N=0]; (* Ne pas exécuter si N=0 *)
 INIT1 TO START:= 1;
 START TO STEP:= CLK [CV < MIN(3,N-1)];
 STEP TO START:= 1;
END_TRANSITIONS
ALGORITHM STEP IN ST:
  CV:= CV+1;
  DTO:= DT[CV];
END_ALGORITHM
ALGORITHM INIT IN ST:
  CV:= 0;
  DTO:= DT[0];
END_ALGORITHM
END_FUNCTION_BLOCK
```



– 234 – 61499-1 © CEI:2012

带有简历的EO；(*DT周期的N个事件，在DT[0]之后开始

DT:时间[4]；(*事件之间的周期*)N:UINT;(*要生成的事件数*)

```
END_VAR
VAR_OUTPUT
  CV; UINT; (* Indice d'événement courant, 0..N-1 *)
END_VAR
FBS
  CTRL: E_TABLE_CTRL;
```

开始CTRL.INIT;

```
                    TART;
```

停下来DLY.STOP;

```
                    ;
END_CONNECTIONS
DATA_CONNECTIONS
  DT TO CTRL.DT;
  N TO CTRL.N;
  CTRL.DTO TO DLY.DT;
  CTRL.CV TO CV;
END_CONNECTIONS
END_FUNCTION_BLOCK
                ================================================
                ABLE_CTRL (* Commande pour E_TABLE *)
```

带DT的初始化，N;

```
 CLK;
END_EVENT
EVENT_OUTPUT
```

TD;时间[4]；(*数组长度取决于实现*)

```
                                                        *)
```

DTO;时间;(*当前延迟间隔*)

```
 CV; UINT;   (* Indice d'événement courant, 0..N-1 *)
END_VAR
EC_STATES
 START;
 INIT0: INIT;
 INIT1: -> CLKO;
```

开始到INIT0:=INIT;

INIT0开始：=[N=0]；(*如果N=0则不执行*)

开始步：=CLK[CV<MIN(3 N-1)];
开始步骤：=1；

ST中的算法步骤：

ST中的算法初始化：

```
  CV:= 0;
  DTO:= DT[0];
END_ALGORITHM
END_FUNCTION_BLOCK
```

```
(*************************************************************************)
FUNCTION_BLOCK E_TRAIN (* Génération d'un train fini d'événements *)
EVENT_INPUT
 START WITH DT, N;
 STOP;
END_EVENT
EVENT_OUTPUT
 EO WITH CV; (* N événements à la période DT, commençant à DT après START *)
END_EVENT
VAR_INPUT
  DT: TIME; (* Période entre événements *)
  N: UINT; (* Nombre d'événements à générer *)
END_VAR
VAR_OUTPUT
 CV: UINT; (* Indice EO (0..N-1) *)
END_VAR
FBS
 CTR: E_CTU;
 GATE: E_SWITCH;
 DLY: E_DELAY;
END_FBS
EVENT_CONNECTIONS
 START TO CTR.R;
 STOP TO DLY.STOP;
 DLY.EO TO EO;
 DLY.EO TO CTR.CU;
 CTR.CUO TO GATE.EI;
 CTR.RO TO GATE.EI;
 GATE.EO0 TO DLY.START;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
 N TO CTR.PV;
 CTR.Q TO GATE.G;
 CTR.CV TO CV;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*************************************************************************)
FUNCTION_BLOCK FB_ADD_INT (* Addition INT *)
EVENT_INPUT
 REQ WITH QI, IN1, IN2;
END_EVENT
EVENT_OUTPUT
 CNF WITH QO, STATUS, OUT;
END_EVENT
VAR_INPUT
 QI; BOOL;   (* Qualificateur d'événement *)
 IN1; INT;   (* Cumulande*)
 IN2; INT;   (* Cumulateur*)
END_VAR
VAR_OUTPUT
 QO; BOOL;    (* Qualificateur de sortie *)
 STATUS; UINT;  (* Statut d'opération *)
 OUT; INT;   (* Somme *)
END_VAR
VAR
 RESULT: DINT;
END_VAR
EC_STATES
 START;
 REQ: REQ -> CNF;
END_STATES
EC_TRANSITIONS
 START TO REQ:= REQ;
 REQ TO START:= 1;
END_TRANSITIONS
ALGORITHM REQ IN ST:
  QO:= QI;
  IF QI THEN
```





(*************************************************************************)FUNCTION
_BLOCKE_TRAIN(*Générationd'untrainfinid'événements*)

以DT、N开头；

带有简历的EO；(*NévénementsàlapériodeDT commen antàDTaprèsSTART*)

```
END_EVENT
VAR_INPUT
  DT: TIME; (* Période entre événements *)
  N: UINT; (* Nombre d'événements à générer *)
END_VAR
VAR_OUTPUT
 CV: UINT; (* Indice EO (0..N-1) *)
END_VAR
FBS
 CTR: E_CTU;
 GATE: E_SWITCH;
```

开始点击率；
停下来DLY.STOP;DLY.EO到E
O；

```
 DLY.EO TO CTR.CU;
 CTR.CUO TO GATE.EI;
 CTR.RO TO GATE.EI;
 GATE.EO0 TO DLY.START;
END_CONNECTIONS
DATA_CONNECTIONS
 DT TO DLY.DT;
 N TO CTR.PV;
```

END_FUNCTION_BLOCK(*************************************************************
*********FUNCTION_BLOCKFB_ADD_INT(*添加INT*)

带QI、IN1、IN2的请求；

带QO、状态、输出的CNF；

