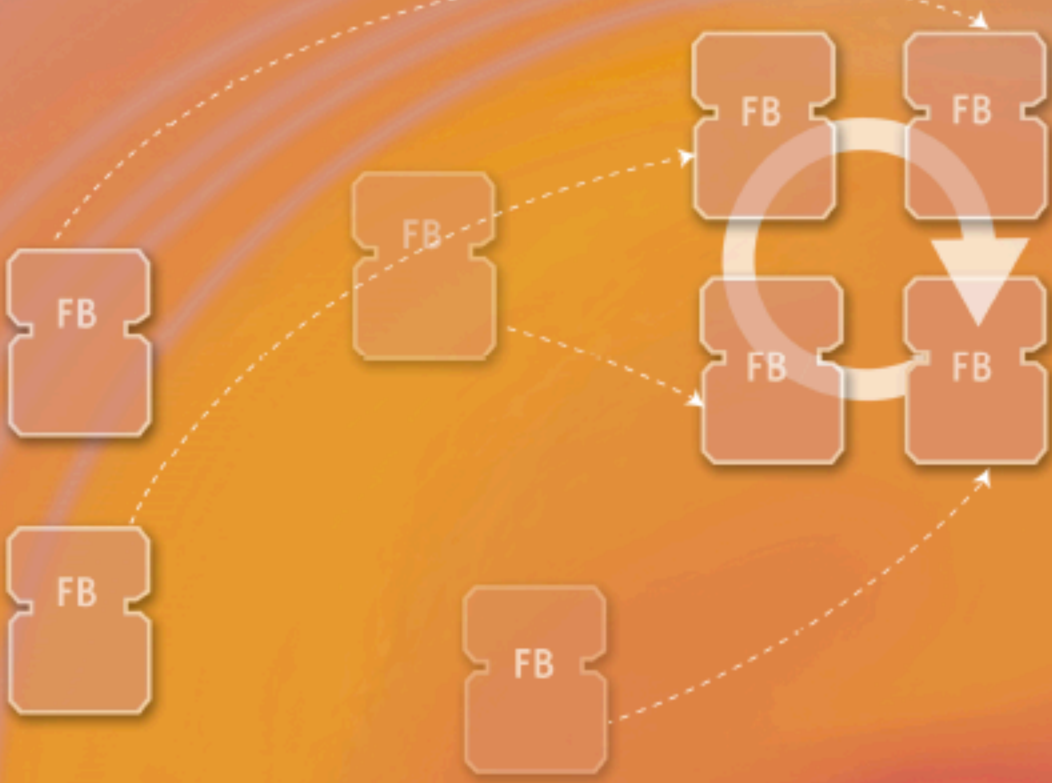
Click to ORDER from ISA webstore

Companion web-site www.fb61499.com

IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design

Third Edition

Selected pages from each chapter



Valeriy Vyatkin

IEC 61499 Function Blocks für den Entwurf von Eingebetteten und Verteilten Systemen, 3. Ausgabe Valeriy Vyatkin

ISA

O3NEDA

Click to ORDER from ISA webstore

Companion web-site www.fb61499.com

IEC 61499 FUNCTION BLOCKS FOR EMBEDDED AND DISTRIBUTED CONTROL SYSTEMS DESIGN

Third Edition

IEC 61499 FUNCTION BLOCKS FOR EMBEDDED AND DISTRIBUTED CONTROL SYSTEMS DESIGN

Third Edition

Valeriy Vyatkin

Luleå Tekniska Universitet, Sweden and

Aalto University, Finland

ISA™

O³NEIDA

Copyright 2015 by ISA—International Society of Automation

67 Alexander Drive

P.O. Box 12277

Research Triangle Park, NC 27709

All rights reserved.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2

ISBN: 978-1-941546-72-7

No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior written permission of the publisher.

Notice

The information presented in this publication is for the general education of the reader. Because neither the author nor the publisher has any control over the use of the information by the reader, both the author and the publisher disclaim any and all liability of any kind arising out of such use. The reader is expected to exercise sound professional judgment in using any of the information presented in a particular application. Additionally, neither the author nor the publisher have investigated or considered the affect of any patents on the ability of the reader to use any of the information in a particular application. The reader is responsible for reviewing any possible patents that may effect any particular use of the information presented.

Any references to commercial products in the work are cited as examples only. Neither the author nor the publisher endorses any referenced commercial product. Any trademarks or tradenames referenced belong to the respective owner of the mark or name. Neither the author nor the publisher makes any representation regarding the availability of any referenced commercial product at any time. The manufacturer's instructions on use of any commercial product must be followed at all times, even if in conflict with the information in this publication.

Library of Congress Cataloging-in-Publication Data in Process

Contents

Acknowledgements

This book was conceived within O^{3}neida, an international network of networks focused on issues related to distributed industrial automation.

The contributions of James Christensen, Hans-Michael Hanisch, Robert Brennan, and José Lastra have been particularly valuable.

Dr. James Christensen started the IEC 61499 education mission as the pioneer of object-oriented engineering in industrial automation and the leader of the Standard's Working Group in IEC. He is the author of the Function Block Development Kit (FBDK) software tool and has created a set of examples accompa nying FBDK.

The concept and the structure of this book were developed together with Prof. Hans-Michael Hanisch while preparing jointly an article on IEC 61499 in 2003-2004 (Hanisch, Vyatkin 2004).

Prof. Robert Brennan has contributed to many areas in the text. In particular, he wrote the section "Industrial Trends" discussed in Chapter 3.

Prof. José Lastra generated several ideas on the knowledge economy in industrial automation discussed in Chapter 17.

The author especially thanks Allan Martel, O^{3}neida's Chief Operating Officer, for his editing and coordinating assistance.

Antonio Valentini, O^{3}neida's CEO, has also provided valuable insights from the perspective of an applications developer along with motivation, dedication and enthusiastic support.

As part of their research work, Cheng Pang and Jeffrey Yan, Ph.D. students of the University of Auckland have developed material included in Chapters 14 and 18.

Having expressed much gratitude to all people involved, the author assumes all responsibility for any errors in the book and will be very grateful to the reader for reporting them to valeriy.vyatkin@gmail.com.

Last but not least, I wish to express my enormous gratitude to my wife Marina and our daughter Maria for their patience and continuous support during the process of completing this book.

Introduction

This book is a practical guide to component-based development of distributed, embedded, and control systems proposed within the new IEC 61499 international standard. IEC 61499 defines a component-based modelling approach using *function blocks*.

At first glance, this model appears to be quite different from the component models of other software engineering approaches such as CORBA, DCOM, or .NET (Sünder et al., 2006). In fact, however, a function block possesses the required characteristics of a software component defined as "a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties." (C. Szyperski, 2006). This statement precisely describes a function block.

In other words, the expression *function block* is broader than *software block* insofar as a *function block* captures functionality that can be implemented as software or hardware or a combination of both. This robust character of the *function block* makes it appropriate for use in the broader embedded systems domain.

The IEC 61499 Standard for Function Blocks for Industrial Process Measurement and Control Systems (IPMCS) is the result of more than 10 years of work within Working Group 6 (TC 65) of the International Electrotechnical Commission (IEC). Working Group 6 comprises leading experts in software architectures of industrial automation systems including among its members the biggest global vendors of automation products and solutions.

The standard was conceived because an increasing portion of the intellectual property (IP) related to the domain of industrial automation is now being expressed in the form of software. Efficient management of this IP becomes the key factor for answering the new challenges of markets that require flexibly reconfigurable production tailored to order. The efficient management of this IP requires easy integration of components, as well as their re-configuration and re-use.

The automation software spans diverse functional domains, such as control, diagnostics, process rendering, modelling, human-machine interface, communication, and so on. Thus, component architectures destined for industrial automation applications need to fit not only the needs of control development but also to meet the diverse requirements of all the other domains referenced above.

The IEC 61499 standard is generally known for devising function blocks with a strange shape, having both a "head" and "body" as illustrated on the cover of this book. Those with a background in networks sometimes confuse the standard with a higher-level network protocol, since the standard refers to the domain of distributed systems. Control engineers sometimes understand it as a new programming language similar to those defined in IEC 61131-3. Some IT specialists take it as a concept for modelling distributed systems that can be implemented by means of existing software and hardware. These preconceptions reflect the "multi-faceted" nature of the standard.

The IEC 61499 standard stimulates the development of new engineering technologies intended to economize design and enable faster and easier reconfiguration. The standard's approach in dealing with architecture is unusual when compared to other standards and their use in the domain of control and automation.

IEC 61499 literally defines a "reference architecture" for distributed process measurement and control systems software (DPMCS). The standard incorporates advanced software technologies, such as the encapsulation of functionality, component-based design, event-driven execution, and distribution. As a result, specific implementations available from different providers of field devices, controller hardware, human-machine interfaces, communication networks, and so on, can be integrated into component-based, heterogeneous systems.

IEC 61499 consists of four parts:

- Part 61499-1 "Architecture" (IEC 61499-1, 2005) was voted and approved as an IEC Standard in July 2004 and published in 2005. Second Edition was published in 2012.
- Part 61499-2 "Software Tools Requirements" (IEC 61499, 2005) was also voted and approved as an IEC standard in July 2004 and published in 2005.
- Part 61499-3 "Application guidelines" was approved as a technical report (IEC 61499-3, 2004).
- A voting draft of 61499-4 "Rules for compliance profiles" (IEC 61499-4, 2005) was approved in 2005.

The second edition of the IEC 61499 standard has been published in 2011. This book includes major additions and corrections that were included in the standard (see Appendix D).

Introduction to the Second Edition

In the three years since the first edition, the position of IEC 61499 has considerably strengthened. Industry is keenly attuned to the potential of function block technology.

The first edition was introduced into the curricula of several Universities around the world (the author is aware of six). Hundreds of young engineers have learned the concept of function block programming, dozens of undergraduate and postgraduate projects have been completed. Based on this experience, many corrections have been suggested and implemented.

The following add-ons are worth mentioning:

- Chapter 4 outlines the benefits of adopting the IEC 61499 for programmable logic controller (PLC) users and for embedded systems users and developers.
- The discussion of tools that support function blocks has been extended into a separate Chapter 19.
- The tutorial material has been revised to conform to the latest version of Function

Block Development Kit (FBDK) software published in 2008.

- In Chapter 14, a new visualization library is described, which extends the capabilities of standard FBDK visualization.
- New chapters have been added:
    - Chapter 18 discusses problems and solutions related to function block semantics;
    - Chapter 19 presents an extended overview of available tools;
- The bibliography has been expanded to include new works that have appeared in the last 3 years;
- Appendix C has been added to present corrections to the IEC 61499 standard, which have been accepted in 2011.

Introduction to the Third Edition

In the third edition of this book, the tutorial material has been re-implemented with the commercial development tool nxtStudio (product of the company nxtControl GmbH (Austria)). This tool supports a wide range of hardware platforms and field buses and shows that IEC 61499 is a serious technology in the industrial sector. The transition from FBDK to nxtStudio has led to a certain change in the material selection. Chapter 12 presents now nxtStudio-specific ways to adapt the hardware peripherals. Chapter 14 is entirely new material on the implementation of human-machine interfaces and Chapter 16 offers a newly developed case study and a tutorial which deals with the development of a pick-and-place robot.

How to use this book

This book is intended for a systematic learning process within the context of a University course, industrial training, or self-study.

The learning approach has been tested during industry training sessions and during undergraduate course delivery at the University of Auckland and at the Martin Luther University of Halle – Wittenberg in Germany.

The approach favours giving students hands-on experience by exploring numerous examples using the software tool NxtStudio. The book does not promote this software package over others that are also compliant with the 61499 standard. Rather, experience demonstrates that higher efficiency learning occurs when students can check and modify examples using the NxtStudio software tool.

Three tutorial guides in each of Chapters 7, 8, and 11 give the reader a step-by-step introduction to the world of function blocks and systems. The book is designed such that each chapter is an independent study unit.

Chapter 1 provides a quick introduction to the concept of function blocks using the example of a simple FLASHER.

Chapter 2 discusses how this system could have been implemented with different generations of automation technologies—from the hardwired relay circuits of the past to the modern distributed embedded systems. This chapter presents the idea of function blocks from the historical perspective of industrial automation.

Chapter 3 continues this discussion, taking into account the challenges of modern manufacturing systems that set the requirements for automation systems development.

Chapter 4 presents new technical opportunities provided by the adoption of IEC 61499 in the context of emerging intelligent automation systems.

Chapter 5 introduces the fundamentals of the new function block concept. Most of the subsequent chapters depart from the examples in Chapter 5 and rely on using the software toolset NxtStudio. (Chapters 4 and 5 are included for specialists interested in a broader reference framework for industrial automation issues. These can be reviewed in a cursory manner or even skipped in more technically oriented courses).

Chapter 6 provides a crash course in the NxtStudio software toolset. The toolset is well documented, and the information given in Chapter 6 will help the reader understand its internal structure and access the documentation provided. This book is not meant as a substitute for the documentation, but as a complementary study guide.

Chapter 7 introduces the concept of a *basic* function block.

Chapter 8 introduces the concept of a *composite* function block.

Chapter 9 shows how to build applications from function blocks. (In conjunction with Chapters 7 and 8, Chapter 9 also provides enough material to learn the abstract definition of desired system behaviour with function blocks).

Chapter 10 discusses another cornerstone concept of distributed automation systems—a model of the device. After learning this material, the reader will be able to understand an integral picture of an automation system.

Chapter 11 introduces a *distributed system configuration* model. The FLASHER system is revisited here and is used to illustrate the distributed automation principles. At this stage the reader may want to learn more about lower level system issues, such as interfacing embedded hardware or visualization.

Chapter 12 introduces the *service interface* function blocks formally (although they have been used already in the examples in previous chapters).

Chapter 13 provides an example of a complete distributed control design for a simple mechatronic system.

Chapter 14 introduces the development of human-machine interfaces using function blocks.

Chapter 15 introduces the development methodologies of automation systems. The object-oriented Model/View/Control (MVC) pattern is systematically introduced. The theory is accompanied by an example of a system that consists of a drill with feeding conveyors.

Chapter 16 discusses an extension of the MVC pattern into automation object design and illustrates this with the same example of a mechatronic system presented in Chapter 13.

Chapter 17 shows how the adoption of IEC 61499 can change the business models of companies involved in the industrial automation business.

Chapter 18 contains an overview of available tools compliant with IEC 61499. In particular, it discusses in detail ISaGRAF integrated development environment, and lightly reviews 4DIAC-IDE and FBDK.

Appendix A lists the details of the standard event processing function blocks.

Appendix B provides some excerpts from the compliance profile of the standard.

Appendix C presents a list of expected add-ons and corrections in the second edition of IEC 61499.

Book Web Site

For more information on this book, go to *www.fb61499.com*. This web site contains more teaching materials, text corrections of any reported errors, and a forum to clarify any further questions.

1

Quick start

This chapter is a primer in the basic concepts of system development using IEC 61499 function blocks. An example of a distributed automation system (FLASHER) is used to highlight the key issues in the new standard. No background in industrial automation systems and/or software development is assumed.

System modelled with function blocks

This chapter illustrates some concepts of the IEC 61499 function block architecture using the example of a system that has been specified, modelled, and implemented within the IEC 61499 architecture. This example will be re-visited in several subsequent chapters. Whereas the system might look primitive for some readers, its simplicity clearly illustrates several important features and includes sufficient detail for the reader to understand.

The system consists of four lamps blinking according to a required mode of operation as shown in Figure 1-1. Operation modes vary and include commands such as “blink all lamps together”, or “chase the light left or rightwards”, and so on. The lamps are “controlled” by a “black box” controller with four output control signals; each of which is connected to a lamp and serves as a switch, turning the lamp on and off.

Now we will consider a function block model that simulates this system working under automatic control. Later we will see that with minor changes the same model program can be used to control the physical device.

Step 1 Open a solution file FLASHER.sln in the NxtStudio as shown in Figure 1-2[1]. For that, choose menu items File-Open-Project/Solution. Next, click on the APP sign under the System sign in the navigation tree area (upper left side of the window) and you will see a screen similar to the one illustrated in Figure 1-2. This editor’s screen is divided into

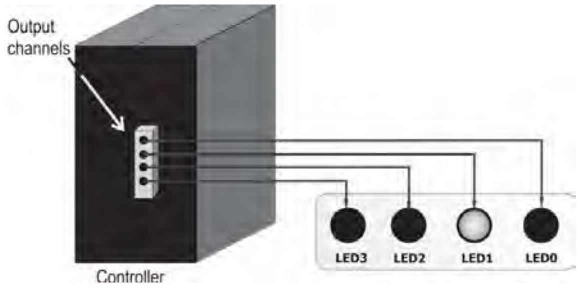


Figure 1-1 FLASHER controlled by a “black box” controller.

three areas:

- A navigation tree on the left shows the system's structure.
- The worksheet on the right (the largest one) shows the "content" of system elements. Thus, in Figure 1-2 it shows the content of the resource RES1 in device FLASHER.
- The *service area* in the bottom contains various tabs supplying service information, such as error messages, warnings, etc.

Step 2 To start the system one needs first to launch the runtime environment by starting the "nxtRT61499F-1"Windows application (with the following icon) that is supplied with NxtStudio. After that, one needs to deploy the system to the runtime, clicking on using the "Devices menu" and on right click selecting "Deploy" item;

- Once the application is deployed, one needs to start visualisation screen. This is done by right clicking on the canvas "1280 X 980" item in the navigation tree and choosing the item "Test HMI Runtime on the local computer"
- After that you must see the new human-machine interface window appeared as in Figure 1-3.

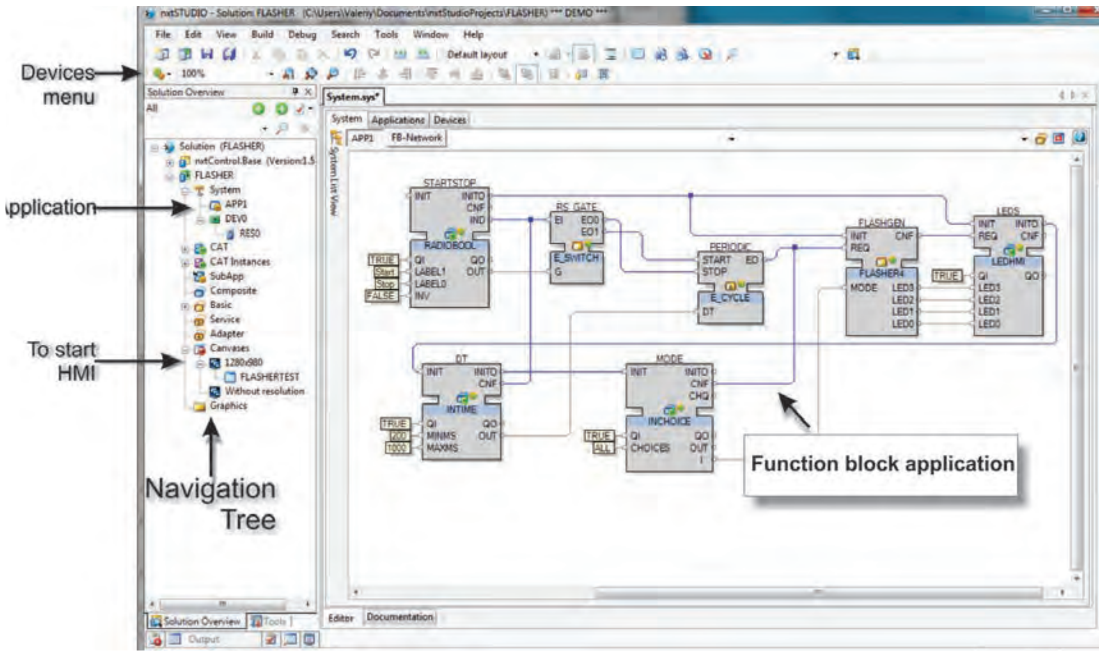


Figure 1-2 FLASHER_TEST system configuration opened in the function block editor.