```
END_EVENT
VAR_INPUT
 QI; BOOL;   (* Qualificateur d'événement *)
 IN1; INT;   (* Cumulande*)
```

质量保证；布尔;(*出击资格*)

```
 STATUS; UINT;  (* Statut d'opération *)
 OUT; INT;   (* Somme *)
END_VAR
VAR
 RESULT: DINT;
END_VAR
EC_STATES
 START;
```

开始请求：=请求；
请求开始：=1；

ST中的算法请求：

如果齐那么

```
   STATUS:= 0;
   RESULT:= INT_TO_DINT(IN1) + INT_TO_DINT(IN2);
   IF (RESULT > 32767) OR (RESULT < -32768) THEN
    QO = FALSE;
    STATUS = 3;
    IF (RESULT > 32767) THEN OUT:= 32767;
    ELSE OUT:= -32768;
    END_IF;
   ELSE OUT:= RESULT;
   END_IF;
  ELSE STATUS = 1;
  END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK INTEGRAL_REAL
EVENT_INPUT
 INIT: INIT_EVENT WITH CYCLE;
 EX WITH HOLD, XIN;
END_EVENT
EVENT_OUTPUT
 INITO: INIT_EVENT WITH XOUT;
 EXO WITH XOUT;
END_EVENT
VAR_INPUT
 HOLD; BOOL;  (* 0 = Exécuter, 1 = Bloquer*)
 XIN; REAL;  (* Intégrande *)
 CYCLE; TIME; (* Période d'échantillonnage *)
END_VAR
VAR_OUTPUT
 XOUT; REAL;  (* Sortie intégrée *)
END_VAR
VAR DT; REAL; END_VAR
EC_STATES
 START;        (* État initial EC *)
 INIT:INIT -> INITO; (* État EC avec algorithme et action EC *)
 MAIN: MAIN -> EXO;
END_STATES
EC_TRANSITIONS
 START TO INIT:= INIT; (* Une transition EC *)
 START TO MAIN:= EX;
 INIT TO START:= 1;
 MAIN TO START:= 1;
END_TRANSITIONS
ALGORITHM INIT IN ST:
 XOUT:= 0.0;
 DT:= TIME_TO_REAL(CYCLE);
END_ALGORITHM
ALGORITHM MAIN IN ST:
 IF NOT HOLD THEN
  XOUT:= XOUT + XIN * DT;
 END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
ADAPTER LD_UNLD (* LOAD/UNLOAD Adapter Interface *)
EVENT_INPUT
 UNLD;   (* UNLOAD Request *)
END_EVENT
EVENT_OUTPUT
 LD WITH WO,WKPC; (* LOAD Request *)
 CNF WITH WO,WKPC; (* UNLD Confirm *)
END_EVENT
VAR_OUTPUT
 WO: BOOL;  (* Pièce à travailler présente *)
 WKPC: COLOR; (* Couleur de la pièce à travailler *)
END_VAR
SERVICE PLUG/SOCKET
SEQUENCE normal_operation
 PLUG.LD(WO,WKPC) -> SOCKET.LD(WO,WKPC);
```



– 236 – 61499-1 © CEI:2012

结果：=INT_TO_DINT(IN1)+INT_TO_DINT(IN2)；如果（结果>32767）
或（结果<–32768）那么
QO=假；状态=3；

如果（结果>32767）则输出：=32767；

```
    ELSE OUT:= -32768;

                    T;
```

其他状态=1；

```
  END_IF;
END_ALGORITHM
END_FUNCTION_BLOCK
================================================================
                     RAL_REAL

                     TH CYCLE;
```

EXWITHHOLD XIN;

```
                  NT WITH XOUT;
```

带XOUT的EXO；

```
END_EVENT
VAR_INPUT
 HOLD; BOOL;  (* 0 = Exécuter, 1 = Bloquer*)
 XIN; REAL;  (* Intégrande *)
 CYCLE; TIME; (* Période d'échantillonnage *)
END_VAR
VAR_OUTPUT
 XOUT; REAL;  (* Sortie intégrée *)
```

开始;(* tat初始EC*)
初始化：初始化–>初始化；(* tatECavecalgorithmeetactionEC*)

开始初始化：=初始化；(*Une过渡EC*)
开始主要：=EX；
初始化开始：=1；主要开始
：=1；

ST中的算法初始化：

```
                          LE);
```

ST中的主要算法：
如果不持有则

END_FUNCTION_BLOCK==========
==========ADAPTERLD_UNLD(*LOADUNLOADAdapterInterface*)

未定；(*卸载请求*)

LD与WO，WKPC；(*加载请求*)
CNF与WO WKPC;(*UNLD确认*)

WKPC：颜色；(*Couleurdelapièceàtravailler*)

```
END_VAR
SERVICE PLUG/SOCKET
SEQUENCE normal_operation
 PLUG.LD(WO,WKPC) -> SOCKET.LD(WO,WKPC);
```

```
 SOCKET.UNLD() -> PLUG.UNLD();
 PLUG.CNF() -> SOCKET.CNF();
END_SEQUENCE
END_SERVICE
END_ADAPTER
================================================================
FUNCTION_BLOCK MANAGER (* Management Service Interface *)
EVENT_INPUT
  INIT WITH QI, PARAMS; (* Initialisation de service *)
  REQ WITH QI, CMD, OBJECT; (* Demande de service *)
END_EVENT
EVENT_OUTPUT
  INITO WITH QO, STATUS; (* Confirmation d'initialisation *)
  CNF WITH QO, STATUS, RESULT; (* Confirmation de Service *)
END_EVENT
VAR_INPUT
  QI; BOOL; (* Qualificateur d'entrée d'événements *)
  PARAMS; WSTRING; (* Paramètres de service *)
  CMD; UINT; (* Commande énumérée *)
  OBJECT; BYTE[512]; (* Objet Command *)
END_VAR
VAR_OUTPUT
  QO; BOOL; (* Qualificateur de sortie d'événements*)
  STATUS; UINT; (* Statut de service *)
  RESULT; BYTE[512]; (* Objet Result *)
END_VAR
SERVICE MANAGER/resource
SEQUENCE normal_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement() -> MANAGER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement(PARAMS) -> MANAGER.INITO-
(STATUS);
END_SEQUENCE
SEQUENCE normal_command_sequence
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.CNF+(STATUS,RESULT);
END_SEQUENCE
SEQUENCE command_error
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.IND-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
  MANAGER.INIT-() -> resource.terminateService() -> MANAGER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
  resource.serviceTerminated(STATUS) -> MANAGER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK PI_REAL
EVENT_INPUT
  INIT WITH KP, KI, CYCLE;
  EX WITH HOLD, PV, SP, KP, KI, CYCLE;
END_EVENT
EVENT_OUTPUT
  INITO WITH XOUT;
  EXO WITH XOUT;
END_EVENT
VAR_INPUT
  HOLD; BOOL; (* Retenir si TRUE *)
  PV; REAL; (* Variable de processus *)
  SP; REAL; (* Valeur de consigne *)
  KP; REAL; (* Constante de proportionnalité *)
  KI; REAL; (* Constante d'intégration,1/s *)
  CYCLE; TIME; (* Période d'échantillonnage *)
END_VAR
VAR_OUTPUT
```



61499-1 © CEI:2012 – 237 –

END_SERVICEEND_ADAPTER==========
==========FUNCTION_BLOCKMANAGER(*管理服务接口*)

用QI、PARAMS初始化；(*初始化服务*)
带QI、CMD、OBJECT的请求；(*服务需求*)

使用QO、状态初始化；(*Confirmationd'initialisation*)CNFWITHQO STATUS RESULT;(*服务确认*)

nts *)
参数；字符串；(*服务参数*)

目的;字节[512]；(*对象命令*)

质量保证；布尔;(*Qualificateurdesortied'événements*)
地位;单位;(*Statutdeservice*)结果；字节[512]；(*对象结果*)

```
END_VAR
SERVICE MANAGER/resource
SEQUENCE normal_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement() -> MANAGER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
  MANAGER.INIT+(PARAMS) -> resource.initManagement(PARAMS) -> MANAGER.INITO-
(STATUS);
END_SEQUENCE
SEQUENCE normal_command_sequence
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.CNF+(STATUS,RESULT);
END_SEQUENCE
SEQUENCE command_error
  MANAGER.REQ+(CMD,OBJECT) -> resource.performCommand(CMD,OBJECT) ->
MANAGER.IND-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
  MANAGER.INIT-() -> resource.terminateService() -> MANAGER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
  resource.serviceTerminated(STATUS) -> MANAGER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
                              ========================================
```

用KP、KI、CYCLE初始化；
EXWITHHOLD、PV、SP、KP、KI、CYCLE；

;
带XOUT的EXO；

抓住;布尔;(*RetenirsiTRUE*)
光伏；真实的;(*过程变量*)

```
  SP; REAL; (* Valeur de consigne *)
  KP; REAL; (* Constante de proportionnalité *)
  KI; REAL; (* Constante d'intégration,1/s *)
  CYCLE; TIME; (* Période d'échantillonnage *)
END_VAR
VAR_OUTPUT
```

```
  XOUT: REAL;
END_VAR
FBS
  CALC: PID_CALC;
  INTEGRAL_TERM: INTEGRAL_REAL;
END_FBS
EVENT_CONNECTIONS
  INIT TO CALC.INIT;
  EX TO CALC.PRE;
  CALC.POSTO TO EXO;
  INTEGRAL_TERM.INITO TO INITO;
  CALC.INITO TO INTEGRAL_TERM.INIT;
  CALC.PREO TO INTEGRAL_TERM.EX;
  INTEGRAL_TERM.EXO TO CALC.POST;
END_CONNECTIONS
DATA_CONNECTIONS
  HOLD TO INTEGRAL_TERM.HOLD;
  PV TO CALC.PV;
  SP TO CALC.SP;
  KP TO CALC.KP;
  KI TO CALC.KI;
  CYCLE TO INTEGRAL_TERM.CYCLE;
  CALC.XOUT TO XOUT;
  CALC.ETERM TO INTEGRAL_TERM.XIN;
  INTEGRAL_TERM.XOUT TO CALC.ITERM;
  0 TO CALC.TD;
  0 TO CALC.DTERM;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=========================================================================*)
SUBAPPLICATION PI_REAL_APPL (* une sous-application *)
EVENT_INPUT
 INIT;
 EX;
END_EVENT
EVENT_OUTPUT
 INITO;
 EXO;
END_EVENT
VAR_INPUT
 HOLD: BOOL; (* Retenir si TRUE *)
 PV; REAL; (* Variable de processus *)
 SP; REAL; (* Valeur de consigne *)
 KP; REAL; (* Gain proportionnel *)
 KI; REAL; (* Gain intégral = Période d'échantillonnage par temps de
réinitialisation *)
 X0; REAL; (* Sortie d'intégrateur initiale *)
END_VAR
VAR_OUTPUT XOUT: REAL; END_VAR
FBS
 ETERM: FB_SUB_REAL;
 INTEGRATOR: ACCUM_REAL;
 CALC: PI_CALC;
END_FBS
EVENT_CONNECTIONS
  INIT TO INTEGRATOR.INIT;
  INTEGRATOR.INITO TO INITO;
  EX TO ETERM.REQ;
  ETERM.CNF TO INTEGRATOR.EX;
  INTEGRATOR.EXO TO CALC.EX;
  CALC.EXO TO EXO;
END_CONNECTIONS
DATA_CONNECTIONS
  X0 TO INTEGRATOR.X0;
  HOLD TO INTEGRATOR.HOLD;
  PV TO ETERM.IN1;
  SP TO ETERM.IN2;
  KP TO CALC.KP;
  KI TO CALC.KI;
  ETERM.OUT TO INTEGRATOR.XIN;
```



– 238 – 61499-1 © CEI:2012