Now one can "play" with this model by starting or stopping the lights with Start and Stop radio

2

Evolution of industrial automation technologies

In this chapter, the FLASHER example is used to illustrate the main features of several generations of industrial automation technologies, from hardwired relays to distributed embedded systems using field area networks.

Industrial automation technology has evolved through a number of generations. The transition from one generation to another has been marked by the following:

- The inability of previous technologies to meet new requirements emerging from new technological and economic realities;
- The availability of new technologies to meet the new requirements.

To understand better the trends discussed above, one must look back several decades. The automated FLASHER shown in Chapter 1 is a representative example of an automated product. Now we will try to follow the "imaginary evolution" of this product through the four generations of automation technologies.

First generation

In the first generation of the FLASHER (circa 1950s), the controller box would typically be realized as an electric circuit built using hardwired relay elements as shown in Figure 2-1. An additional internal relay element OFLO is used to implement a transition between states when the LED0 is ON and the state when all lamps are off. By simply switching the relays slowly, one could have implemented the desired "running lights" even without extra delay elements.

Let us assume that the vendor had three versions of this product with three different operating modes as follows:

- all lamps are flashing simultaneously,
- the lamps are flashing in the chasing-up mode, and
- the lamps are flashing in the chasing-down mode (as illustrated in Figure 2-1).

For each of these operating modes a separate relay circuit would have been required.

Second generation

In the next product generation, hardwired relays or solid-state logic circuits are replaced by general- purpose computing devices, such as microcontrollers, or special-purpose devices, such as programmable logic controllers (PLCs). Control logic is now implemented in software rather than in fixed hardware (as shown in Figure 2-2).

PLCs have a flexible architecture with a variable number of input/output (I/O) modules. These can be added as needed to the central processing unit (CPU) in order to connect the CPU with sources of input data and destinations of output data (in our case the lamps). Thus, the structure can be customized according to the requirements of a particular application in a cost-effective way, since I/O modules are selected from a range of available standard modules. For example, the PLC in Figure 2-2 has only one output module with eight logic outputs, four of which are used.

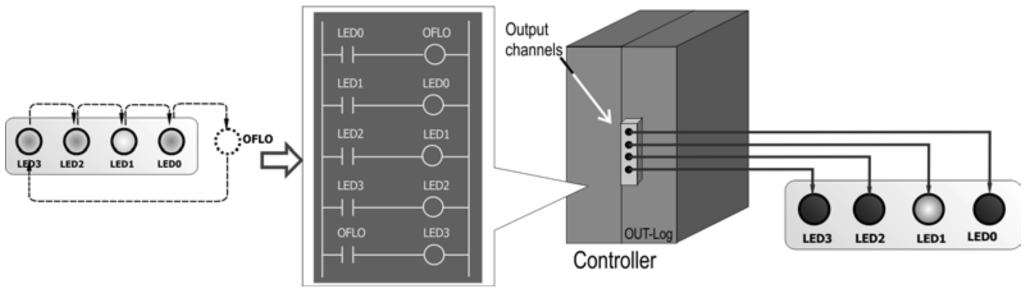
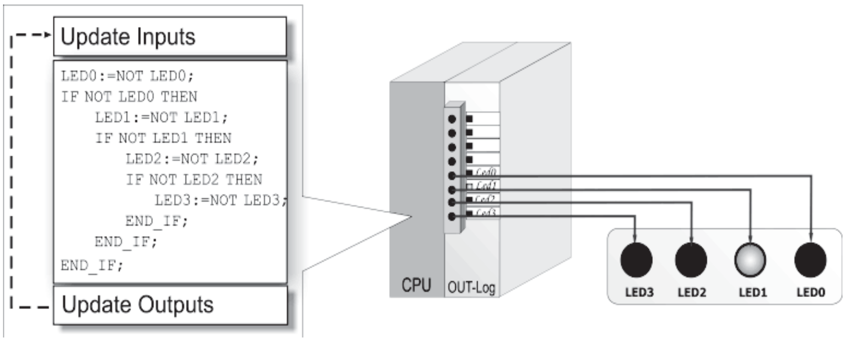


Figure 2-1 Relay implementation of the FLASHER's logic "chase down".



```
LED0:=NOT LED0;
IF NOT LED0 THEN
    LED1:=NOT LED1;
    IF NOT LED1 THEN
        LED2:=NOT LED2;
        IF NOT LED2 THEN
            LED3:=NOT LED3;
        END_IF;
    END_IF;
END_IF;
```

Figure 2-2 A programmable FLASHER—the COUNT-UP algorithm programmed in Structured Text.

The CPU runs a program that implements the desired logic behaviour of the system. Thus, creating a device with new functionality does not require design of a new circuit board. Different behaviour could be achieved by programming, with the hardware part of the system remaining unchanged, meaning cheaper production and easier maintenance, etc. In our example, the use

Summary

The frontier of automation technology is characterized by the ability of automated machines to adapt to the ever-changing conditions of their environments. This chapter showed the progress of industrial automation—from primitive ladder logic to flexible distributed systems. Industrial automation technology has evolved through several generations. This chapter identified five of them. Each new generation saw increases in the flexibility of automation systems, along with higher efficiency. The legacy of past generations is carried into each new generation; for example, many controllers are still programmed in ladder logic. This progress is summarized in Table 2-1.

Table 2-1 Properties of the four past generations of industrial automation systems and prospective intelligent automation features

	Generation 1 **Relays**	Generation 2 PLC	Generation 3 High-end PLC	Generation 4 Embedded control	Generation 5 I n t e l l i g e n t Systems
Automated routine operations	x	x	x	x	x
Programmability		X	X	X	X
Configurability			X	X	X
Distributed flexible architecture				X	X
Self configurability					X

Review questions for Chapter 2

1. The code in Figure 2-2 is simplified. Will it perform as intended? If not, why not? What corrections are needed to have it perform as intended?
2. Summarize the limitations of the first four generations of automation technologies.
3. Sketch the functions that intelligent components could provide in the FLASHER example.

3

Automation: From mass production to flexibility

This chapter discusses the requirements for automation systems imposed by new manufacturing paradigms, such as production to order, small lot sizes and re-configurability. Current practices in industrial automation and key problems with centralized control within this context are described. This discussion considers the list of specifications for the next generation of control systems.

Industrial trends

Flexibility has been a central theme of industrial and academic research in manufacturing since the latter half of the twentieth century. In the post-World War II United States, for example, the virtual lack of competition internationally combined with a domestic population eager for new products allowed manufacturers to focus myopically on production efficiency rather than on the consumer [Sipper and Bulfin, 1997]. In this environment, customers could be taken for granted (i.e., they were more than pleased to buy what was offered). As a result, mass-production of high-quality, standardized goods was the norm and efficiency of production was all-important.^{1}

The latter part of the twentieth century, however, saw more sophisticated consumers and an increase in foreign competition that quickly made this approach obsolete. Customers became more demanding and started to seek more variety, lower costs, and superb quality; thus, they no longer had any qualms about where their products were manufactured. As a result, flexibility and responsiveness overtook efficiency of production as the key benchmarks for world-class manufacturing.

Among the first to address this issue in detail was the Massachusetts Institute of Technology team, led by James Womak et al. (1991), which studied "the future of the automobile." This group examined how Japan was able to dominate the automotive industry in the 1970s and 1980s through their move from mass production to lean production.

Mass production differs from lean production in several ways. For example, mass production sets limited "good enough" goals (e.g., "acceptable number of defects" and "acceptable low product variety"). Continuous improvement was never an important consideration. By contrast, lean production focuses on high performance goals, such as zero defects, declining costs, higher flexibility and more product variety, all of which are directly aligned with current customer requirements. In their study, Womack et al. (1991) focused on the importance of produced. The machines themselves are usually equipped with a control device that executes pre-programmed functions; however, the integration with other machines may bring certain changes in the way the machine operates.

Internally, such machines are often composed of unified modules that are also controlled by embedded devices and capable of performing certain pre-defined operations. Even the modules inside can contain some intelligent mechatronic devices, like pneumatic cylinders or sensors with network connection (even wireless), which are "intelligent" to a certain extent.

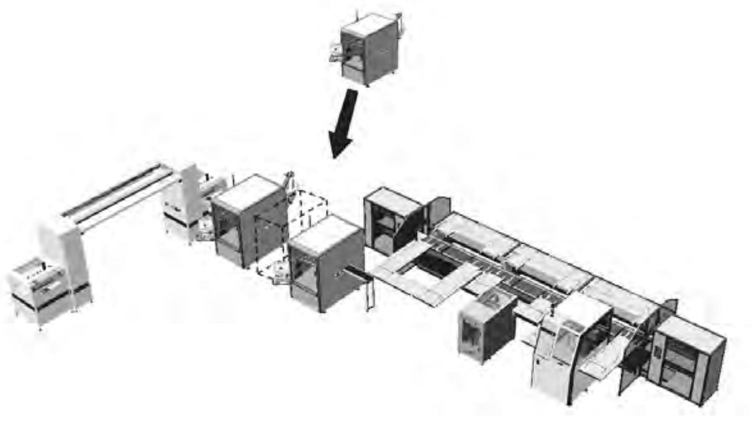


Figure 3-1 Typical modular production system from the light-assembly domain.[2]

Thus, the whole production system is very software and communication-intensive. Configuring and maintaining such production systems requires software tools that can access and program each device, download and upload software blocks, and perform dynamic remote debugging.

Naturally, this requires interoperability across tools, networks, and devices, or even better—the ability to use a single tool to program and test all devices, which employs a middleware that provides transparent communication independently of the particular combinations of networks and fieldbuses, and which has programmable compatibility of all embedded hardware/software platforms.

The modular nature of current production systems has led to a set of specific needs for automation and control. In particular, these systems should be:

Adaptable to heterogeneous environments—They should cover the whole range of available and prospective hardware platforms from embedded on-chip controllers through the range of traditional programmable logic controllers (PLCs) and their followers, up to the level of industrial personal computers and PC-based devices.

Capable of working in distributed networks—They should provide transparent and reliable access to all sorts of the data from values of sensors and actuators via field buses up to the intranets of enterprises and the Web.

Based on a modular open architecture—The architecture should ensure interopera-

- Human-Machine Interface (HMI) components, and
- software agent components.

Industrial automation and control systems have a huge base installed with physical equipment that may have a lifetime of 10 years or longer. Hence, an important requirement for timely adoption of any specific architecture will be its provisions for *retrofit* of existing systems and their *migration* to the new architecture over time.

The architecture must be *open*; that is, it must exhibit the following characteristics as illustrated in Figure 3-5:

- **Portability—**Software tools and agents can accept and correctly interpret library elements (software components and system configurations) produced by other software tools.
- **Interoperability—***Devices* can operate together to perform the *autonomous* and/or *cooperative* functions specified by one or more distributed applications.
- **Configurability—***Devices* and their *software components* can be dynamically *configured* (selected, assigned locations, interconnected and parameterized) by multiple software *tools* and/or software *agents*.

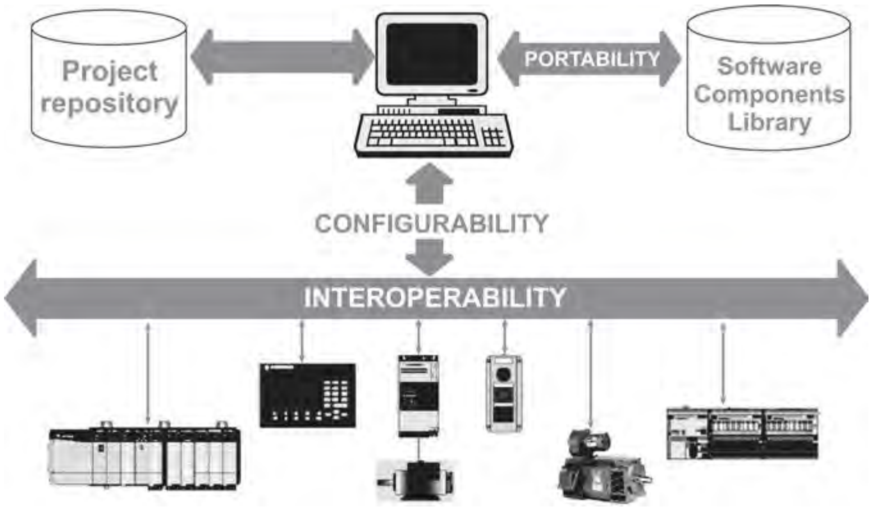


Figure 3-5 Attributes of open industrial systems.[3]

Summary

Automation systems are becoming more and more software intensive, which increases their flexibility. But the increasing complexity of the software portion compromises flexibility and reconfigurability. The next generation of automated systems will be adaptable to heterogeneous environments, be capable of working in distributed networks, be based on a modular open architecture, will incorporate human interfaces based on intelligent software and hardware, and be scalable and fault tolerant. The main workhorse of industrial automation systems are PLCs. But they were designed for centralized automation systems and are not easily integrated with the newly emerging distributed systems. The next-generation automation architecture will bring better encapsulation of automation-related intellectual property to software components and their portability, along with interoperability and configurability of automation devices.

Review questions for Chapter 3

1. What is the driving force behind the progress in industrial automation technologies?
2. What are the attributes of an open architecture?
3. Why do the general-purpose software technologies have only limited application in industrial automation?

Notes

1 The section "Industrial trends" was completely written by Robert Brennan, Professor at the University of Calgary, Canada.

2 Picture courtesy of Professor José Luis Martinez Lastra, Tampere University of Technology, Finland.

3 Picture courtesy of Dr. James Christensen, Holobloc Inc., USA.

4

Horizons of distributed intelligent automation

This chapter discusses new technical opportunities that are provided by IEC 61499 from the perspective of intelligent automation systems.

Autonomous intelligent devices

The requirements of flexible manufacturing (e.g., the need for rapid integration) and reconfiguration, as well as the growing information intensity of the production environments imply that manufacturing equipment is getting more autonomous and intelligent. Figure 4-1 exemplifies a list of functions that an intelligent device may need, using the boring mill as an illustration.

Intelligence at the device level is intended to increase functionality, performance, reliability, and to facilitate the device's integration into more complex production systems. Later on, this chapter will consider some of these functions and then discuss implementation challenges.

Connectivity and web services—Network connectivity is a very basic function of an intelligent device. An intelligent device must be capable of communicating with other intelligent devices, as well as with enterprise information systems and human operators. It is quite easy to imagine a web server embedded to the device, which can provide direct "remote control" access to authorized clients as illustrated in Figure 4-2. In applications where the Internet Protocol (IP) does not provide sufficient real-time properties, more specific industrial network protocols can be used, which are known as *fieldbuses* (e.g., Profibus, DeviceNet, Modbus, and many others; see Zurawski 2005 for details). More protocols can be supported with corresponding software components. Thus, an intelligent device may need flexible mechanisms for adding more software components to extend its communication capabilities.

System modelling and simulation

Intelligent automation devices can be relatively autonomous. When integrated into a manufacturing system with other autonomous intelligent devices, however, several important issues must be considered:

Analysis—Properties of the resulting system must be checked before its operation can be started, and this can be done either by simulation or by formal verification[1].

The simulation scenario can be implemented as illustrated in Figure 4-3. First, autonomous devices provide descriptions of their functionality in the form of function blocks for

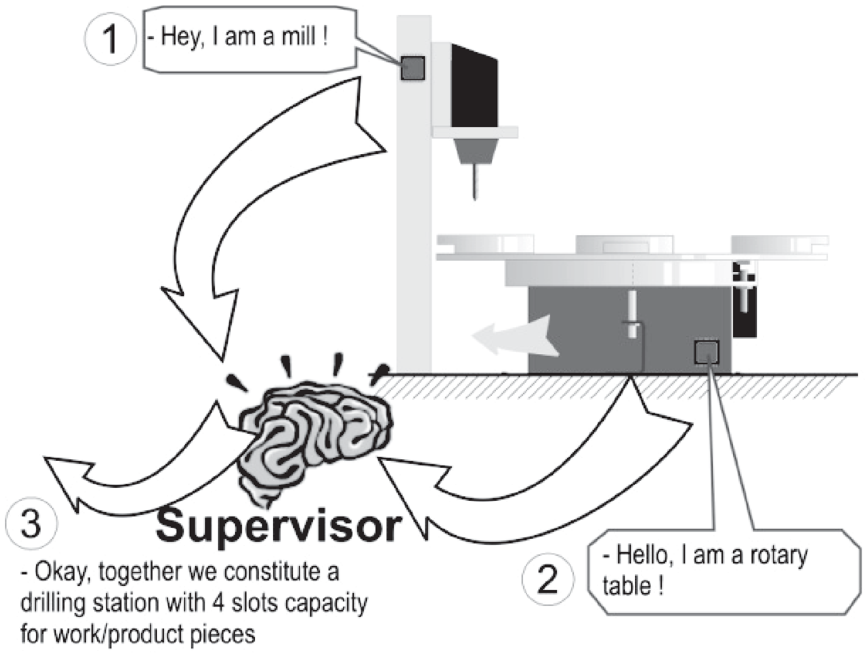

1
- Hey, I am a mill !
2
- Hello, I am a rotary table !
3
Supervisor
- Okay, together we constitute a drilling station with 4 slots capacity for work/product pieces

Figure 4-6 Intelligent negotiation and formation of manufacturing cells.

Various business-related considerations may require the use of different automation hardware to control a manufacturing cell, as illustrated in the example below. This implies the need for portability and distribution of software components implementing functionality of autonomous intelligent devices. Ideally, it should not matter where the function blocks are physically executed as long as they communicate via some transparent media.

Some extreme cases are shown in the following figures. In the hardware configuration shown in Figure 4-9, a central controller collects the function blocks from the embedded data storage devices and executes them in a cyclic way like in PLCs. The data can be directly received by the controller through its own peripheral modules or via a network. The use of IEC 61499 function blocks would make the order of execution irrelevant, so that the configuration can be carried out even automatically as the mechatronic units are plugged into the net.

the particular number and topology of computational resources, nor on their cyclic scans. This means that no additional synchronization between devices will be required. This will be true also for more complex applications that are represented as hierarchical networks of function blocks. Thus, system engineering can now be done in an implementation independent way.

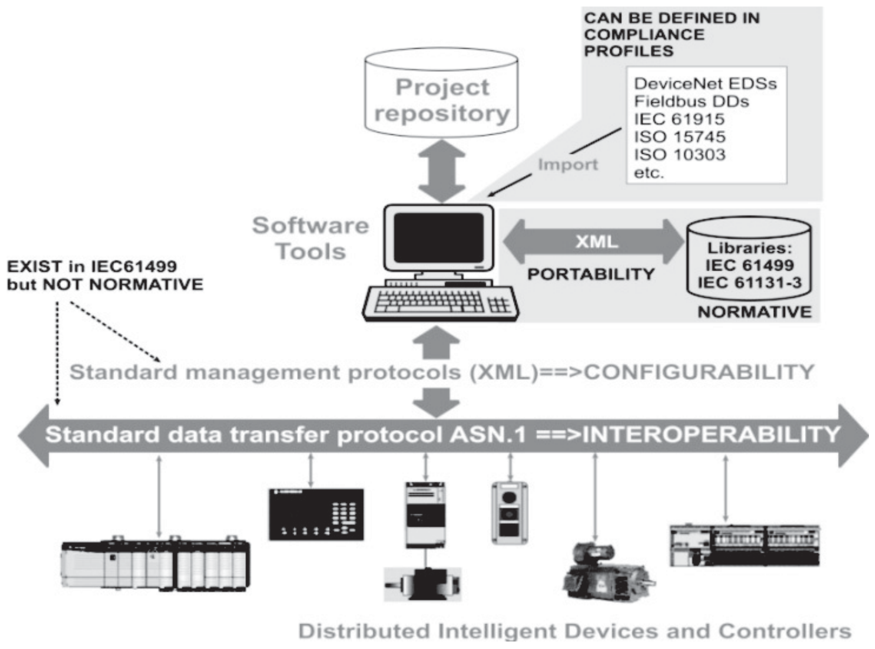

CAN BE DEFINED IN COMPLIANCE PROFILES
Project repository
DeviceNet EDSs
Fieldbus DDs
IEC 61915
ISO 15745
ISO 10303
etc.
Import
Software Tools
EXIST in IEC61499 but NOT NORMATIVE
XML
PORTABILITY
Libraries: IEC 61499 IEC 61131-3
NORMATIVE
Standard management protocols (XML)==>CONFIGURABILITY
Standard data transfer protocol ASN.1 ==>INTEROPERABILITY
Distributed Intelligent Devices and Controllers

Figure 4-11 Means of achieving open architecture in industrial automation provided by IEC 61499.[2]

- An abstract *Device Model* provides a mechanism for creating new device types as a set of resource types and function block libraries. This method of device modelling is very modular and uses a limited number of instruments. It allows modelling a great variety of system configurations with sufficient precision but without unnecessary detail.
- The open device management mechanism of IEC 61499 provides configurability of compliant devices by compliant software tools.
- Service Interface Function Blocks provide a mechanism for encapsulation of hardware dependent functions into the same language constructs as the rest of the application.
- Part 2 of the Standard defines the XML-based data format of function blocks and also provides compatibility on the level of source files.
- Specific protocols of device configuration or management may be defined as compliance profiles to the standard. Part 4 of the Standard defines the rules regarding the organization of compliance profiles.
- The ASN.1 protocol enables platform independent data representation.

- Main IEC 61499 benefits for PLC users
- *Communication* between parts of the program is based on events. This makes the logic clearer and independent from the components' location with respect to one another in the program;
- *Performance:* reaction time of event-driven processing can be much faster than the scan-based execution of PLCs.
- *Portability* enabled by the open XML format of IEC 61499;
- *Migration and maintenance*: the algorithms in basic FBs still can be programmed in the familiar
- PLC languages simplifying the migration and maintenance.

Potential benefits of IEC 61499 for embedded control users

IEC 61499 is a system-level architecture widely sought after in the embedded systems world. The architecture enables design and re-design of distributed embedded control systems within a single development framework. It also provides some features of model-based software engineering, such as state-machine programming of basic function blocks. These features increase the level and efficiency of programming when compared with the standard high level programming languages, such as C or C#.

Programming in function blocks increases the re-use potential of the developed code and the potential of deploying it to various distributed hardware architectures. The code semantics becomes clearer thanks to the visual block—diagram and state machine representation.

Summary

Distributed automation will enable intelligent machines to implement a custom set of functions by means of adding/removing software components—called in IEC 61499 function blocks. These functions may include:

- System simulation and modelling,
- Programmability,
- Service provision mechanism, and
- Ability to be integrated with other intelligent machines in a seamless way.

The IEC 61499 architecture provides the ability to execute the same application on different distributed hardware structures with minimal or no change. The IEC 61499 provides a number of key technologies for encapsulation, portability, interoperability and configurability of heterogeneous distributed automation systems.

Review questions for Chapter 4

1. What is portability? How is it achieved in IEC 61499?
2. Explain the means by which the IEC 61499 standard fosters *device interoperability*.
3. Why is re-use important? What can be re-used? How does IEC 61499 facilitate re-use?
4. If several automation companies are supplying their devices for an automation project, what does *interoperability* mean for them? How does one achieve interoperability.

Notes

1 Formal verification is a validation technique that provides formal proof of certain properties of a system. For example, formal verification of IEC 61499 was addressed in (Vyatkin, Hanisch 2003).

2 Picture courtesy of Dr. James Christensen.

5

Basic concepts of IEC 61499

This chapter presents the basic concepts of the IEC 61499 paradigm: the concept of event and of event- driven invocation of software components; types and instances of function blocks; and the interface of function blocks.

As shown in the previous chapter, the new generation of automation systems requires a better software component model than a program or a function block of IEC 61131-3. The component model proposed in IEC 61499 also bears the same name, *function block,* although its meaning is completely different. Before explaining the differences, a few new fundamental concepts must be introduced. One of these is *event*.

Events

In IEC 61499, a new concept of function block invocation is introduced, which is based on *events.* Information about events is passed from one function block to another by means of *event variables*. In a slightly different form, the event concept is also used in automation systems engineering as falling and rising edge variables, which indicate value changes introduced by a Boolean variable.

An *event variable* can have the same 0 or 1 value as Boolean variables, but value 1 can be taken only for instantaneous moments. For example, events may indicate changes of values in Boolean data as shown in Figure 5-1. Obviously, events have no duration. Event variables can be used in programming in the same way as Boolean data is used—you can check them in conditional operators like IF-THEN-ELSE.

Events can originate in the environment (in the controlled object) and then activate certain function blocks. Moreover, in the IEC 61499 function blocks can be activated only by events (event input variables).

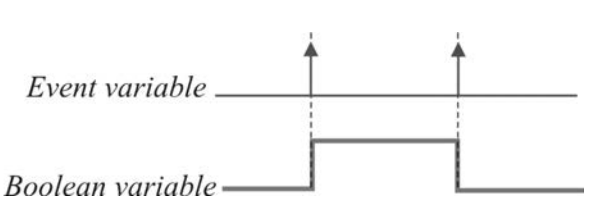


Figure 5-1 Event versus Boolean variable.

Function blocks

The function block of IEC 61499 is an abstraction that represents a component, which is usually implemented in the form of software, but can also take the form of hardware. That is why it is an *abstract* yet *executable description of* functionality, which is not dependent on a particular implementation.

The interface of a function block (Figure 5-2) is defined by the list of input event variables, input data variables and adapter sockets, output event variables, output data, and adapter plugs. In graphical representation, the block has “head” and “body.”[1]

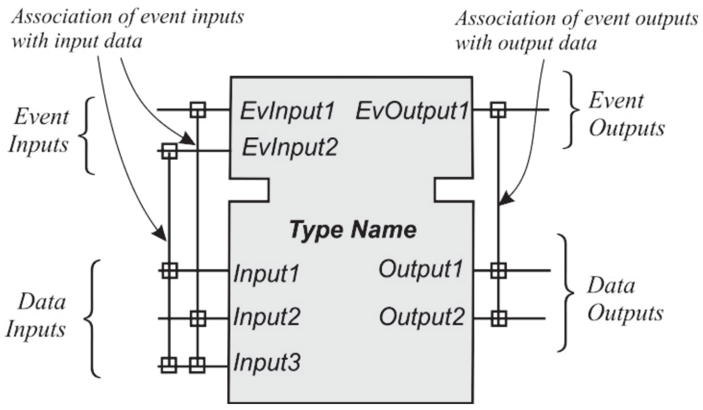


Figure 5-2 Interface of a Function Block.

The upper part of the block’s interface is often referred to as the “head” and the lower as the “body” of the function block interface. Event input/outputs are listed in the head, while data inputs and outputs are connected to the body.

Events stand for the synchronization and the interactions among the execution control mechanisms in interconnected networks of function blocks. A block sending data must emit an event and pass it on to the receiving block in order to ensure the data are read.

The data are typed, as in any programming language. The 61499 standard refers particularly to the data types of IEC 61131.

The definition of the function block’s external interface also includes an association between the events and relevant data. In the graphical representation of a function block, the association is denoted by vertical lines connecting an event and its associated data inputs/outputs. The association literally means that the values of the variables associated with a particular event (and only these!) will be updated when the event occurs.

Example 1 *How the event/data association works at the data transfer between blocks.*

In the example given in Figure 5-3 the event output “*eo*” of the function block FB1 is connected by an event connection with the event input “*ei*” of the function block FB2. Once the block FB1 emits the event “*eo*”, it triggers the execution of the block FB2. The values of input parameters “*d*” and “*e*” will be updated before the execution starts because they are associated with the event input “*ei*”.

6

NxtStudio Integrated Development Environment

This chapter provides essential information about the NxtStudio function block tool. In the German edition of this book, NxtStudio has been selected as most representative tool based on IEC 61499. We discuss several other tools for function block design in Chapter 19.

General information on NxtStudio

NxtStudio is an engineering support software tool for IEC 61499 software development. It is a product of the Austrian company nxtControl GmbH. As shown in Figure 6-1, it contains the integrated development environment NxtStudio and the run-time target nxtRT61499F. The integrated development environment (IDE) supports graphical development of function blocks and systems and their translation to executable code. The latter can be executed using nxtRT61499F.

Versions of the nxtRT61499F software run-time target are available for personal computer and for a number of embedded hardware platforms.

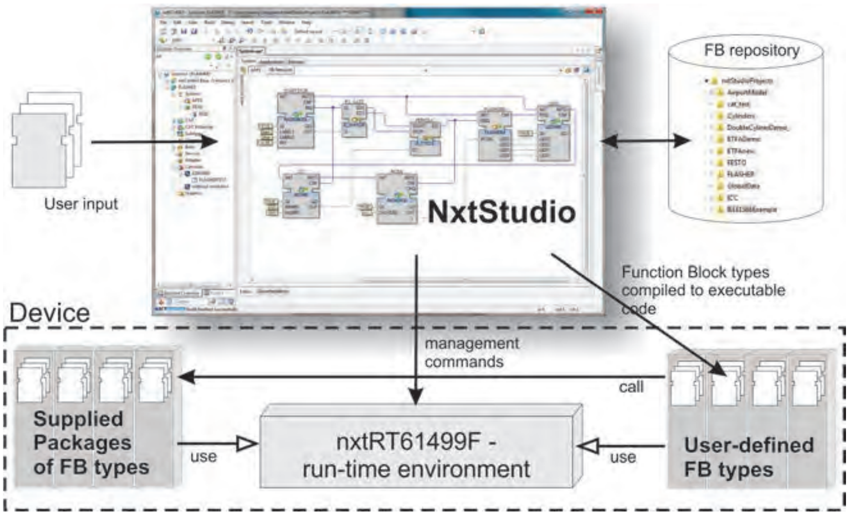

FB repository
User input
NxtStudio
Function Block types compiled to executable code
Device
management commands
call
Supplied Packages of FB types
use
nxtRT61499F - run-time environment
use
User-defined FB types

Figure 6-1 Structure of NxtControl software: NxtStudio IDE interacting with run-time environment and libraries.

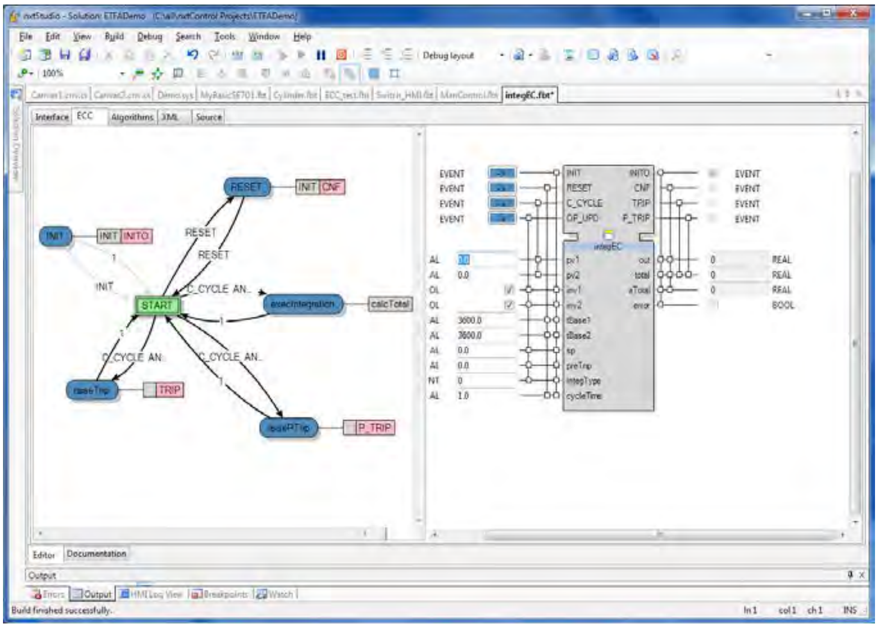


Figure 6-2 Testing environment for basic function blocks in NxtStudio.

The editor of composite function blocks and of applications implements several “tricks” improving the readability of block diagrams. For example, it automatically creates connection points on event and data connection lines, to reduce duplication of such lines cluttering the diagram. This is illustrated in Figure 6-3.

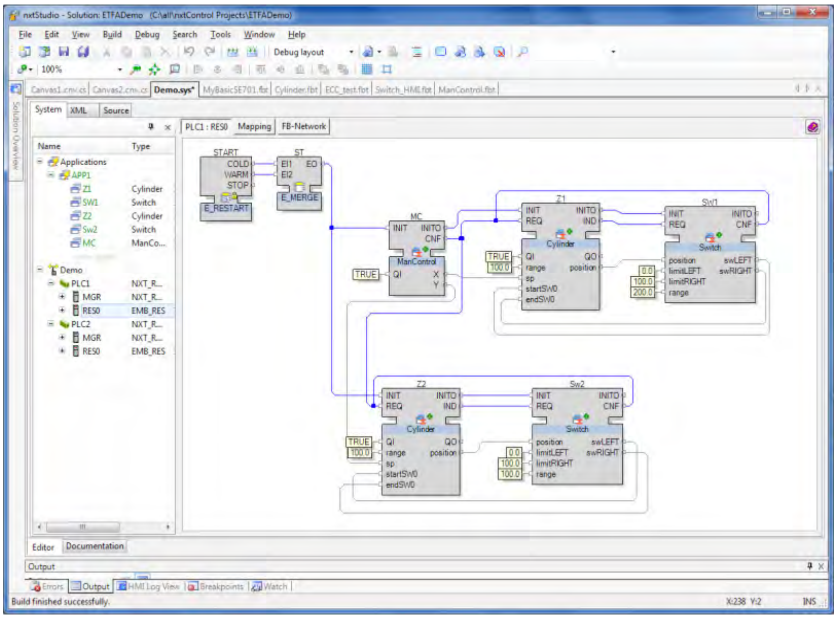


Figure 6-3. A function block application in NxtStudio.

Execution

For execution of function block applications, NxtStudio uses a modified open source run-time platform FORTE. NxtStudio compiler generates C++ code from function blocks, so that one function block type is compiled to one C++ class. C++ code of the entire application is compiled then to the particular target microprocessor code using a cross-compiler. It is linked with the run-time libraries responsible for scheduling events.

Pilot applications

NxtControl has already applied their technology in a few dozens of projects in the domain of building automation. The largest project has been a training centre building with 19 control devices controlling about 2500 I/Os (heating, ventilation, air-condition, lighting, etc.) with IEC 61499.

The range of hardware platforms, compliant with NxtStudio includes industrial personal computers (IPC) of Siemens (MicroBox), Beckhoff (CX 1020), and WAGO (IPC 871). Some of these devices are shown in Figure 19-29. Similarly to PLCs, these devices are hardened and certified for use in tough industrial environments.

a)

b)

Figure 6-4 a) WAGO IPC 758-870; b) Beckhoff CX 1020.

Focusing on the European automation market, NxtControl has developed support of such field area networks as Profibus and Ethernet. Via these interfaces the compliant devices can communicate with a broad spectrum of automation devices, such as motion control devices.

The application areas where the use of NxtControl software can be especially efficient include the ones with highly modular machines, such as material handling systems. Figure 6-5 shows an example of a material handling automation application implemented in NxtStudio. One can see a clear similarity between the structure of the physical part (right top) and the structure of the program (right bottom). The function blocks are CAT instances, implementing control and visualisation of a conveyor group.

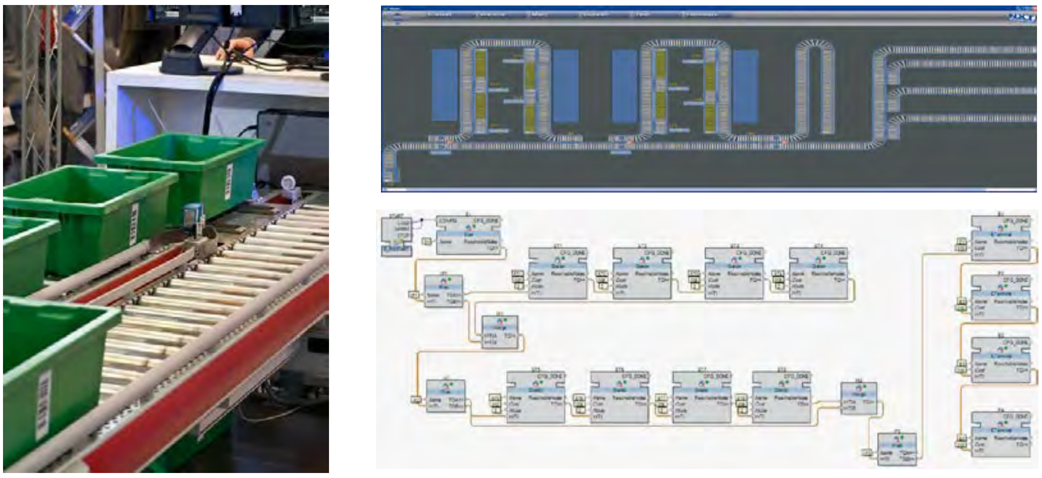


Figure 6-5 Material handling application of NxtControl.