```
  XOUT: REAL;
END_VAR
FBS

                        GRAL_REAL;

EVENT_CONNECTIONSINIT到C
ALC.INIT；EXTOCALC.PRE；

  CALC.POSTO TO EXO;
  INTEGRAL_TERM.INITO TO INITO;
  CALC.INITO TO INTEGRAL_TERM.INIT;
                              X;
                              ST;

DATA_CONNECTIONS保持到INTEGRAL_TER
M.HOLD；
PV到CALC.PV;SP到CALC.
SP;KP到CALC.KP;KITOCA
LC.KI;

循环到INTEGRAL_TERM.CYCLE；
  CALC.XOUT TO XOUT;
  CALC.ETERM TO INTEGRAL_TERM.XIN;
  INTEGRAL_TERM.XOUT TO CALC.ITERM;
  0 TO CALC.TD;
  0 TO CALC.DTERM;
END_CONNECTIONS
END_FUNCTION_BLOCK
(*=========================================================================*)
SUBAPPLICATION PI_REAL_APPL (* une sous-application *)
EVENT_INPUT
 INIT;
 EX;
END_EVENT
EVENT_OUTPUT
 INITO;



保持：布尔；(*RetenirsiTRUE*)
光伏；真实的;(*过程变量*)
SP;真实的;(*Valeurdeconsigne*)KP;真实的;(*增益比
例*)
基;真实的;(*Gainintégral=Péroded'échantillonnagepartempsderéinitialisation*)X0;真实的;(*首字母
组合*)

END_VAR
VAR_OUTPUT XOUT: REAL; END_VAR
FBS
 ETERM: FB_SUB_REAL;
 INTEGRATOR: ACCUM_REAL;



初始化到积分器.INIT；
                        TO INITO;
EXTOETERM.REQ；
  ETERM.CNF TO INTEGRATOR.EX;
  INTEGRATOR.EXO TO CALC.EX;
  CALC.EXO TO EXO;



持有积分。持有；
                       ;
                       ;
KP到CALC.KP;KITOCALC.
KI;
  ETERM.OUT TO INTEGRATOR.XIN;
```

```
   ETERM.OUT TO CALC.ETERM;
   INTEGRATOR.XOUT TO CALC.ITERM;
   CALC.XOUT TO XOUT;
   1 TO ETERM.QI;
END_CONNECTIONS
END_SUBAPPLICATION
================================================================
FUNCTION_BLOCK REQUESTER
           (* Interface de demandeur de service *)
EVENT_INPUT
 INIT WITH QI, PARAMS;    (* Initialisation de service *)
 REQ WITH QI, SD_1, SD_m;  (* Demande de service *)
END_EVENT
EVENT_OUTPUT
 INITO WITH QO, STATUS;      (* Confirmation d'initialisation *)
 CNF WITH QO, STATUS, RD_1, RD_n; (* Confirmation de Service *)
END_EVENT
VAR_INPUT
 QI; BOOL;   (* Qualificateur d'entrée d'événements *)
 PARAMS; ANY; (* Paramètres de service *)
 SD_1; ANY;  (* Données à transférer, extensible *)
 SD_m; ANY;  (* Dernier élément de données à transférer *)
END_VAR
VAR_OUTPUT
 QO; BOOL;   (* Qualificateur de sortie d'événements *)
 STATUS; ANY; (* Statut de service *)
 RD_1; ANY;  (* Données reçues, extensible *)
 RD_n; ANY;  (* Dernier élément de données reçu *)
END_VAR
SERVICE REQUESTER/RESOURCE
SEQUENCE normal_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE normal_data_transfer
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF+(RD_1,...,RD_n);
END_SEQUENCE
SEQUENCE data_transfer_error
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
 REQUESTER.INIT-() -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
 -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK
================================================================
FUNCTION_BLOCK XBAR_MVCA (* XBAR_MVC + Adapters *)
EVENT_INPUT
 INIT WITH VF,VR,DTL,DT,BKGD,LEN,DIA,DIR; (* Initialiser *)
END_EVENT
EVENT_OUTPUT
 INITO;
END_EVENT
VAR_INPUT
 VF: INT;= 20;    (* Vitesse ADVANCE en +%/s *)
 VR: INT;= -40;    (* Vitesse RETRACT en -%/s *)
 DTL: TIME;= t#750ms; (* Retard LOAD *)
 DT; TIME;= t#250ms; (* Intervalle de simulation *)
 BKGD; COLOR;= COLOR#blue; (* Couleur barre de transfert *)
 LEN; UINT;= 5;  (* Longueur de barre en diamètres *)
 DIA; UINT;= 20; (* Diamètre de la pièce à travailler *)
 DIR; UINT;     (* Orientation: 0=Gauche/Droite, 1=Haut/Bas, 2=D/G, 3=B/H *)
END_VAR
SOCKETS
```



61499-1 © CEI:2012 – 239 –