Summary

NxtStudio is a second generation IEC 61499 tool with well developed functionality and extensive set of supported services. Along with that, the tool introduces several restrictions on the implemented features of IEC 61499: thus it does support only ST language for programming algorithms in basic function blocks and does not support user-defined data types.

7

Basic function blocks

This chapter introduces the basic function block concept, which is an abstraction for a software capsule. It describes the interface, the execution control chart, the various ways to program algorithms and the behaviour of basic function blocks.

Capsule for functions

Basic function blocks are software structures intended to be used in the implementation of basic functions of distributed control applications. A basic function block may have *internal variables*, one or several *algorithms,* and an *execution control function*, defined by means of the *execution control chart* (ECC), shown in Figure 7-1. A function block's internal data cannot be accessed from outside and can be used only by the internal algorithms of the particular function block. An algorithm is a programming structure of the finest granularity. It represents a piece of software code operating on the common input, output and internal data of the function block.

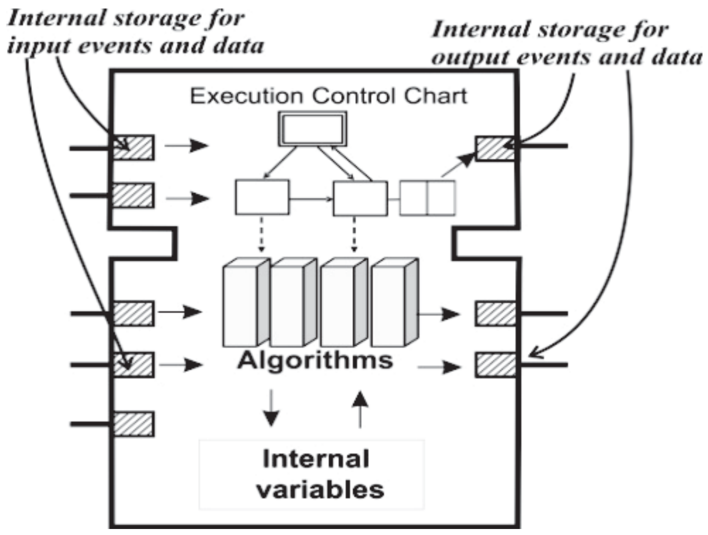

Internal storage for input events and data
Internal storage for output events and data
Execution Control Chart
Algorithms
Internal variables

Figure 7-1 A basic function block: interface, execution control chart, algorithms, and internal variables.

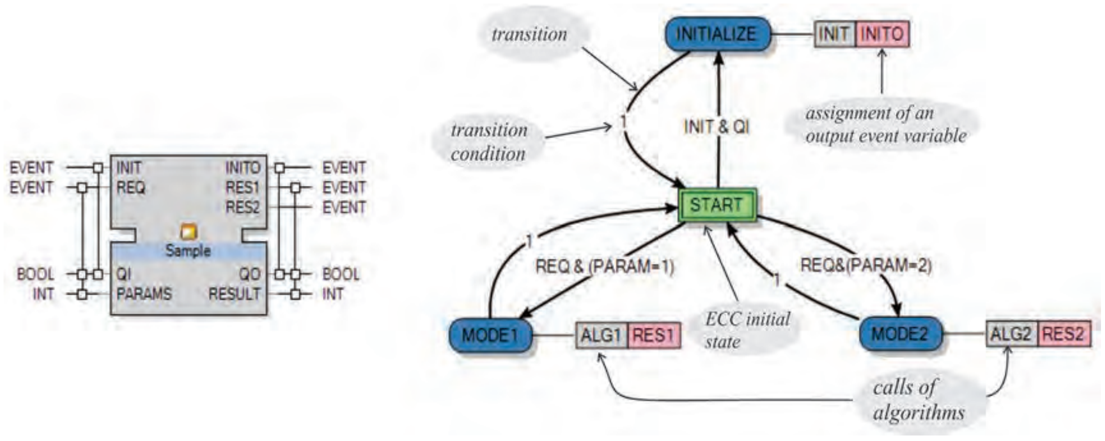


Figure 7-3 An example of a basic function block with its execution control chart.

Syntax of ECC transitions

Each state's transitions are labelled with a condition that is a Boolean expression whose syntax is limited to one of the following forms:

```
event_input_name guard_condition
event_input_name ('&'|'AND') (guard_variable| '(' guard_con-
    dition')')
```

The guard variable is an input, output or internal variable of the functional block of BOOL type. The guard_condition is a Boolean expression not containing events. It shall evaluate to a BOOL value. Examples of correct condition expressions for the function block type in Figure 7-3 are as follows:

Form 1	**Form 2**	**Form 3**
REQ	**`(PARAM=1) OR (PARAM>3) OR NOT QI`** *This is a Boolean expression over input variables of integer and Boolean type. It follows the syntax defined for Boolean expressions in Structured Text of IEC 61131-3*	$\underbrace{\texttt{REQ}}_{\text{event input name}}\ \overset{\text{AND operation}}{\texttt{\&}}\ \underbrace{\texttt{(PARAM=1)}}_{\text{guard condition}}$

Figure 7-4 Forms of transition condition expressions.

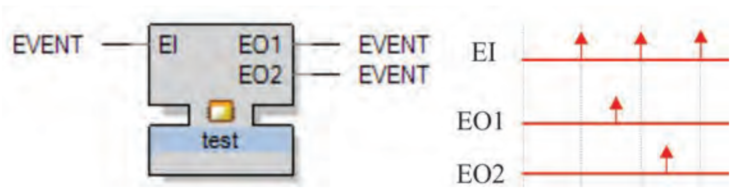


Figure 7-9 Behaviour of an event demultiplexer function block.

A straightforward (but incorrect) solution attempt, illustrated in Figure 7-10, emits both output events simultaneously on the first occurrence of EI. Finding a correct solution is left to the reader as an exercise!

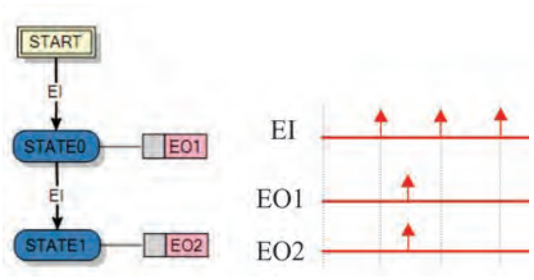


Figure 7-10 Incorrect implementation of the “event demultiplexer”.

To facilitate implementation of such event behaviour NxtStudio provides an option of “clearing” the event-input variables by assigning them with value FALSE. For example, one can create an algorithm CLEAR with the content: EI:=FALSE and attach it to the action in STATE0.

Standard libraries

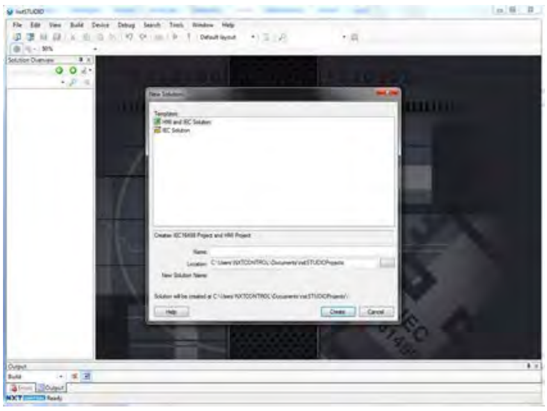
The IEC 61499 defines a number of standard function blocks for manipulations with events, such as splitting or merging events, generation of events with delays or cycles, etc. Descriptions of these blocks are provided in Appendix A.

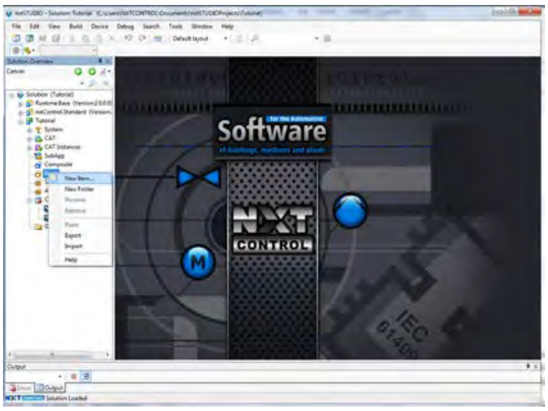
Tutorial 1: Creating a new basic function block type

The goal of this tutorial is to get practical experience of developing function blocks in NxtStudio.

In NxtStudio one cannot create just a function block type, but only within a solution that is a structure analogous to project in other programming environments. In more detail this structure will be discussed later, in Chapter 11.

Let us create a new solution with name 'Tutorial' as shown in Figure 7-11(a), then position the cursor at the Basic item in the solution tree, press right mouse button and select New Item from the pull down menu. In the appeared dialog assign name X2Y2 to this function block type.

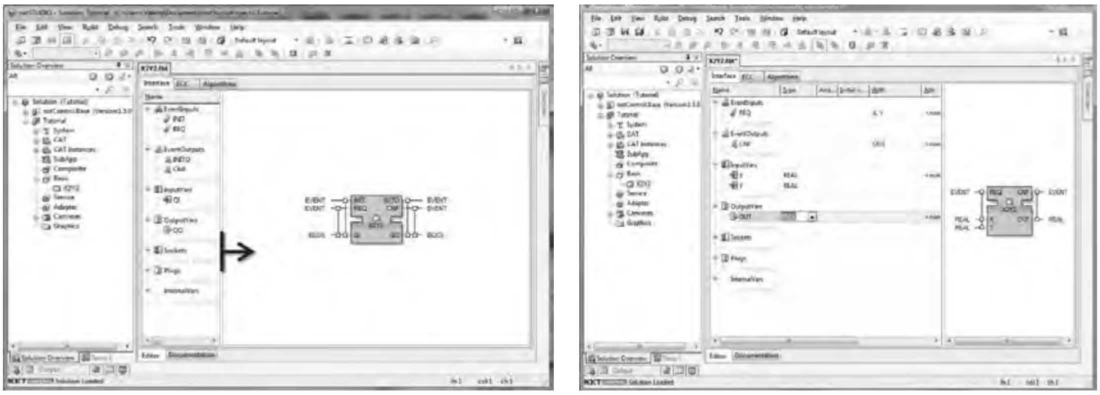




a) b)

Figure 7-11 (a) Creating a new solution; (b) Creating a new basic function block type in the solution.

You will see function block type created with a standard interface as shown in Figure 7-12(a). Drag the boundary of Interface pane to the right and edit the interface elements in the table, as shown in Figure 7-12(b). For that, we deleted event inputs INIT and INITO, and renamed data intputs QI and QO to X and OUT, changing their types to REAL. Then we added one more input Y, also of type REAL.

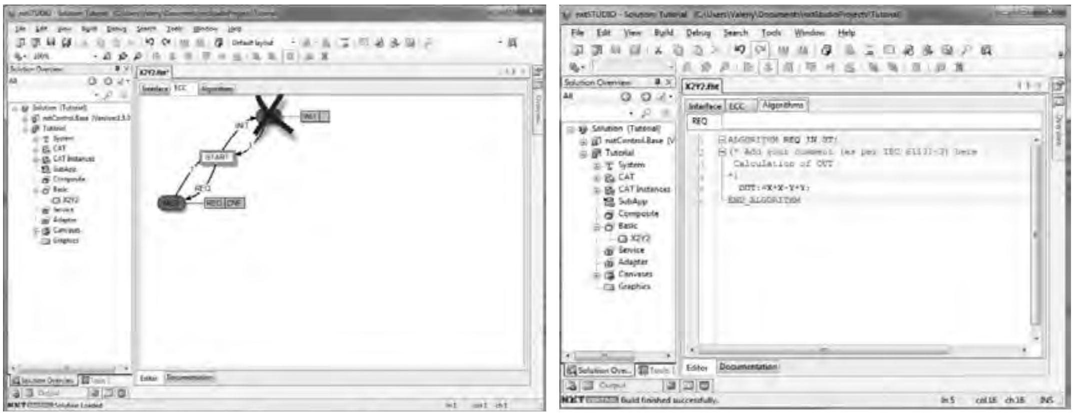


a) b)

Figure 7-12 (a) Opening the newly created FB type; (b) Modifying interface of the created FB type.

To create an association between an event input/output and data input/output, use field "With" in the interface table.

Now, switch to the ECC tab and edit the ECC as illustrated in Figure 7-13. You will need to delete state INIT because in this simple example we don't need initialization. To do so, hover mouse over the state and make a right click. Then select the "Delete" option from the menu. Alternatively, select the state with left mouse click and press Del key. After that, delete the algorithm INIT (right click at the tab and select "Delete algorithm") and edit the algorithm REQ as shown in Figure 7-13(b).



```
ALGORITHM REQ IN ST:
(* Add your comment (as per IEC 61131-3) here
  Calculation of OUT
*)
  OUT:=X*X-Y*Y;
END_ALGORITHM
```

Figure 7-13 (a) Delete state INIT in ECC; (b) Modify algorithm REQ.

8

Composite function blocks

This chapter describes the composite function block, which is needed to develop some functions based on a modular combination of other function blocks and to encapsulate them into a single software component.

Along with basic function blocks, the *composite function blocks* are intended to be the main instruments of an application developer. The functionality of composite function blocks, in contrast to basic function blocks, is determined by a network of interconnected function blocks inside. Figure 8-1 illustrates the idea.

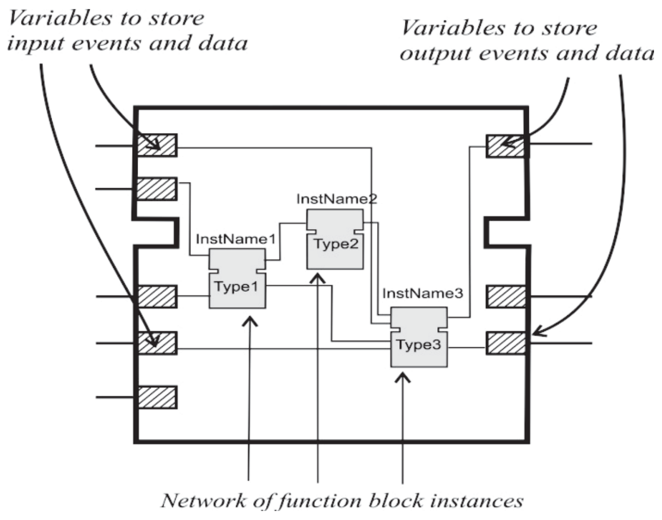

Variables to store input events and data
Variables to store output events and data
InstName1
Type1
InstName2
Type2
InstName3
Type3
Network of function block instances

Figure 8-1 Composite function block.

More precisely, the network is built from instances of function block types. These can be basic function blocks, service interface function blocks, or other composite function blocks. Thus, hierarchical applications can be built as shown in Figure 8-2.

Composite function blocks have no internal variables, except for those storing the values of input and output events and data. Thus, the functionality of composite function blocks completely depends on the *state* and *behaviour* of the constituent function blocks and their interconnections by events and data.

9

Applications and sub-applications

This chapter discusses applications and sub-applications. An application is a network of function blocks that completely defines the desired functionality of an automation system but does not say anything about number and types of devices on which it will be executed. Thus, an application is still a machine-independent functionality description.

Application

An *application* is defined in IEC 61499 as a network of function block instances whose data inputs and outputs and event inputs and outputs are interconnected (Figure 9-1).

An application does not have an interface, nor does it own variables. Its behaviour is completely defined by the function block instances and their interconnections by event and data connections. Application is an abstract definition of the desired behaviour of a system. It defines the function explicitly but its execution semantics cannot be completely determined before function block instances are associated with particular resources and devices.

An application can be considered as an intermediate step in system development. Its role is to capture functional and structural properties of the system in a platform independent way.

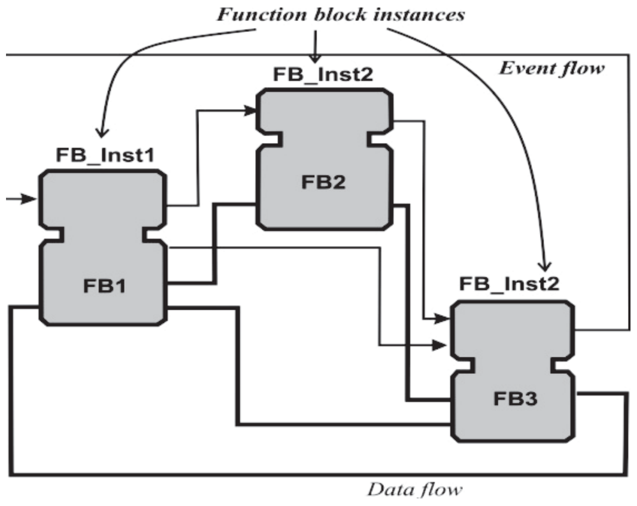


Figure 9-1 An application is a network of function blocks.

10

Models for devices and resources

This chapter discusses models for devices. In the context of this book, a device is understood as a functional unit like a computer, microcomputer, or an embedded chip that is capable of interacting with automation equipment and process information. Examples of devices are programmable logic controllers, on-chip controllers, embedded chips of intelligent sensors and actuators, numeric controllers, network hubs, gateways, and so on (see Figure 10-1).

Device

A *device* in IEC 61499 is an abstract model that captures the information processing properties of embedded computing and control devices. A device is an atomic element of a system configuration. The standard provides an architectural framework for creating models of devices, including their subdivision on computationally independent resources.

A *device type* (Figure 10-2) is specified by its process interface and communication interfaces. A device can contain zero or more resources (see the description of resource below) and a function block network (this option, however, is reserved for devices having no resources).

A “process interface” provides a mapping between the physical process (e.g., analog and discrete sensors and actuators) and the resources. Information received from the physical process is presented to the resource as data, events, or both.

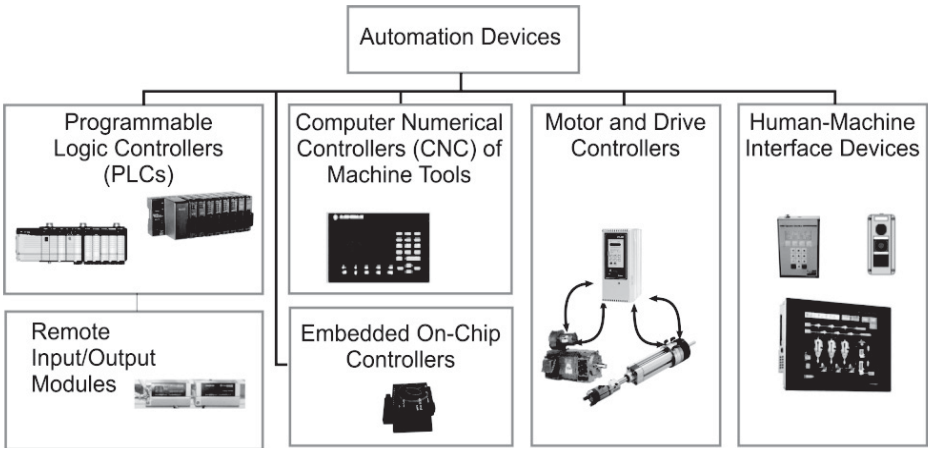

Automation Devices
Programmable Logic Controllers (PLCs)
Computer Numerical Controllers (CNC) of Machine Tools
Motor and Drive Controllers
Human-Machine Interface Devices
Remote Input/Output Modules
Embedded On-Chip Controllers

Figure 10-1 Examples of automation devices.

11

Distributed system configurations

This chapter describes the concept of a system. A system configuration is the most complex structure of the IEC 61499 architecture. It includes instances of device types along with function block applications.

System configuration

A system configuration is described in IEC 61499 as a set of devices (instances of pre-defined device types) populated by function blocks in one or several applications, as illustrated in Figure 11-1. The devices can communicate with each other over communication networks. Also, the devices are linked with the controlled process via sensor and actor signals.

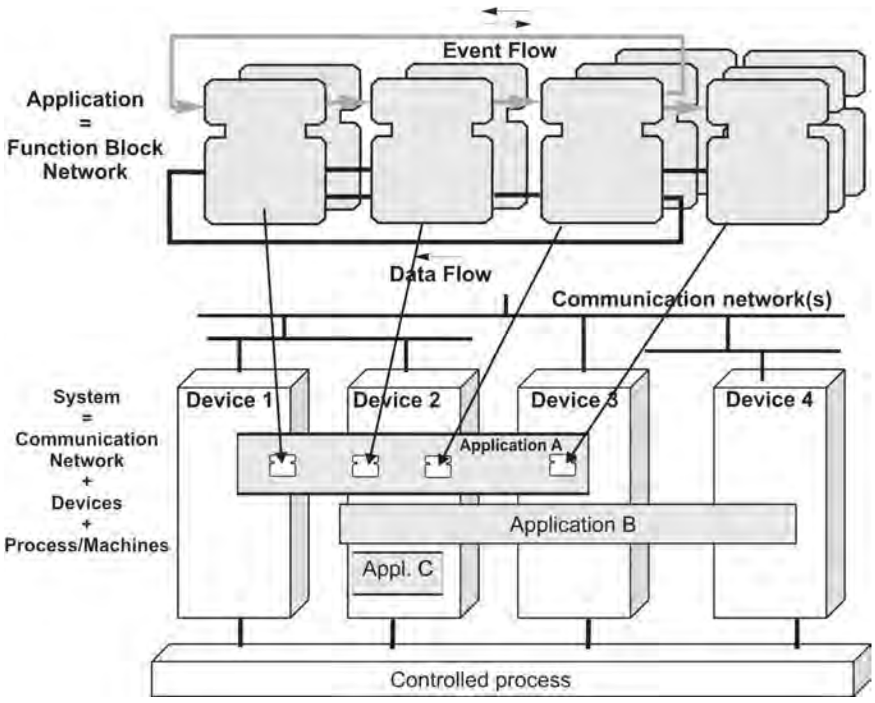


Figure 11-1   A system configuration.

Applications are said to be *mapped* on a particular architecture of devices, as illustrated by the arrows in Figure 11-1. This means that the function blocks of the application are assigned to the resources of the corresponding devices. In this way, a system configuration is formed.

output values and sends them to the device DISPLAY, given the input parameters received from the CTL_PANEL. There are two communication flows: between CTL_PANEL and FLASHER device, and between FLASHER and DISPLAY devices.

The FLASHERR system configuration can be executed on a network of several (1 or 2) network connected devices.

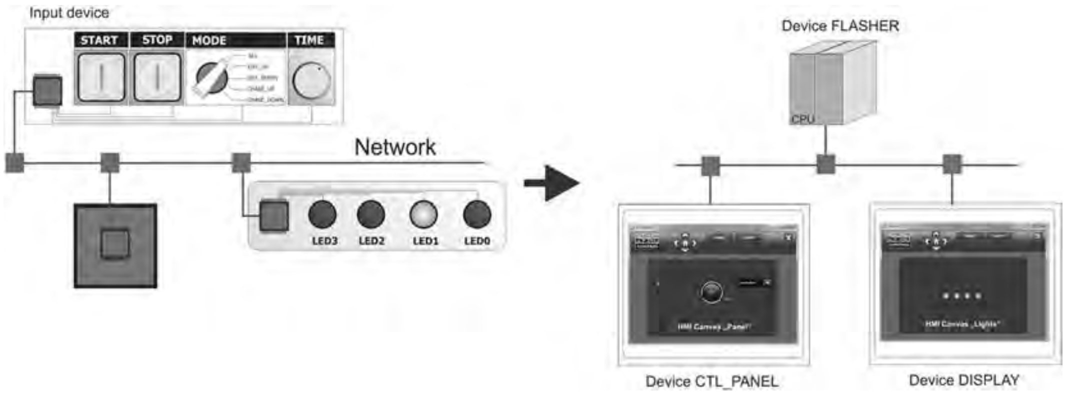
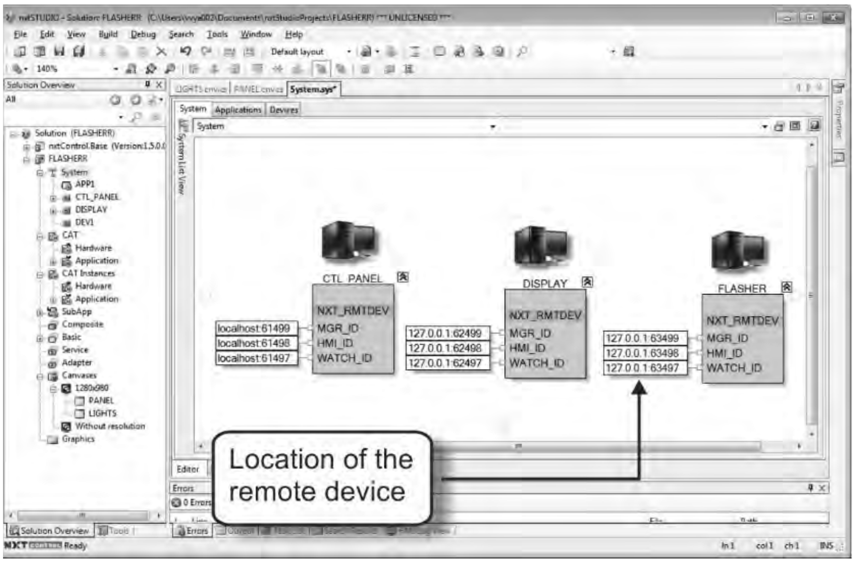


Figure 11-7 FLASHER implemented with three devices.



```
localhost:61499    MGR_ID
localhost:61498    HMI_ID
localhost:61497    WATCH_ID
```

NXT_RMTDEV (CTL_PANEL)

```
127.0.0.1:62499    MGR_ID
127.0.0.1:62498    HMI_ID
127.0.0.1:62497    WATCH_ID
```

NXT_RMTDEV (DISPLAY)

```
127.0.0.1:63499    MGR_ID
127.0.0.1:63498    HMI_ID
127.0.0.1:63497    WATCH_ID
```

NXT_RMTDEV (FLASHER)

Figure 11-8 Devices of the FLASHER_TEST3D system: two local frame devices (CTL_PANEL and DISPLAY) and a remote device FLASHER.

12

Service interface function blocks

This chapter describes the concept of service interface function blocks, which are explicitly defined mechanisms for interacting function block applications with hardware resources. For example, function blocks that read the values of input variables from peripheral input modules are service interfaces.

Services

In general-purpose programming languages, like C or JAVA, most of the structures work similarly on different hardware and software platforms, while some of them, such as those commands for reading input or displaying graphics require implementation specific to a particular operating system or hardware platform. The same is true for measurement and control applications. While the control logic can be the same, the functions that read values of sensors and set values to actuators will differ from platform to platform. Reading inputs is a *service* that a particular *resource* provides to the function blocks allocated to it.

The *service interface function blocks* serve this role, wrapping the hardware dependencies and allowing the application developer to focus on the application logic. Their implementation requires knowledge of the low-level details of particular hardware. Thus, in contrast to basic and composite function blocks, service interface function blocks are not intended to be developed by an application developer. They have to be provided by vendors of the corresponding equipment (e.g., of controllers, fieldbuses, remote input/ output modules, intelligent sensors, and so on).

To hide the implementation-specific details of the system from the application, the IEC 61499 defines the concept of services the system provides to an application. A service is a functional capability of a resource that is made available to an application.

A service interface function block is an abstraction of the software component implementing the services. Figure 12-1 shows an example of a service interface function block REQUESTER that provides some service to an application (examples of possible services include: read the values of sensors, increase the memory used by a resource, shut down a resource, send a message, access remote database, etc.). The IEC 61499 standard pre-defines some names for input/output parameters of service interface function blocks, such as INIT for initialization, INITO for confirmation of the initialization, QI for input qualifier, and so on.

and adds a unique (for that device) port number at that location.

The CLIENT, located in Device 1, has an ID that is composed of two parts: IP address of the server's device and the same port number as used in the SERVER's ID.

Process interface: read inputs and write outputs

The process interface of a device is defined by service interface function blocks that can read values of sensors connected to the input ports and set values of actuators via the output ports. Figure 12-11 shows an example of using such service interface function blocks in a control application. In the Figure, inputs and outputs of the controller function block FEED_CTL are connected to the signals of controller ADAM 6650 via the corresponding service interface function block of type A6650. These function block type is supplied by the hardware vendor. The number of its data inputs and outputs corresponds to the number of input and output contacts in the controller hardware. The IO function block will scan input modules of the controller every time interval as defined by the input variable DT (50 ms in this example). In case if any change is detected from the previous scan, the event IND will be emitted

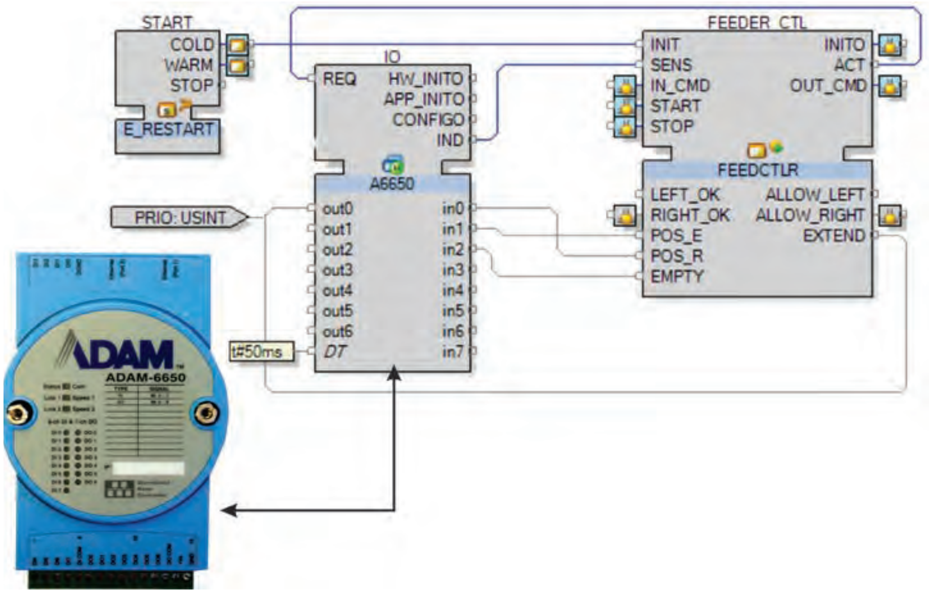


Figure 12-11 Discrete inputs and outputs of the ADAM 6650 interfaced by A6650 function block.

Symbolic Links

Another method of interfacing peripheral modules is applied to control devices with modular structure of input/output modules. It is based on the concept of "Symbolic links". The main benefits of this method as compared to the previously considered one are flexibility and more neat design avoiding "spaghetti" of connection arcs.

13

Simple application with decentralized control

This chapter applies function blocks to build a simple "component-based" system with decentralized control. The design of autonomous controllers and the idea of coordinating their activities without any external supervisor are illustrated. Moreover, it is demonstrated that this idea works for more complex hierarchical systems. The autonomous controllers are encapsulated in basic function blocks. They receive input signals from sensors and set values to actuators via service interface function blocks. When allocated to different devices, the controllers pass data to each other via communication function blocks.

Description of a controlled object

This chapter describes a small function block application that controls the mechatronic system Distribution Station shown in Figure 13-1.

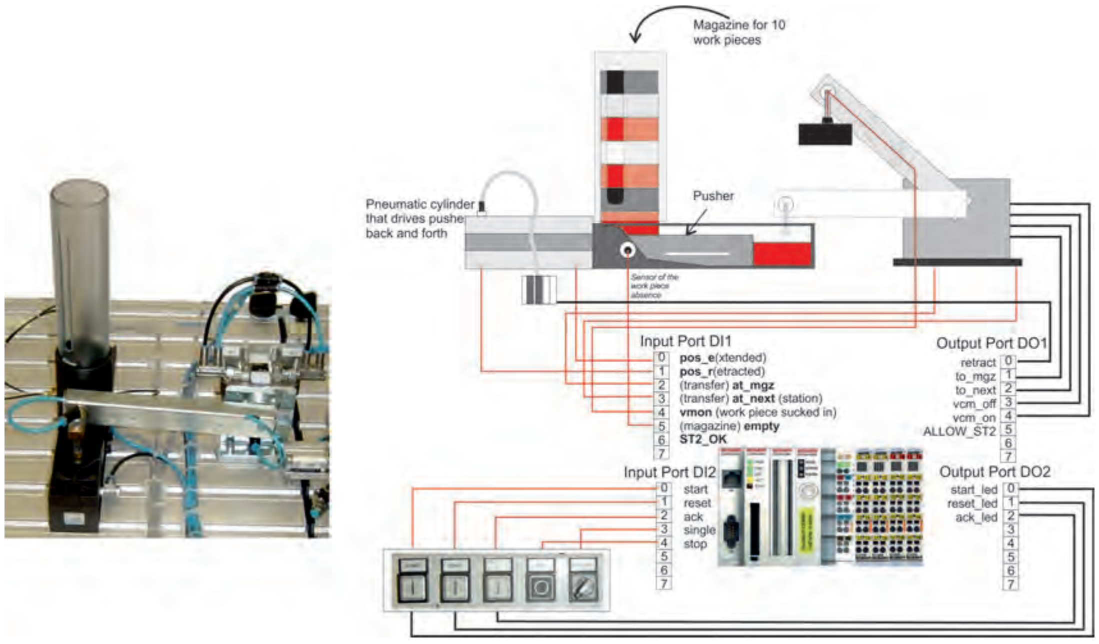


Figure 13-1 Controlled object: a simple mechatronic system.

14

Programming User Interface with Composite Automation Types

This chapter describes a specific mechanism for programming human-machine interface functions in NxtStudio, called Composite Automation Types (CATs). This concept enables object-oriented design, where both control logic and visualization functions can be encapsulated in the same software component.

HMI Architecture of NxtStudio

Process visualization and taking input from an operator are important parts of any industrial automation system. In the current systems, this part is usually developed using different software tools from those used for programming controllers. NxtControl attempted to simplify this by combining developments of both control and visualization into one architecture. Therefore, a system architecture in NxtStudio always includes one more device, an implicit HMI panel, as illustrated in Figure 14-1. Here, the Cylinders application is distributed across two usual control devices, but it also interacts with the "implicit" HMI panel that visualizes the process and receives input from operator.

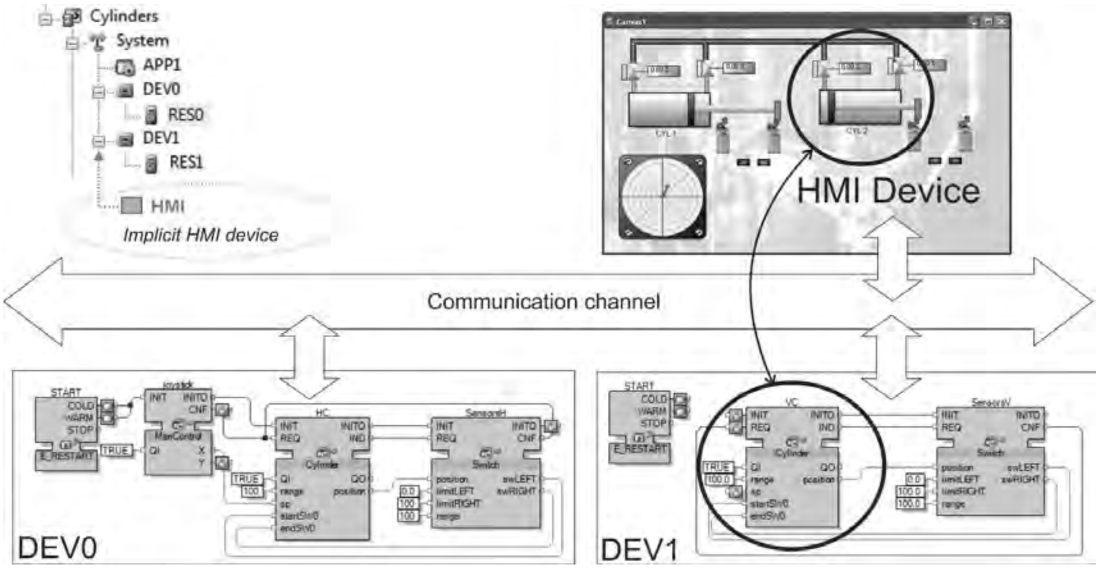


Figure 14-1 System architecture with an implicit HMI device.

15

Object-oriented design of applications

This chapter introduces an object-based design of applications with artifacts of the function block architecture. It aims to facilitate the development of more complex automation systems, including visualization, simulation models, and human-machine interface along with the usual control and communication.

Object-oriented Design

The concept of Object-Oriented Programming (OOP) in the general purpose software engineering aims at higher efficiency of software development on account of reusing software components. Model-driven engineering (MDE), in particular, the Unified Modelling Language (UML), complements the object-oriented design, offering graphical representation for better capturing design features.

The same goals are important in industrial automation, but there also specific requirements to design of complex automation systems, as follows:

- Integrate simulation and visualization of the controlled machine or process into the control and automation systems design process.
- Define a methodology to facilitate the transition from centralized to distributed and from simulated to physical systems.
- Incorporate facilities for machine and process diagnostics and fault recovery in the design pattern.