```
   ETERM.OUT TO CALC.ETERM;
   INTEGRATOR.XOUT TO CALC.ITERM;
```

END_SUBAPPLICATION==========
======FUNCTION_BLOCKREQUEST(*ServiceRequesterInterface*)EVENT_INPUT

用QI、PARAMS初始化；(*服务初始化*)
请求QI、SD_1、SD_m；（*服务请求*）

使用QO、状态初始化；(*初始化确认*)
带有QO、状态、RD_1、RD_n的CNF；(*服务确认*)

参数；任何;(*服务设置*)

SD_m;任何;(*要传输的最后一条数据*)

质量保证；布尔;(*事件输出限定符*)
地位;任何;(*服务状态*)

RD_n;任何;(*收到的最后一个数据项*)

```
END_VAR
SERVICE REQUESTER/RESOURCE
SEQUENCE normal_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO+();
END_SEQUENCE
SEQUENCE unsuccessful_establishment
 REQUESTER.INIT+(PARAMS) -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE normal_data_transfer
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF+(RD_1,...,RD_n);
END_SEQUENCE
SEQUENCE data_transfer_error
 REQUESTER.REQ+(SD_1,...,SD_m) -> REQUESTER.CNF-(STATUS);
END_SEQUENCE
SEQUENCE application_initiated_termination
 REQUESTER.INIT-() -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
SEQUENCE resource_initiated_termination
 -> REQUESTER.INITO-(STATUS);
END_SEQUENCE
END_SERVICE
END_FUNCTION_BLOCK                                                    ========
```

用VF VR DTL DT BKGD LEN DIA DIR初始化；(*初始化*)

```
END_EVENT
EVENT_OUTPUT
```

VF：INT；=20；(*前进速度+%s*)
房车：INT；=-40；(*RETRACT速度in-%s*)
DTL：时间；=t#750ms；(*加载延迟*)
TD;时间;=t#250ms;(*模拟间隔*)
BKGD;颜色;=颜色#蓝色;(*转移条颜色*)
伦;单位;=5;(*棒材的直径长度*)
直径；单位;=20;(*工件直径*)

```
 DIR; UINT;     (* Orientation: 0=Gauche/Droite, 1=Haut/Bas, 2=D/G, 3=B/H *)
END_VAR
SOCKETS
```

```
 LDU_SKT: LD_UNLD;
END_SOCKETS
PLUGS
 LDU_PLG: LD_UNLD;
END_PLUGS
FBS
 MVC: XBAR_MVC;
END_FBS
EVENT_CONNECTIONS
  INIT TO MVC.INIT;
  MVC.INITO TO INITO;
  MVC.LOADED TO LDU_SKT.UNLD;
  LDU_SKT.LD TO MVC.LOAD;
  MVC.ADVANCED TO LDU_PLG.LD;
  LDU_PLG.UNLD TO MVC.UNLOAD;
  MVC.UNLOADED TO LDU_PLG.CNF;
END_CONNECTIONS
DATA_CONNECTIONS
  LDU_SKT.WO TO MVC.WI;
  LDU_SKT.WKPC TO MVC.LDCOL;
  MVC.WO TO LDU_PLG.WO;
  MVC.WKPC TO LDU_PLG.WKPC;
  VF TO MVC.VF;
  VR TO MVC.VR;
  DTL TO MVC.DTL;
  DT TO MVC.DT;
  BKGD TO MVC.BKGD;
  LEN TO MVC.LEN;
  DIA TO MVC.DIA;
  DIR TO MVC.DIR;
END_CONNECTIONS
END_FUNCTION_BLOCK
================================================================
```



– 240 – 61499-1 © CEI:2012

```
 LDU_SKT: LD_UNLD;
END_SOCKETS
PLUGS
 LDU_PLG: LD_UNLD;
END_PLUGS
FBS
```

初始化到MVC.INIT;

```
  MVC.INITO TO INITO;
  MVC.LOADED TO LDU_SKT.UNLD;
  LDU_SKT.LD TO MVC.LOAD;
  MVC.ADVANCED TO LDU_PLG.LD;
  LDU_PLG.UNLD TO MVC.UNLOAD;
  MVC.UNLOADED TO LDU_PLG.CNF;
END_CONNECTIONS
DATA_CONNECTIONS
  LDU_SKT.WO TO MVC.WI;
  LDU_SKT.WKPC TO MVC.LDCOL;
  MVC.WO TO LDU_PLG.WO;
  MVC.WKPC TO LDU_PLG.WKPC;
  VF TO MVC.VF;
  VR TO MVC.VR;
```

;

LEN到MVC.LEN;DIA转MVC.DIA；目录到MVC.DIR;