These and other requirements lead to object-oriented approach to the engineering of automation systems, which is similar but not identical to OOP. The goal of object-oriented automation system design will be to facilitate the development process making the structure of the software similar to the structure of the physical objects. The automation functions of physical objects are implemented by the corresponding hardware and software components.

The IEC 61499 architecture includes artifacts for implementing the most essential features of object-oriented design, such as encapsulation into function blocks, combined with support of higher level executable modes, such as state machines and block diagrams. However, the concept of object in automation need to be extended beyond only software part, to define a collection of data and knowledge elements that belong to, or relevant to, and describe the physical building blocks of automated manufacturing systems, which can range from simple sensors to complex machine tools or manufacturing cells.

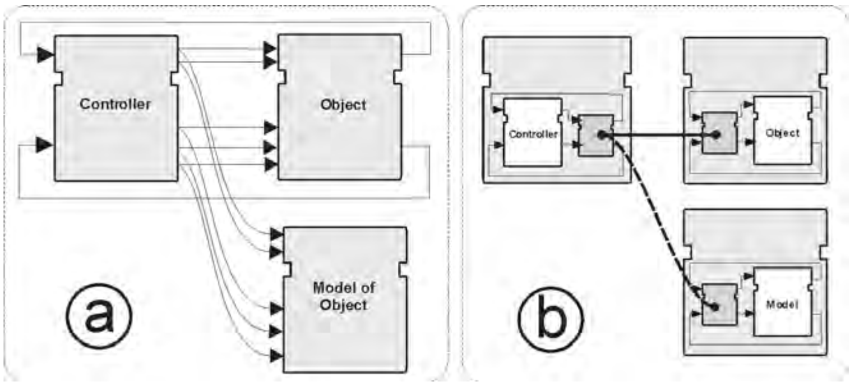


Figure 15-3 Reconnection of function blocks.

The alternatives are illustrated in Figure 15-4, using the Transfer object as an example. The usual connection between the Model and the Controller is shown on the left side of Figure 15-4, and the adapter connection is shown on the right. The right side of Figure 15-4 also shows the complexity of interconnections encapsulated in the adapter connection, which looks like a multi-wire cable. While encapsulation produces simpler looking components, the price paid for this simplicity is another layer of hierarchy between the actual controller function block and the composite function block having an adapter as a socket.

In cases where the blocks connected by adapter links have to be mapped to different devices, the adapter interfaces can be transferred over communication networks. For this purpose, specific communication function blocks can be developed. The idea underlying such blocks is illustrated in Figure 15-5, where a pair of customized communication blocks with adapter interfaces as input and output implement the seamless passage of adapters across the communication channel.

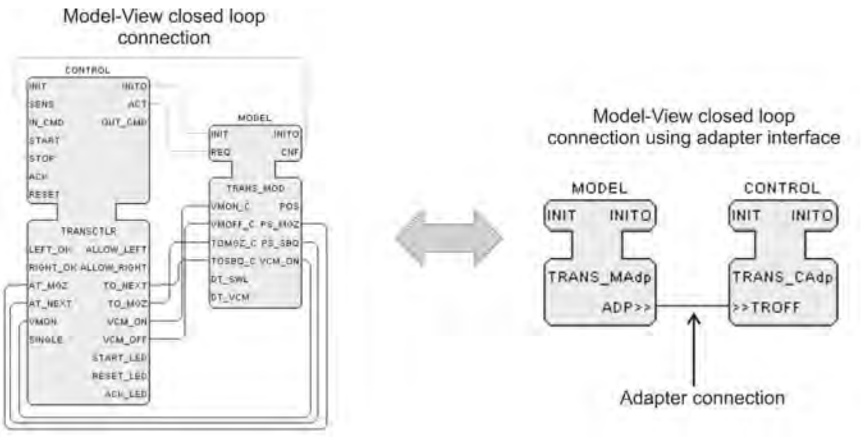


Figure 15-4 Simplifying connections between Controller and Model function blocks.

16

Case study: Modular Mechatronic System

In this chapter we consider in detail an example of modular software design with function blocks that will follow modularity and structure of the machinery under control. We will show how the function blocks architecture allows assembling a complete software solution for control, simulation and visualization from a library of predefined software components, and how different hardware and communication architectures can be explored. In particular, it will be shown how to check their impact on the functionality and performance of the system.

Machine description

The machine used in this chapter for the case study is a simplified version of a pick and place manipulator with remote control, as shown in Figure 16-1. The manipulator is composed of two pneumatic cylinders, and a joystick for remote control (for the sake of simplicity, we will not consider the gripper part at all). Each cylinder is equipped with two valves enabling forward and backward motion, and two end position sensors, indicating extended and retracted positions. There is great variety of valves and sensors that can be provided by suppliers, each of which may set specific requirements to control implementation. We will try to encapsulate these details into the corresponding software components, trying to demonstrate, how particulars can be masked if they are not essential at the top integration level. For the purposes of this study, we will assume that each constituent part of the manipulator can have its own control and communication capabilities. To take the situation to extremes, we can even assume that communication is wireless, which is no longer impossible in industrial systems.

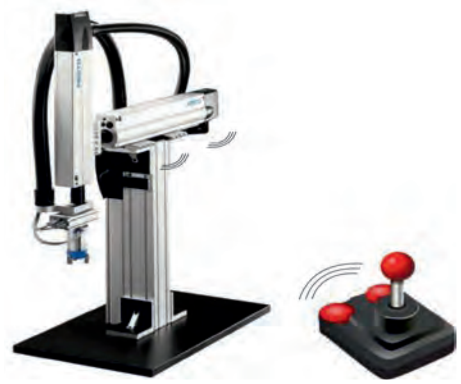


Figure 16-1 Pick and place manipulator with remote control.

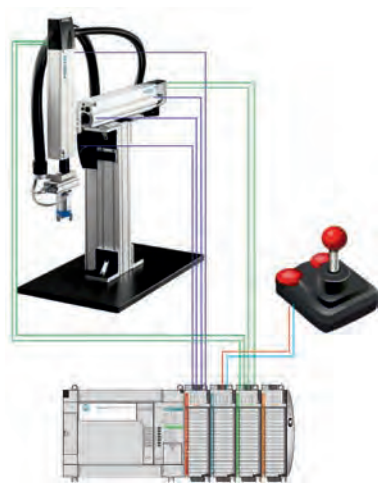
As it was said above, each cylinder will be considered as composed of two valves, two sensors, and a piston freely moving in both directions. The valves will be flow control valves whose throttle can be set in the range of 0-100%.

Desired functionality

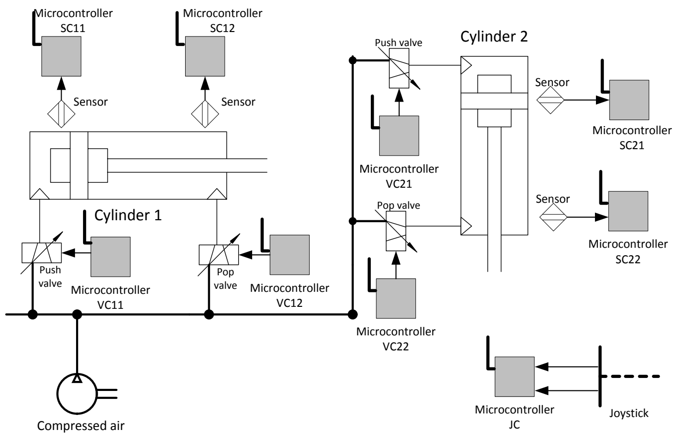
With the joystick the operator determines the direction of motion in X and Y coordinates and the desired force to move. The control system receives the two parameters and translates them into speed of motion. The motion should stop if end positions are reached, as indicated by sensors.

Range of hardware architectures

One can apply different control architectures in order to implement the desired functionality. The central controller architecture is on the one side of the spectrum. On the other extreme side, one can have a fully distributed solution where joystick and each sensor and valve is equipped with a microcontroller and wireless communication interface. These extremes are illustrated in Figure 16-2.



a)

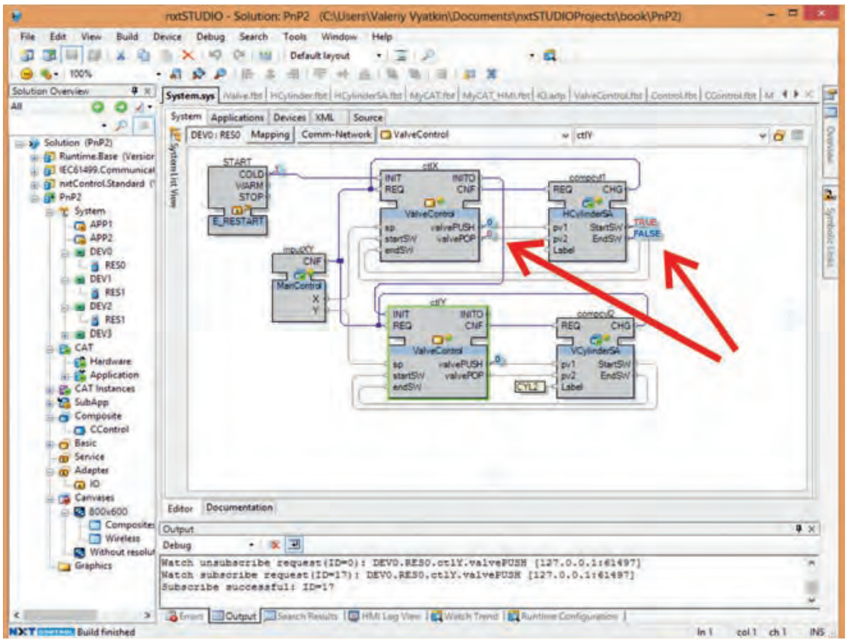


b)

Figure 16-2 a) Central control architecture; b) Fully distributed control architecture.

The central control architecture in Figure 16-2, a) implies that all sensors and actuators of the manipulator are connected to the interface modules of one control device, such as a PLC or a Programmable Automation Controller (PAC) encountered in previous sections.

The pneumatic connection diagram in Figure 16-2, b) illustrates another, distributed architecture. The diagram is also helpful to understand the nature of signals operated by the control system. Flow control of the valves is determined by an analog control signal in the range of


nxtSTUDIO - Solution: PnP2 (C:\Users\Valeriy Vyatkin\Documents\nxtSTUDIOProjects\book\PnP2)
System.sys
DEV0 : RES0
ValveControl
ctlY
TRUE
FALSE
CYL2

```
Watch unsubscribe request(ID=0): DEV0.RES0.ctlY.valvePUSH [127.0.0.1:61497]
Watch subscribe request(ID=17): DEV0.RES0.ctlY.valvePUSH [127.0.0.1:61497]
Subscribe successful: ID=17
```

Figure 16-28 Observing values of variables.

Summary

This chapter presented a comprehensive case study of designing automation system following the object-based concept. The presented solution illustrates the potential of Model-View-Control architecture and CATs to achieve the plug and play goal of software design. The chapter has also illustrated the problem of automation systems design in an abstract way that can adapt to the variety of available mechatronic components and communication possibilities.

Review questions for Chapter 16

1. How many different deployment scenarios exist for the application in Figure 16-6? What about the application in Figure 16-13?
2. Present arguments in favour and against the fully distributed architecture in Figure 16-2,b.
3. Why is it said that Figure 16-5 presents an open-loop solution, although there are feedback lines from plant to the controller? What would be a proper closed-loop control diagram?
4. Explain how the **iValve** CAT is configured in order to implement push and pop valves.

17

New business models

This chapter shows the potential of open software architecture for creating a knowledge economy in automation. Business models are presented for tool and service providers, device vendors, system integrators and end users.[1]

Structure of the automation market

The distributed automation architecture presented in this book will be successful in the industrial automation and control market if it substantially enhances the ability of each set of players to deliver added value by encapsulating, reusing, and deploying their specialized intellectual property (IP) at successively higher levels of integration, as illustrated in Figure 17-1.

The automation market exhibits the characteristics of the network economy described by Shapiro and Varian (1999). These characteristics include large network externalities (user economies of scale), where the value of a technology increases exponentially with the number of users and positive feedback, where successful application of the technology encourages additional users and vendors to enter the market.

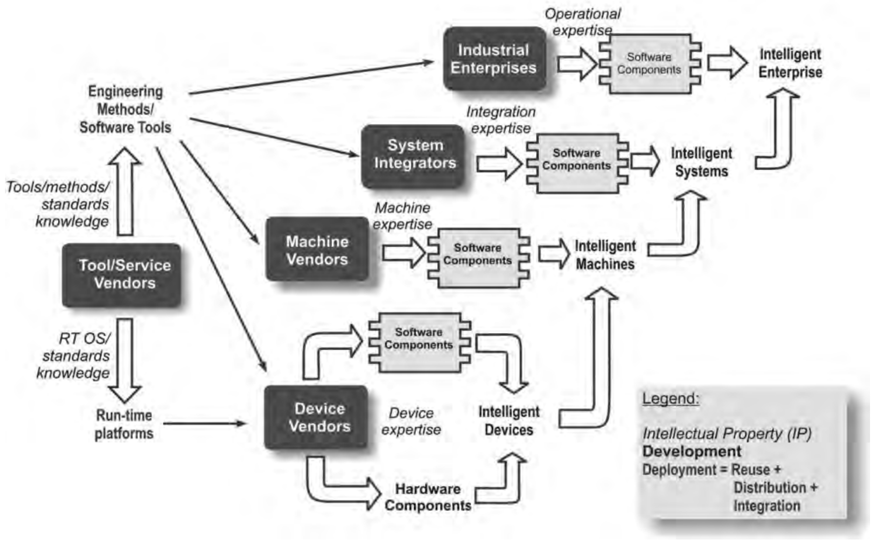


Figure 17-1 The value creation chain in industrial automation.[2]

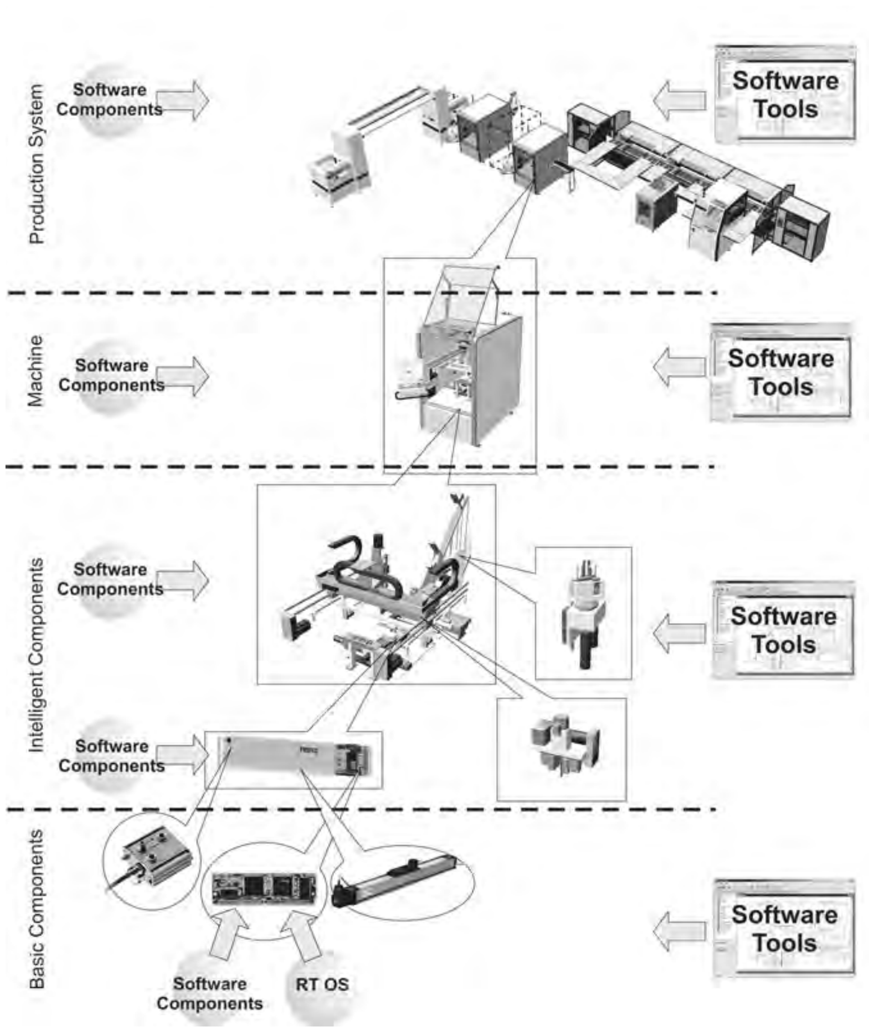


Figure 17-2 An example of a knowledge-based automation value creation chain in manufacturing (Auinger et al, 2005).

The vendor of these intelligent components may produce and maintain a range of similar devices. The differences between them would lie in their basic structures and characteristics, as well as in the model and performance of embedded control device within the set of supported networks and protocols, etc. Since the basic components comply with the future automation object standard, design and maintenance of the whole product range is very much simplified. It will always be possible to substitute one component by another with minimal re-design and re-programming effort, because the automation object concept provides an abstraction level that hides unnecessary detail.

A vendor of intelligent production machines will use a range of available intelligent components. With the help of their virtual representation, new machines will be designed and simulated.

18

More tools for function block design

This chapter presents several other software tools compliant with IEC 61499: ISaGRAF, FBDK, and 4DIAC IDE.

ISaGRAF

Overview

ISaGRAF is a product of Rockwell Automation, Canada. Starting with version 5, released in 2005, this IEC 61131-3 product has been enhanced with support for IEC 61499. It is now possible to develop distributed control applications in IEC 61499 together with application parts in IEC 61131-3.

ISaGRAF is the first fully fledged automation product supporting the complete design chain with IEC 61499. Based on the well-established IEC 61131 standard, this solution shows how the new standard can be incorporated. As shown in Figure 18-1, ISaGRAF consists of Workbench and portable firmware.

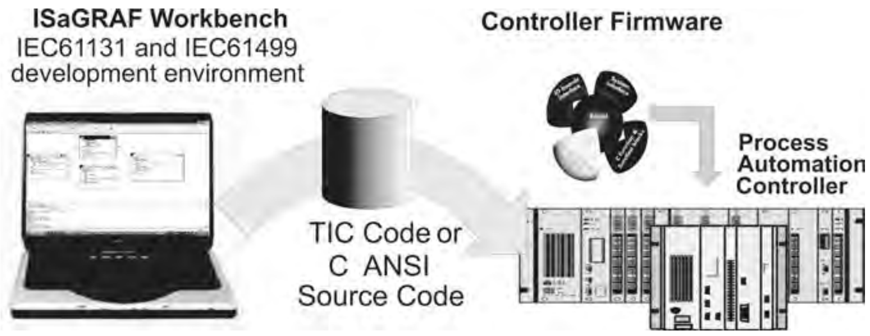

ISaGRAF Workbench
IEC61131 and IEC61499 development environment
Controller Firmware
Process Automation Controller
TIC Code or C ANSI Source Code

Figure 18-1 Structure of ISaGRAF: Workbench and Firmware for IEC 61499 and IEC 61131-3 system development.

At the same time, not all concepts of IEC 61499 have been implemented. The communication between the application parts is achieved through network variables instead of service interface FBs. The appearance of FBs in ISaGRAF follows IEC 61499, but their internals look a bit different. The Execution Control Chart of a basic FB, for example, is defined in a way similar to the IEC 61131-3 Sequential Function Chart language. The event-driven IEC 61499 FBs are executed on top of a cyclic-scanned and time-triggered IEC 61131-3 runtime system.

ISaGRAF allows the use of the traditional PLC programming languages of the IEC 61131-3 standard in function blocks; for example, it allows ladder logic, which is shown in Figure 18-2.

Reference example

The ‘LED Chaser’ reference example provided with the ISaGRAF demo kit (Figure 18-3) will be used to illustrate differences between this implementation and other tools, such as NxtStudio and FBDK. The demo kit consists of 3 identical microprocessor devices with some input/output ports connected with 2 buttons, 2 switches and four light emitting diodes (LED).

The ‘LedChaser’ application works as follows. On start up, the light begins ‘running’ from the left- most LED (LED3) to the right. The direction changes when the light reaches the right-most LED, and so on. Pressing the B1 button on either device increases the speed of running, and pressing B2 decreases it. At any given time there is only one lit lamp in all three devices. An FB network controller for this system in the ISaGRAF version of IEC 61499 function blocks is shown in Figure 18-4. The control is decentralised; pressing a button in any device will immediately impact on the speed of the run, even when the light is already ‘running’ in another device.

If you press a button on device 3 while the light is in device 1, for example, the event will go to the PeriodManagement block, and then a new period will be transmitted to LedSequence1. As a result, the speed (of the light movement in device 1) will change instantaneously.

The program includes six instances of the ‘ButtonManagement‘ FB, one per button. This FB detects a released button, and generates an event signal, which is sent to the ‘PeriodCalc’ FB, and that generates the value of time interval between the illuminations of two neighbouring LEDs. This value is passed to the FBs responsible for sending signals to the lamps within one device. These are the three instances of the ‘LedSequence’FB type. This FB has two input events: ‘Inc’ and ‘Dec’. When Inc is received, the FB sequentially displays all 5 possible patterns, in binary coding: ‘0000’, ‘1000’, ‘0100’, ‘0010’, ‘0001’ and ‘0000’, delaying the duration between each change of the output pattern.

The ’LedSequence’ is a basic function block, whose logic is defined by means of an Execution Control Chart (ECC). In ISaGRAF, the ECC is implemented using a Sequential Function Chart (SFC) programming language. Apparently, ISaGRAF developers wanted to re-use the existing SFC editor, on account of control engineers’ familiarity with this language.

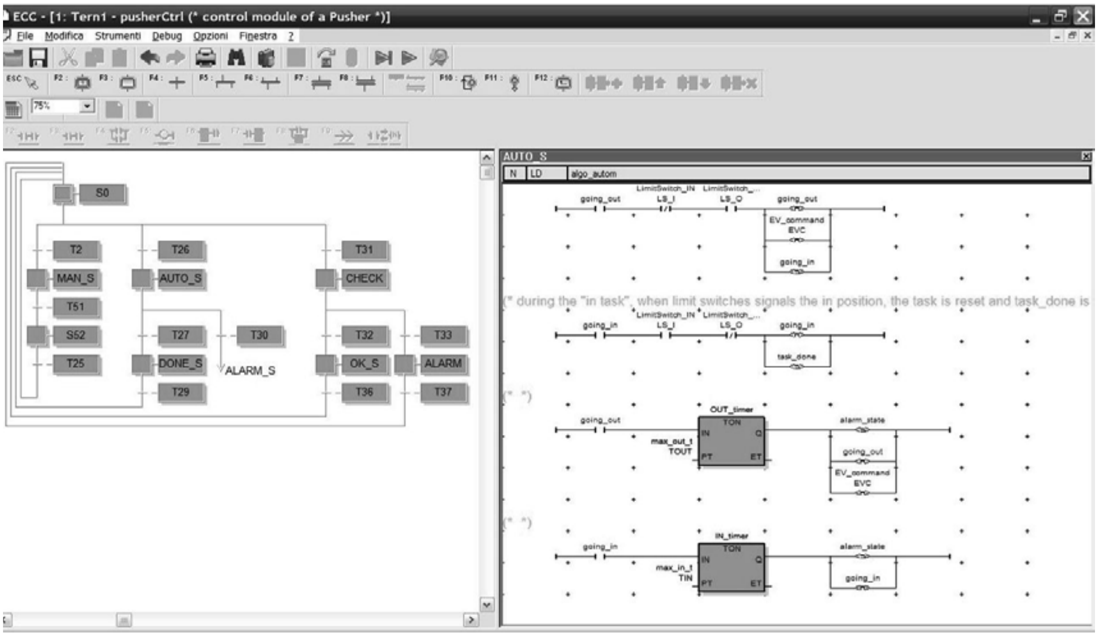


Figure 18-2 ISaGRAF workbench enables the use of Ladder Logic to program algorithms in basic function blocks.

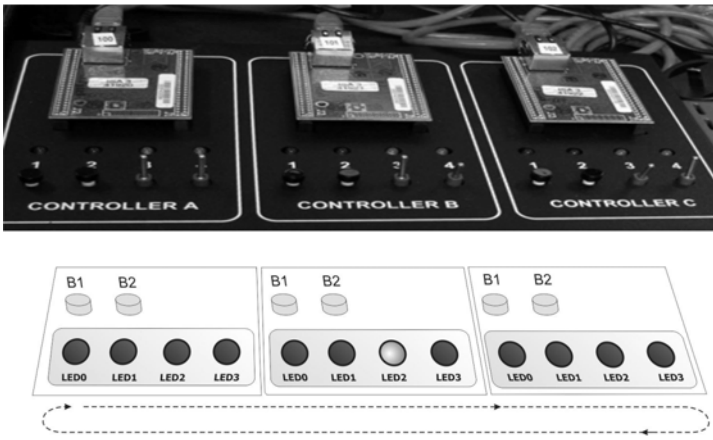


Figure 18-3 LED Chaser system.

Despite the similar look, the ECC semantics is different from pure SFC semantics in IEC 61131-3. In the “pure” SFC, no more than one transition per scan can occur. After a transition, SFC is “put on hold”.

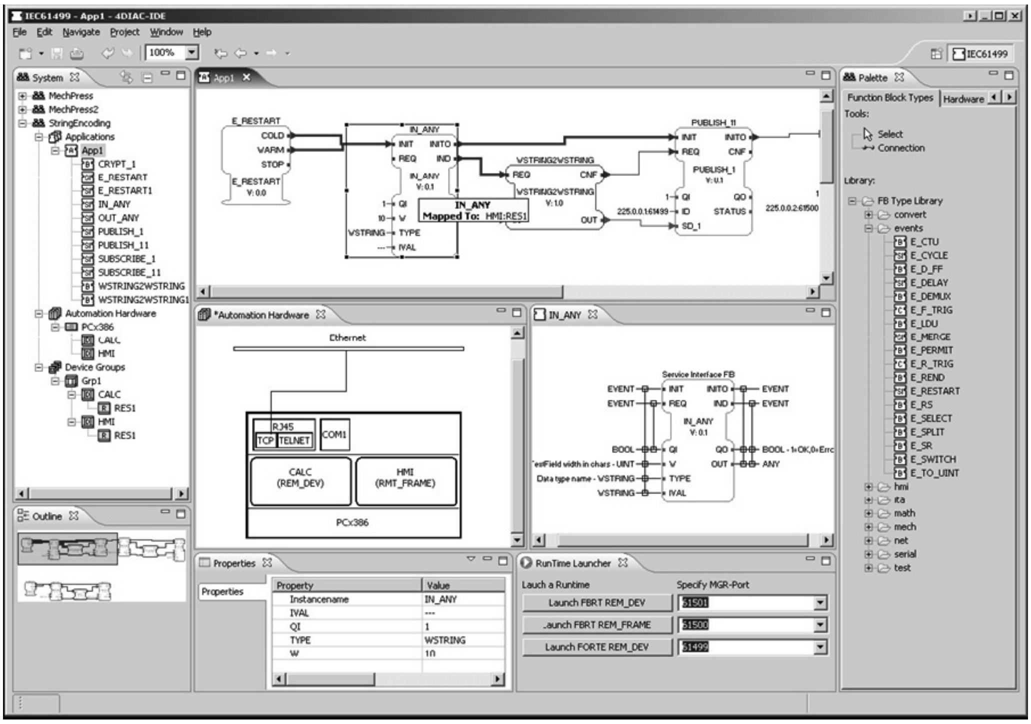
4DIAC IDE and Forte

The 4DIAC consortium was formed in 2007 by several academic and industrial organizations with the goal of delivering open source design tools and run-time platforms for IEC 61499 (4DIAC, 2009). The consortium clustered around the Odo Struger lab at Vienna Technical University and the research company PROFACTOR (Austria).

4DIAC has developed an open source integrated development environment (IDE) under Eclipse (see Figure 18-18). It uses the plug-in mechanism of Eclipse to enable tool flexibility and extensibility. The tool can generate code for and execute function block applications on two run-time platforms: Forte and FBRT.

Forte is another deliverable of the 4DIAC consortium. It is a runtime environment written in C++. This admittedly makes it more efficient than FBRT (although some tests on a PC, surprisingly, show the opposite result). Forte has been ported to a great range of hardware targets. It has proven to have real-time capabilities, as elaborated in Zoitl (2009). Figure 18-19 shows an example of such an application: a robot with 6 degrees of freedom and distributed function block control was created by PROFACTOR (Austria). The control system is implemented as three networking embedded controllers, each responsible for two axes.

Forte is used in the commercial tool-chain of NxtControl. In addition, 4DIAC-IDE and Forte represent a good alternative to FBDK/FBRT, especially for academic and training purposes.



Property	Value
Instancename	IN_ANY
IVAL	---
QI	1
TYPE	WSTRING
W	10

Lauch a Runtime	Specify MGR-Port
Launch FBRT REM_DEV	61501
Launch FBRT REM_FRAME	61500
Launch FORTE REM_DEV	61499

Figure 18-18 Screenshot of 4DIAC-IDE.

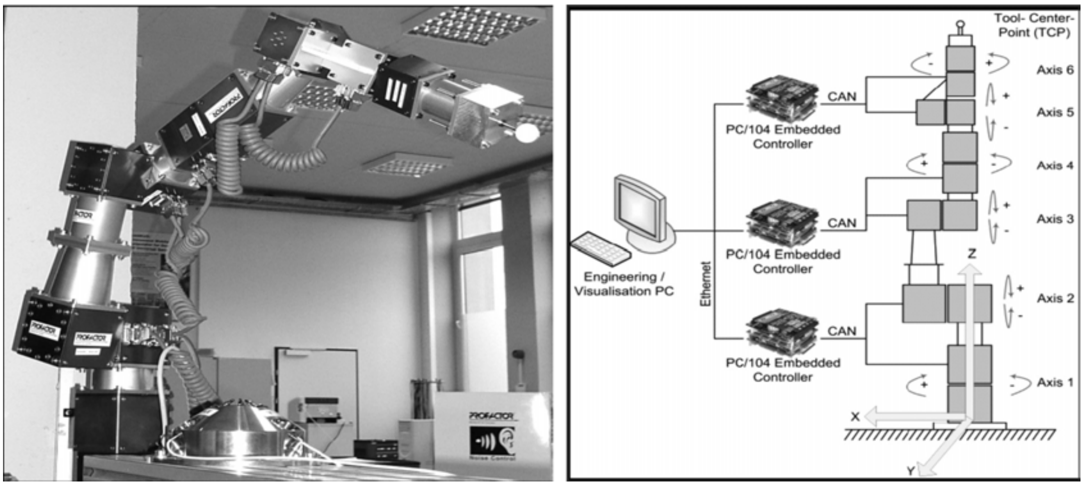


Figure 18-19 Robot with 6 degrees of freedom and distributed function block control (PROFACTOR, Austria).

19

Conclusion

Our journey through IEC 61499 is finally over! We have tried to touch upon all important artifacts of the IEC 61499 reference architecture. But the reader should bear in mind that this book is just an introduction to the distributed automation world of IEC 61499. For that reason, we could not cover many important issues.

First, the book is definitely not a reference for the IEC 61499 standard. It does not describe all the standard's features in all its detail. The goal of the book was to help the reader understand the main concepts, get a hands-on feel for the standard and appreciate its potential. Accordingly, the material covered in this book is not sufficient for developing IEC 61499–compliant devices and tools and proving their compliance—that requires more work with the text of the standard.

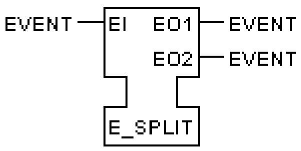
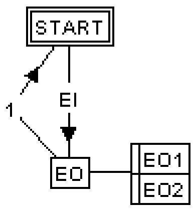
Secondly, we completely omitted discussion of such important issues as performance and networking. These are very important for automation systems. There are several sources where the reader can gain more knowledge on the subject. The book by Zoitl (2009) presents several interesting approaches to the real-time execution of IEC 61499. Another source is the chapter in Vyatkin, De Sousa and Zoitl (2010) that fills the gap in the details related to communication in IEC 61499 implementations.

The author hopes that this text will help promote IEC 61499. The author firmly believes that this standard can improve the software engineering practices in industrial automation and embedded control systems development!

Appendix A: Event function blocks

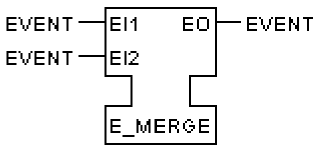
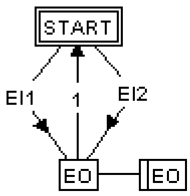
The following table is taken from Table A1 of the IEC 61499 standard.

Split an event

Interface	ECC/Algorithms/Service sequences
	

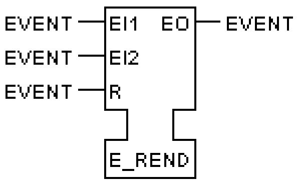
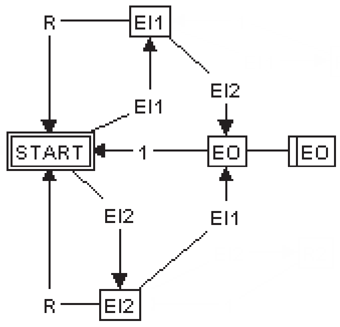
The occurrence of an event at EI causes the occurrence of events at EO1, EO2,..., EOn (n=2 in the above example).

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).

Rendezvous of two events

 EVENT EI1 EO EVENT EVENT EI2 EVENT R E_REND	 R EI1 EI2 EI1 START 1 EO EO EI2 EI1 R EI2

Appendix B: IEC 61499 Compliance Profile for Feasibility Demonstrations

The compliance profile (ITA, 2002) has been agreed on by several organisations participating in research around the IEC 61499 feasibility demonstration. In this appendix, we provide parts of that profile that are relevant to the device management.

Part 4. CONFIGURABILITY AGREEMENT

Parties to this agreement who are providers of software tools, and parties to this agreement who are providers of devices, mutually agree to the following conditions for the demonstration of configurability.

4.1 Software tools

Software tools shall be capable of utilizing the management capabilities of devices that are configured according to the functional equivalent of Figure 4.3-1.

EXAMPLE – The hostport value to access the management services in a remote device named m51568 would typically be represented as "m51568:61499". If a Domain Name System (DNS) server were not available, but the remote device's Internet Protocol (IP) address were known to be 161.153.19.227, the corresponding hostport value would be "161.153.19.227:61499".

4.2 Device management services

The device management services to be implemented are provided by the functional equivalent of an instance of the DEV_MGR type shown in Figure 4.2-1.

The types and semantics of the inputs and outputs of the DEV_MGR type are identical to the correspondingly named inputs and outputs of the MANAGER type defined in subclause 3.3.2 of IEC 61499-1, with the following differences:

The DST input designates the destination of the RQST input as follows:

- A value of "" (the empty string) designates the device;
- A value containing an IEC 61131-3 identifier designates a resource within the device;
- A value containing a sequence of IEC 61131-3 identifiers separate by periods (the "." character) indicates a resource in a containment hierarchy of resources, with the leftmost identifier corresponding to the outermost resource and the rightmost identifier corresponding to the innermost resource.

EXAMPLE 1 - A DST value of "RES1" indicates that the RQST input is destined for a resource named RES1 contained in the managed device.

EXAMPLE 2 - A DST value of "MOTOR1.WINDING2" indicates that the RQST input is

Bibliography

AO Automation Objects for industrial-process measurement and control systems—International Electro- technical Commission, SB3/ Technical Committee 65, Working draft, available at *www.holobloc.com/stds/iec/sb3/ao/aowd1.zip*, 2002.

Auinger, F., R. Brennan, J. Christensen, J. L. M. Lastra, and V. Vyatkin. "Requirements and Solutions to Software Encapsulation and Engineering in Next Generation Manufacturing Systems: O^{3}neida Approach," International Journal of Computer Integrated Manufacturing, 2005.

Black, G., and V. Vyatkin. "Intelligent Component–based Automation of Baggage Handling Systems with IEC 61499," IEEE Transactions on Automation Science and Engineering, 2010, 7(2), DOI: 10.1109/TASE.2008.2007216, pp. 337–351.

Bonfe, M., and C. Fantuzzi. "Design and verification of mechatronic object-oriented models for industrial control systems," IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03), Proceedings, Vol. II, pp. 253–260, Lisbon, 2003.

Brennan, R. W., X. Xu Y. Zhang, and D. H. Norrie. "A reconfigurable concurrent function block model and its implementation in real-time JAVA," Integrated computer Aided Engineering, 2002.

Čengić, G., O. Ljungkrantz, and K. Åkesson. "Formal Modelling of Function Block Applications Running in IEC 61499 Execution Runtime" in Proc. of 11th IEEE Intl Conf. ETFA 2006, Prague.

Christensen, J. H. Remark on the execution semantic in the new version of FBDK, 2006, March,—private communication.

———. "IEC 61499 Architecture, Engineering, Methodologies and Software Tools," 5th IFIP International Conference BASYS'02, Proceedings, Cancun, Mexico, 2002.

———. "HMS/FB Architecture and Its Implementation," in S. M. Deen (ed.), Advances in the Holonic Approach to Agent-Based Manufacturing, Springer-Verlag, 2003.

———. "Design patterns for systems engineering with IEC 61499," Verteilte Automatisierung, Proceedings, Magdeburg, Otto-von-Guericke-Universitaet, 2000.