```
END_CONNECTIONS
END_FUNCTION_BLOCK
================================================================
```

# Annexe G
(informative)

# Attributs

## G.1 Principes généraux

Des *attributs* peuvent être associés aux *types de données, variables, applications* et aux *types* et *instances* de *blocs fonctionnels, équipements, ressources* et leurs éléments constitutifs. Les attributs ont des valeurs qui peuvent être modifiées et accessibles en divers points du cycle de vie du type ou de l'instance de bloc fonctionnel.

Outre les descriptions des *algorithmes* de bloc fonctionnel, des informations complémentaires sont nécessaires pour prendre en charge un bloc fonctionnel au cours de la durée de vie de son logiciel. Ces informations peuvent être données en joignant des *attributs* aux éléments constitutifs des *types* ou *instances* de bloc fonctionnel.

Des attributs peuvent être appliqués à des éléments tels que les *types de données, les variables* et les *paramètres* qui sont utilisés pour la spécification des types ou instances de bloc fonctionnel. Les éléments de langage graphiques peuvent nécessiter des attributs complémentaires pour contenir des informations telles que la position, la couleur, la taille, etc.

Les attributs peuvent aussi être appliqués directement à des types et instances de bloc fonctionnel, par exemple pour contenir la version d'une spécification du type de bloc fonctionnel.

Certains attributs peuvent être utilisés pendant tout le cycle de vie d'un bloc fonctionnel. Par exemple, un attribut lié à une spécification du type de bloc fonctionnel peut être accessible lorsque le type de bloc fonctionnel est sélectionné dans une bibliothèque, lorsqu'une instance du type de bloc fonctionnel est interrogée, etc.

D'autres attributs peuvent n'exister qu'en certains points du cycle de vie. Par exemple, le texte définissant le but d'une instance de bloc fonctionnel particulière ne pourrait être appliqué que lorsque le bloc fonctionnel est instancié, et pourrait être modifié au cours de la vie de l'instance de bloc fonctionnel.

Certains attributs de bloc fonctionnel peuvent être installés dans des *ressources* associées et être accessibles au cours de la durée de vie de l'*application* distribuée. De tels attributs sont typiquement utilisés pour prendre en charge l'accès à des valeurs de paramètres de bloc fonctionnel par des équipements externes, par exemple, pour confiner à des limites de sécurité prédéfinies les valeurs des paramètres qui peuvent être fixées à l'aide d'un configurateur manuel.

## G.2 Définitions des attributs

Une définition d'attribut donne les informations spécifiées dans le Tableau G.1. Chaque attribut a un nom et un type de données de sa valeur associée. Un attribut peut avoir une valeur par défaut qui sera utilisée tant qu'une valeur n'est pas donnée à un certain stade du cycle de vie du logiciel. Dans l'exemple donné en G.1, l'attribut `DESCRIPTION` a une valeur initiale de '' (c'est-à-dire: la chaîne vide) qui peut être écrasée en écriture par une description plus explicite lorsqu'une instance de bloc fonctionnel sera configurée ou même au cours de son utilisation active.

Les attributs eux-mêmes peuvent nécessiter des informations complémentaires à celles montrées dans le Tableau G.1. De telles informations sont appelées des *sous-attributs*.



61499-1 © CEI:2012 – 241 –

Annexe G
(informative)

Attributs

属性可以与数据类型、变量、应用程序以及功能块、设备、资源及其组成元素的类型和实例相关联。属性具有可以在功能块类型或实例的生命周期中的各个点更改和访问的值。

除了功能块算法的描述外，还需要其他信息来支持功能块在其软件的生命周期内。该信息可以通过将属性附加到功能块类型或实例的组成元素来给出。

属性可以应用于诸如数据类型、变量和用于指定功能块类型或实例的参数之类的东西。图形语言元素可能需要附加属性来包含位置、颜色、大小等信息。

属性也可以直接应用于功能块类型和实例，例如保存功能块类型规范的版本。

某些属性可以在功能块的整个生命周期中使用。例如，当从库中选择功能块类型、查询功能块类型的实例等时，可以访问与功能块类型规范相关的属性。

其他属性可能只存在于生命周期中的某些点。例如，定义特定功能块实例用途的文本可能仅在功能块被实例化时应用，并且可能在功能块实例的生命周期内改变。

一些功能块属性可以安装在相关资源中，并且可以在分布式应用程序的生命周期内被访问。此类属性通常用于支持外部设备对功能块参数值的访问，例如，将可以使用手动配置器设置的参数值限制在预定义的安全范围内。

属性定义提供表G.1中规定的信息。每个属性都有其关联值的名称和数据类型。一个属性可以有一个默认值，直到在软件生命周期的某个时间点给出一个值为止。在G.1中给出的示例中，DESCRIPTION属性的初始值为''（即：空字符串），当配置功能块实例时，甚至在配置功能块实例时，该属性可以被更明确的描述以书面形式覆盖。主动使用。

属性本身可能需要表G.1所示的附加信息。此类信息称为子属性。

**Tableau G.1 – Eléments de définitions d'attributs**

| Elément | Exemple | |
|---|---|---|
| **Nom** | DESCRIPTION | |
| **Type de données** | WSTRING(30) | |
| **Valeur par défaut** | "" | |
| **Élément associé** | **Types de bloc fonctionnel** | **Instances de bloc fonctionnel** |
| **Usage** | Configuration | Run-time (temps d'exécution) |

## G.3 Exemples

NOTE Les exemples ci-après sont donnés dans le but d'illustrer l'utilisation des attributs et ne doivent pas être considérés comme des définitions normatives d'attributs normalisés.

Un exemple d'*attribut de type de données* est:

- `Max_System_Value` – Cet attribut définit la valeur maximale prise en charge d'un type de données numérique. Il est appliqué au type de données générique `ANY_NUM` et, de ce fait, tous les types numériques tels que `INT` et `REAL` hériteront de cet attribut. Noter que chaque type de données spécifique aura sa propre valeur pour cet attribut et que les valeurs normalisées pour cet attribut pour un certain nombre de types de données sont consignées dans le Tableau E.1.

Les exemples d'attributs qui s'appliquent à des *variables* sont:

- `Diagnostic_Access` – Celui-ci détermine si, oui ou non, la valeur d'une variable est accessible par un système de diagnostic en cours d'exécution.
- `Write_Access` – Celui-ci définit le niveau d'accès requis pour modifier la valeur d'une variable, par exemple, `'Operator', 'System', 'Diagnostics'`.
- `Units` – Les unités dimensionnelles qui s'appliquent à une variable, par exemple, `'l', 'm/s', 'cm'`.
- `Usage` – Une description textuelle en plusieurs lignes de l'usage de la variable associée.

Des exemples d'*attributs de type de bloc fonctionnel* sont:

- `Usage_Class` – Celui-ci décrit l'usage général du bloc fonctionnel, par exemple, `'Input', 'Output', 'Control'`.
- `Version` – Celui-ci décrit le numéro de version de la définition de type de bloc fonctionnel, par exemple, `'1.2'`.
- `Help` – Une description textuelle en plusieurs lignes qui peut être accessible en divers points du cycle de vie.

Les attributs qui sont pertinents pour la programmation d'*algorithmes* en vue de l'*exécution* comprennent:

- `ExecutionTime` – Cet attribut, de type `TIME`, spécifie le temps du cas le plus défavorable pour l'exécution d'un *algorithme* particulier d'un *type de bloc fonctionnel* spécifié dans un *type de ressource* particulier.
- `Priority` – Cet attribut est associé à une *connexion d'événements* particulière au sein d'une *ressource* et peut être hérité du *type de ressource*. Cet attribut peut être utilisé par une ressource qui prend en charge le *multitâche* préemptif pour déterminer la priorité





**表G.1 属性定义的元素**

| Elément | Exemple | |
|---|---|---|
| | DESCRIPTION | |
| **数据类型** | WSTRING(30) | |
| **Valeur par défaut** | "" | |
| **Élément associé** | **功能块类型** | **功能块实例** |
| **Usage** | Configuration | Run-time (temps d'exécution) |

注：以下示例是为了说明属性的使用而给出的，不应被视为标准属性的规范性定义。

数据类型属性的一个示例是：

- Max_System_Value 此属性定义数字数据类型的最大支持值。它应用于ANY_NUM通用数据类型，因此所有数字类型（如INT和REAL）都将继承此属性。请注意，每个特定的数据类型都有自己的该属性值，表E.1中列出了许多数据类型的该属性的标准化值。

适用于变量的属性示例如下：

- Diagnostic_Access 这决定了一个变量的值是否可以被正在运行的诊断系统访问。
- Write_Access 这定义了修改变量值所需的访问级别，例如“操作员”、“系统”、“诊断”。
- 单位 适用于变量的维度单位，例如，'l'、'ms'、'cm'。

  用法-关联变量用法的多行文本描述。

功能块类型属性的示例如下：

- Usage_Class 这描述了功能块的一般用法，例如，“输入”、“输出”、“控制”。
- 版本 这描述了功能块类型定义的版本号，例如，“1.2”。
- 帮助-可以在生命周期的不同点访问的多行文本描述。

与执行的编程算法相关的属性包括：

- ExecutionTime TIME类型的此属性指定在特定资源类型中执行指定功能块类型的特定算法的最坏情况时间。
- 优先级——此属性与资源中的特定事件连接相关联，并且可以从资源类型继承。支持抢占式多任务的资源可以使用此属性来确定优先级

d'*exécution* d'un *algorithme* invoqué par une *action EC* associée à un *état EC* qui est activée par un événement avec la priorité spécifiée.

## G.4 Sources d'attribut

Les attributs peuvent venir des principales sources suivantes:

- Les attributs **implicites** tels que les *noms de types* de bloc fonctionnel, *noms d'instances*, *noms de variables* et leurs *types de données*, sont définis comme partie intégrante du processus normal de *déclaration* pour le bloc fonctionnel.
- Les attributs **normalisés** sont ceux qui sont requis comme partie intégrante d'une norme, tels que les versions de types de bloc fonctionnel, la plage maximale de paramètres, les descriptions de paramètres, etc.
- Les attributs **spécifiques à un produit** sont ceux qu'un vendeur de système a fournis, tels que les codes des produits des types de bloc fonctionnel, les adresses matérielles d'instances de bloc fonctionnel, etc.
- Les attributs **spécifiques à une application** sont ceux qu'un développeur de système spécifie pour prendre en charge l'utilisation d'un type de données ou d'un bloc fonctionnel particulier d'une application, tels qu'un identificateur d'instance supplémentaire de bloc fonctionnel pour adapter un style souhaité par un client, une valeur par défaut pour les paramètres de sortie, une variante de description de paramètres en langue nationale, etc.

## G.5 Héritage d'attributs

Les éléments de bloc fonctionnel hériteront des attributs provenant d'éléments plus primitifs. Par exemple, une *variable* au sein d'une *déclaration de type de bloc fonctionnel* héritera des attributs de son *type de données* associé, et une *instance* de bloc fonctionnel héritera d'attributs du *type* de bloc fonctionnel associé.

Les *types de données* hériteront d'attributs en descendant dans la hiérarchie du type générique définie dans la CEI 61131-3. Par exemple, les attributs appliqués à ANY_REAL s'appliqueront aussi à LREAL et à REAL.

## G.6 Syntaxe de déclaration

L'attribution d'une valeur d'attribut à un élément déclaré est semblable à l'attribution d'une valeur à une *instance* d'un *type d'attribut* dans laquelle l'instance a le même nom que le type.

La déclaration d'un *type* d'attribut utilise la même syntaxe que la déclaration d'un *type de données* tel que défini dans la CEI 61131-3, avec l'exception que les mots-clés délimiteurs sont ATTRIBUTE...END_ATTRIBUTE en lieu et place de TYPE...END_TYPE. Par exemple, la déclaration du type d'attribut DESCRIPTION dans le Tableau G.1 serait:

```
ATTRIBUTE DESCRIPTION: WSTRING(30); END_ATTRIBUTE
```

L'attribution d'une valeur à une *instance* d'attribut utilise la même syntaxe que celle pour l'attribution d'une valeur initiale à une *variable* telle que décrite dans la CEI 61131-3, avec les extensions suivantes:

a) le nom de l'instance d'attribut est le même que le nom du type d'attribut correspondant;

b) aucun type de données n'est spécifié pour l'instance d'attribut;

c) l'attribut de valeurs est enfermé dans la construction **pragma** définie dans la CEI 61131-3;

d) plusieurs attributions de valeurs d'attributs, séparées par des points-virgules, peuvent être incluses dans la construction pragma;





执行由与EC状态相关联的EC操作调用的算法，该状态由具有指定优先级的事件激活。

属性可以来自以下主要来源：

- 隐式属性，例如功能块类型名称、实例名称、变量名称及其数据类型，被定义为功能块正常声明过程的一部分。
- 标准属性是作为标准的一部分所必需的属性，例如功能块类型版本、最大参数范围、参数描述等。
- 产品特定属性是系统供应商提供的属性，例如功能块类型产品代码、功能块实例硬件地址等。
- 特定于应用程序的属性是系统开发人员指定以支持使用特定数据类型或应用程序构建块的属性，例如功能块的附加实例标识符以适应客户所需的样式，默认值输出参数，国家语言参数描述的变体等。

功能块元素将继承更多原始元素的属性。例如，功能块类型声明中的变量将从其关联的数据类型继承属性，而功能块实例将从关联的功能块类型继承属性。

数据类型将继承IEC61131-3中定义的通用类型层次结构的属性。例如，应用于ANY_REAL的属性也将应用于LREAL和REAL。

将属性值分配给已声明的元素类似于将值分配给属性类型的实例，其中该实例与该类型具有相同的名称。

属性类型的声明使用与IEC61131-3中定义的数据类型声明相同的语法，但分隔关键字是ATTRIBUTE…END_ATTRIBUTE而不是和而不是TYPE…END_TYPE。例如，表G.1中说明属性类型的声明将是：

将值分配给属性实例使用的语法与IEC61131-3中描述的为变量分配初始值的语法相同，但具有以下扩展：

a)属性实例名称与对应的属性类型名称相同；

b)没有为属性实例指定数据类型；

c)values属性包含在IEC61131-3中定义的pragma构造中；

d)多个分号分隔的属性值分配可以包含在pragma构造中；

e) la construction pragma doit être placée de telle manière que la déclaration à laquelle elle s'applique puisse être déterminée sans ambiguïté.

Un exemple de l'application de ces règles est:

```
FUNCTION_BLOCK PID
     {DESCRIPTION:= "Proportional + Integral + Derivative Control;
      AUTHOR:= "JHC"; VERSION:= "19990103/JHC"}
     INPUT_EVENT
      INIT WITH QI, PARAMS; {DESCRIPTION:= "Initialization Request"}
     ...etc.
```

e)pragma结构的放置方式应使其适用的语句可以明确地确定。

应用这些规则的一个例子是：

```
FUNC
     {描述：="比例+积分+微分控制；作者：="JHC"；版本：="19990103JHC"

      用QI、PARAMS初始化；{DESCRIPTION:="初始化请求" ...等。
```

## Bibliographie

CEI 60050-351:2006, *Vocabulaire Electrotechnique International – Partie 351: Technologie de commande et de régulation*

CEI 61131-5:2000, *Automates programmables – Partie 5: Communications*

CEI 61499 (toutes les parties), *Bloc fonctionnels*

CEI 61499-2, *Bloc fonctionnels – Partie 2: Exigences pour les outils logiciels*

IEC 61499-4:2012, *Blocs fonctionnels – Partie 4: Règles pour les profils de conformité*

ISO/CEI 7498-4, *Systèmes de traitement de l'information – Interconnexion de systèmes ouverts – Modèle de référence de base – Partie 4: Cadre général de gestion*

ISO/CEI 8824-1:2008, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base*

ISO/CEI 8825-1:2008, *Technologies de l'information – Règles de codage ASN.1: Spécification des règles de codage de base (BER), des règles de codage canoniques (CER) et des règles de codage distinctives (DER)*

ISO/CEI 10040:1998, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Aperçu général de la gestion-système*

ISO/CEI/IEEE 60559, *Information technology – Microprocessor systems – Floating-point arithmetic* (disponible en anglais seulement)

ISO 2382 (toutes les parties), *Technologies de l'information – Vocabulaire*



61499-1 © CEI:2012 – 245 –

IEC60050-351:2006，国际电工词汇-第351部分：控制和调节技术

IEC61131-5:2000，可编程逻辑控制器 第5部分：通信

IEC61499（所有部分），功能块

IEC61499-2，功能块 第2部分：软件工具的要求

IEC61499-4:2012，构建模块-第4部分：一致性配置文件规则

ISOIEC7498-4，信息处理系统-开放系统互连-基本参考模型-第4部分：通用管理框架

ISOIEC8824-1:2008，信息技术 抽象语法符号一(ASN.1)：基本符号规范

基本编码规则(BER)、规范编码规则(CER)和区别编码规则(DER)规范

ISOIEC10040:1998，信息技术 开放系统互连(OSI) 系统管理概述

ISOIECIEEE60559，信息技术 微处理器系统 浮点运算

ISO2382（所有部分），信息技术 词汇

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

3, rue de Varembé
PO Box 131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch



INTERNATIONAL
NICAL

3 ruedeVarembé

邮政信箱131
CH-1211 Geneva 20
Switzerland

Tel: + 41 22 919 02 11
Fax: + 41 22 919 03 00
info@iec.ch
www.iec.ch