———. "Holonic Manufacturing Systems: Initial Architecture and Standards Directions," First European Conference on Holonic Manufacturing Systems, Proceedings, Hanover, Germany, 1994.

Chouinard, J., D. Lavallée, J.-F. Laliberté, N. Landreaud, K. Thramboulidis, P. Bettez-Poirier, F. Desy, F.Darveau, N. Gendron, and C.-D. Trang. "An IEC 61499 configuration with 70 controllers; challenges, benefits and a discussion on technical decisions," IEEE Conference on Emerging Technologies and Factory Automation, Patras, Greece, 2007.

CORFU Engineering Support System, Online: *seg.ee.upatras.gr/corfu/dev/index.htm*, access date: February 2010.

Douglass, B. P. "Real-time UML: developing efficient objects for embedded systems." 2 ed., 1. print. Reading Mass. [u.a.]: Addison-Wesley, 1999.

Doukas, G., and K. Thramboulidis. "A Real-Time Linux Execution Environment for Function-Block Based Distributed Control Applications," 3rd IEEE International Conference on Industrial Informatics (INDIN´05), Perth, Australia, August 2005.

Dubinin, V., V. Vyatkin, and T. Pfeiffer. "Engineering of Validatable Automation Systems using UML- FB," Higher Education Institute Letters, No.2, pp.136–146, ISSN: 1728-628X, 2004.

Dubinin, V., V. Vyatkin, and T. Pfeiffer. "Engineering of Validatable Automation Systems Based on an Extension of UML Combined with Function Blocks of IEC 61499," Proceedings of IEEE International Conference on Robotics and Automation, Barcelona, pp. 4007–4012, April 2005.

Ferrarini, L., M. Romanò, and C. Veber. "Automatic Generation of AWL Code from IEC 61499 Applications," in 4th IEEE Conference on Industrial Informatics (INDIN'06), Proceedings, Singapore, August 2006.

Ferrarini, L., and C. Veber. "Implementation Approaches for the Execution Model of IEC 61499 Applications," IEEE Conference INDIN'04, Berlin, 2004.

———. "IEC 61499—Uno standard per sistemi distribuiti di automazione industriale," Pitagora Editrice Bologna, 2004.

Ferrarini, L., C. Veber, and K. Lorentz. A case study for modelling and design of distributed automation systems. In proceedings of IEEE/ASME Int. Conference on Advanced Intelligent Mechatronics (AIM), 2003.

4DIAC Consortium. "Framework for Distributed Automation and Control (4DIAC)," Online Available: *www.fordiac.org*, June 2009.

4DIAC-RTE (FORTE): IEC 61499 Compliant Runtime Environment, Online: *www.fordiac.org/8.0.html*, June 2009.

Hagge, N., and B. Wagner. "A New Function Block Modelling Language Based on Petri Nets for Automatic Code Generation," IEEE Trans. on Industrial Informatics, Vol. 1, No 4, pp. 226–237, Nov. 2005.

Hanisch, H.-M., and V. Vyatkin. "Achieving Reconfigurability of Automation Systems by Using the New International Standard IEC 61499: A Developer's View," The Industrial Information Technology Handbook, Chapter 66, CRC Press, 2004.

Hirsch, M., and H.-M. Hanisch. "Systemspezifikation mit SysML für eine Fertigungstechnische Laboranlage," Fachtagung zum Entwurf komplexer Automatisierungssysteme (EKA 08) Proceedings, pp. 23–34, Magdeburg, Germany, 2008.

HMS Holonic Manufacturing Systems Consortium, "Holonic manufacturing systems overview," Online: *hms.ifw.unihannover.de*, 2002.

HOLOBLOC™.com Function Block-Based, Holonic Systems Technology, *www.holobloc.com*, Access Date: March 2005.

Huber, B., R. Obermaisser, and P. Peti. "MDA-based development in the DECOS integrated architecture—modelling the hardware platform," Proceedings of 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06), pp. 10–15, April 2006.

Iacocca Institute, "21st Century Manufacturing Enterprise Strategy: an Industry-Led View," Iacocca Institute, Bethlehem, Pennsylvania, Tech. Rep., 1991.IDA Interface for Distributed Automation Group *www.ida-group.org*, 2000.

IEC 61131-3 Standard, Programmable Controllers, International Electrotechnical Commission, Geneva, Switzerland, 1993.

IEC 61804—Function Blocks for Process Control, Part 1—General Requirement; Part 2—Specification, Publicly Available Specification, International Electrotechnical Commission, Working Group 6, Geneva, 2002.

IEC 61499, Part 1 "Architecture" Function Blocks for Industrial Process Measurement and Control Systems, Standard, International Electrotechnical Commission, Geneva, 2005.

IEC 61499, Part 2 "Software tools requirements: Function Blocks for Industrial Process Measurement and Control Systems," Standard, International Electrotechnical Commission, Geneva, 2005.IEC 61499, Part 3 "Application guidelines": Function Blocks for Industrial Process Measurement and Control Systems, Technical report, Geneva, 2005.

IEC 61499, Part 4—Rules for Compliance Profiles, Function Blocks for Industrial Process Measurement and Control Systems, Standard, International Electrotechnical Commission, Geneva, 2005.

IEC 61499 Compliance Profile for CIP based communication function blocks, Online Available: *www- ist.massey.ac.nz/functionblocks/CIPCompliance.htm*, Access date: March 2010.

IEC 61499 Compliance Profile for Feasibility Demonstrations, Online Available: *www.holobloc.com/ doc/ita/index.htm*, June 2009.

ICS Triplex ISaGRAF Inc., "ISaGRAF User's Guide," Online Available: *www.isagraf.com*, Jun. 2009. ISO TR 8509-(1987), Information processing systems—Open Systems Interconnection—Service conventions

ITA IEC 61499 Compliance Profile for Feasibility Demonstrations, Online: *www.holobloc.org*, 2002.

Jammes F., and H. Smit. "Service-Oriented Architectures for Devices-the SIRENA View," IEEE Transactions on Industrial Informatics, 1(1), 2005.

Lastra J. L. M., A. Lobov, L. Godinho, and A. Nunes. "Function Blocks for Industrial-Process Measurement and Control Systems: IEC 61499 Introduction and Run-time Platforms," Institute of Production Engineering, Tampere University of Technology, Tampere, 2004.

Lewis, R. "Modelling Distributed Control Systems using IEC 61499", Institution of Electrical Engineers; London, 2001.

Marik, V., V. Vyatkin, and A. W. Colombo (Eds). Lecture Notes in Artificial Intelligence, vol. 4659. Springer-Verlag, Berlin, Heidelberg, 1-14. DOI: *dx.doi.org.ezproxy.auckland.ac.nz/10.1007/ 978-3-540-74481-8_1*.

Noble, D. "Forces of Production: A social history of industrial automation," Alfred A. Knopf, New York, 1984.

nxtControl GmbH, "nxtControl—next generation software for next generation customers," Online Available: *www.nxtcontrol.com/*, June 2009.

OMG Object Management Group. "Model Driven Architecture," Online Available: *www.omg.org/mda/faq_mda.htm*, 2009.

O^3neida. IEC 61499 Compliance Profile—Execution Models of IEC 61499 Function Block Applications, draft in progress, Online Available: *www.oooneida.org/standards_development_Compliance_Profile.html*, March 2009.

O^3neida. Open Object-oriented Knowledge Economy in Intelligent Industrial Automation, IMS Community of Common Interest, official web site: *www.oooneida.info*, Access date: January 2006.

O^3neida. "Compliance Profile," Online Available: *www.oooneida.org/ standards_development_ Compliance_Profile.html*, June 2009.

Panjaitan, S., and G. Frey. "Combination of UML Modelling and the IEC 61499 Function Block Concept for the Development of Distributed Automation Systems," IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06), Proceedings, pp.766–773, Prague, Czech Republic.

———. "Development Process for Distributed Automation Systems Combining UML and IEC 61499," International Journal of Manufacturing Research, Vol.2, No.1, pp.1–20, Inderscience Publishers, 2007.

———. "Functional Design for IEC 61499 Distributed Control Systems using UML Activity Diagrams," International Conference on Instrumentation, Communications and Information Technology (ICICI'05) Proceedings pp. 64–70, Bandung, Indonesia 2005.

Riedl, M., C. Diedrich, and F. Naumann. "SFC in IEC 61499," 13th IEEE Conference on Emerging Technologies and Factory Automation, Prague, pp.662–667, 20–22 Sept. 2006.

Rumpl R., C. Dutzler, A. Zoitl, and F. Auinger. (2002), Platforms for Scaleable Flexible Automation considering concepts of IEC 61499, In 5th IEEE/IFP International Conference on Information Technology for Balanced Automation Systems in Manufacturing and Services, Cancun, Mexico.

Schoop, R. and H. D. Ferling. (1997), Function Blocks for Distributed Control Systems, DCCS'97, Proceedings, pp.145–150.

Shapiro, H., and C. Varian (1999). Information Rules: A Strategic Guide to the Network Economy. Harvard Business School Press, Boston, ISBN: 0-87584-863-X.

SIEMENS: *www.siemens-industry.co.uk/news/pressrel.asp?ID=109*, access date April 2004. Sipper, D., and R. Bulfin. Production: Planning, Control and Integration, McGraw-Hill, 1997.

Shaw, G., P. Roop, and Z. Salcic. A hierarchical and concurrent approach for IEC 61499 function blocks, IEEE International Conference on Emerging Techonologies and Factory Automation, 2009.

Stanica M., and H. Guéguen. "Timed Automata Model of IEC 61499 Basic Function Blocks Semantic," ECRTS'03 Euromicro European Conference on Real-Time Systems, Porto, Portugal, 2003.

Strasser, T., C. Sünder, M. Rooker, O. Hummer-Koppendorfer, A. Zoitl, and I. Müller. "Enhanced IEC 61499 system model for evolution of control applications in distributed industrial-process measurement and control systems," in European Control Conference, July 2007.

Strasser, T., M. Rooker, G. Ebenhofer, A. Zoitl, C. Sünder, A. Valentini, and A. Martel. "Structuring of large scale distributed control programs with IEC 61499 subapplications and a hierarchical plant structure model," in IEEE International Conference on Emerging Technologies and Factory Automation, (ETFA'08), Sept. 2008.

Sünder, C., A. Zoitl, J. Christensen, M. Colla, and T. Strasser. "Execution models for the IEC 61499 elements composite function block and subapplication," in 5th IEEE International Conference on Industrial Informatics, Vol. 2, pp. 1169–1175, June 2007.

Sünder, C., A. Zoitl, J. Christensen, V. Vyatkin, R. Brennan, A. Valentini, L. Ferrarini, T. Strasser, J. Martinez-Lastra, and F. Auinger. "Usability and interoperability of IEC 61499 based distributed automation systems," in IEEE International Conference on Industrial Informatics, pp.31–37, Aug. 2006.

Szyperski, C. "Component Software: Beyond Object-Oriented Programming," 2nd ed, New York: ACM Press, 2002.

Tait, P. "Commercialising Distributed Control," 3rd IEEE International Conference on Industrial Informatics, Perth, Western Australia, 2005.

Tata, P., and V. Vyatkin. "Proposing a novel IEC 61499 Runtime Framework implementing the Cyclic Execution Semantics," INDIN '09, Cardiff, Wales, June 2009.

TÜV Rheinland. "ISaGRAF 5.1 assessment according to IEC 61499," Online Available: *www.isagraf.com/pages/documentation/TUV/ISaGRAF_5-1_1499-TUVreport.pdf*, June 2009.

Testbed (2006): Mechatronic testbed with IEC614999 control, *at.iw.uni-halle.de/~testbeds/*, Access Date: Jan. 2006.

Thramboulidis, K. and G. Doukas. "IEC 61499 execution model semantics," in Innovative Algorithms and Techniques in Automation, Industrial Electronics and Telecommunications, 2007, pp. 223–228.

Thramboulidis, K. "Model Integrated Mechatronics-Toward a New Paradigm in the Development of

Manufacturing Systems." IEEE Transactions on Industrial Informatics 1(1), pp. 54–61, 2005.

———. "IEC 61499 in factory automation," in Proceedings of the International Conference on Industrial Electronics, Technology and Automation. Institute of Electrical and Electronics Engineers Inc., Piscataway, NJ, USA, 2005.

Thramboulidis, K. and A. Zoupas. "Real-Time JAVA in Control and Automation: A Model Driven Development Approach," 10th IEEE Intl. Conf. on Emerging Technologies and Factory Automation, Catania, Italy, Sept. 2005.

Tranoris, C., and K. Thramboulidis. "Integrating UML and the Function Block Concept for the Development of Distributed Applications," IEEE Intl. Conference on Emerging Technologies and Factory Automation (ETFA'03), Proceedings, vol. II, pp.87—94, Lisbon, 2003.

TÜV Rheinland, "ISaGRAF 5.1 assessment according to IEC 61499," Online Available: *www.isagraf.com/pages/documentation/TUV/ISaGRAF_5-1_1499-TUVreport.pdf*, June 2009.

Upton, D. "A flexible structure for computer-controlled manufacturing systems," Manufacturing Review, 5(1), pp. 58–74, 1992.

Vyatkin, V. "Software Engineering in Factory and Energy Automation: State of the Art Review", IEEE Transactions on Industrial Informatics, 9(3), pp. 1234 - 1249, 2013

V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State of the Art Review", IEEE Transactions on Industrial Informatics, 7(4), 2011, pp. 768-781

Vyatkin, V., M. De Sousa, and A. Zoitl. 'Communication aspects of IEC 61499 architecture,' Industrial Electronics Handbook, Chapter 54, USA: Taylor & Francis, 2010.

Vyatkin, V., H. M. Hanisch, C. Pang, and J. Yang. "Application of closed-loop modelling in integrated component design and validation of manufacturing automation," IEEE Transactions on Systems, Machine and Cybernetics—C, Vol. 39, pp. 17–28, 2008.

V. Vyatkin, Salcic, Z., Roop, P., and J. Fitzgerald, "Information Infrastructure of Intelligent Machines based on IEC 61499 Architecture," IEEE Industrial Electronics Magazine, 1(4), 2007.

Vyatkin, V. and V. Dubinin. "Sequential axiomatic model for execution of basic function blocks in IEC61499," in 5th IEEE International Conference on Industrial Informatics (INDIN'07), Jul. 2007.

Vyatkin, V., V. Dubinin, L. Ferrarini, and C. Veber. "Alternatives for execution semantics of IEC61499,"in 5th IEEE Intl. Conference on Industrial Informatics (INDIN'07), pp. 1151–1156, 2007.

Vyatkin V., H. M. Hanisch. "Verification of Distributed Control Systems in Intelligent Manufacturing," Journal of Intelligent Manufacturing, Vol.14, N.1, pp.123–136, 2003.

Vyatkin V., and J. L Martinez Lastra. "Architectural Foundations for Reconfigurable Manufacturing Systems," 3rd International Symposium on Open Control Systems SoftSympo'03, Helsinki, 2003.

Vyatkin V., J. Christensen, and J. L. Martinez Lastra. "O^{3}neida: An Open, Object-Oriented Knowledge Economy for Intelligent Distributed Automation," IEEE Transactions on Industrial Informatics, Vol. 1, 2005.

Vyatkin, V. and J. Chouinard. "On comparisons of the ISaGRAF implementation of IEC 61499 with FBDK and other implementations," in 6th IEEE International Conference on Industrial Informatics, pp. 289–294, Jul. 2008.

Vyatkin V. "The Potential Impact of the IEC 61499 Standard on the Progress of Distributed Intelligent Automation," International Journal of Manufacturing Technology and Management, 2005.

———. "Intelligent Mechatronic Components: Control System Engineering using an Open Distributed Architecture," IEEE Conference on Emerging Technologies and Factory Automation (ETFA'03), Proceedings, Vol. II, pp.277–284, Lisbon, 2003.

———. "Modelling and execution of reactive function block systems with Condition/Event nets," 4th IEEE Conference on Industrial Informatics (INDIN '06), Proceedings, Singapore, 2006.

———. "The IEC 61499 standard and its semantics," Industrial Electronics Magazine, 3(4), 2009.

Vyatkin, V., H. M. Hanisch, S. Karras, T. Pfeiffer, and V. Dubinin. "Rapid Engineering and Re-Configuration of Automation Objects Using Formal Verification," Int. J. Manufacturing Research, vol. 1, pp.382–404, 2006.

Vyatkin, V., H. M. Hanisch, P. Starke, and S. Roch. “Formalisms for verification of discrete control applications on example of IEC 61499 function blocks,” Intl. Conference “Verteilte Automatisierung (Distributed Automation),” Proceedings, Magdeburg, Germany, March 2000.

Weehuizen, F., A. Brown, C. K. Sünder, and O. Hummer-Koppendorfer. “Implementing IEC 61499 Communication with the CIP Protocol,” In 12th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA’07), Proceedings, Patras, Greece, Sept. 2007.

Weehuizen, F., and A. Zoitl. “Using the CIP Protocol with IEC 61499 Communication Function Blocks,” 5th IEEE International Conference on Industrial Informatics (INDIN’07), Proceedings, pp.261–265, 2007.

Womak, J., D. Jones, and D. Roos. “The Machine that Changed the World: The Story of Lean Production,” Harper Perennial, 1991.

Xia, Feng, Zhi Wang, and Youxian Sun, (2004). “Design and evaluation of event-driven networked real-time control systems with IEC function blocks,” Systems, Man and Cybernetics, 2004 IEEE International Conference on, Vol. 6, pp.5148–5153, 10-13 Oct. 2004.

Yan, J., Vyatkin, V. “Distributed Software Architecture Enabling Peer-to-Peer Communicating Controllers”, IEEE Transactions on Industrial Informatics, 9(4), pp. 2200-2209 , 2013

Yoong, L. H., P. S. Roop, V. Vyatkin, and Z. Salcic. A Synchronous Approach for IEC 61499 Function Block Implementation, IEEE Transactions on Computers, Vol. 58(12), 2009, pp. 1599–1614, DOI: 10.1109/TC.2009.128.

Zoitl, A., T. Strasser, K. Hall, R. Staron, C. Sünder, and B. Favre-Bulle. “The Past, Present, and Future of IEC 61499.” In Proceedings of the 3rd international Conference on industrial Applications of Holonic and Multi-Agent Systems: Holonic and Multi-Agent Systems for Manufacturing, Regensburg, Germany, 3–5 Sept. 2007.

Zoitl, A., T. Strasser, C. Sünder, and T. Baier. “Best of both worlds: IEC 61499 in harmony with IEC 61131-3?” Industrial Electronics Magazine, 4(3), 2009.

Zoitl, A. Real-Time Execution for IEC 61499. USA: ISA and O^{3}neida, 2009.

Zurawski, R. (Ed). The Industrial Information Technology Handbook, CRC Press, 2005.

20 Index