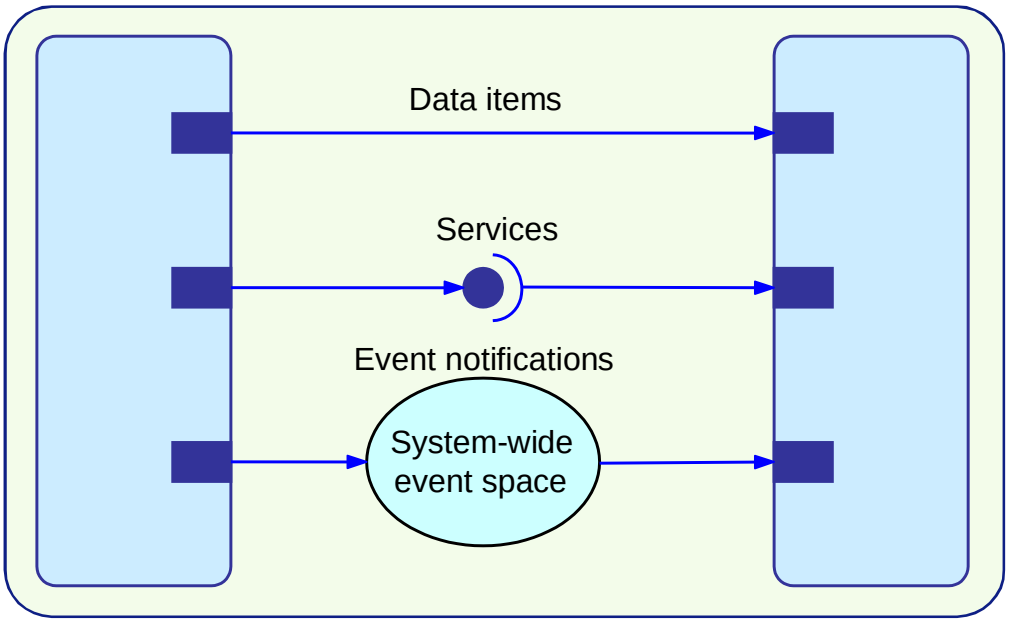


Teemu Tommila, Juhani Hirvonen, Lauri Jaakkola, Jyrki Peltoniemi, Jukka Peltola, Seppo Sierla & Kari Koskinen

Next generation of industrial automation

Concepts and architecture of a component-based control system

VTT TIEDOTTEITA – RESEARCH NOTES 2303

Next generation of industrial automation

Concepts and architecture of a component-based control system

Teemu Tommila, Juhani Hirvonen, Lauri Jaakkola & Jyrki Peltoniemi
VTT Industrial Systems

Jukka Peltola, Seppo Sierla & Kari Koskinen
Helsinki University of Technology

ISBN 951–38–6726–9 (soft back ed.)
ISSN 1235–0605 (soft back ed.)

ISBN 951–38–6727–7 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455–0865 (URL: http://www.vtt.fi/inf/pdf/)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 2000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 4374

VTT, Bergsmansvägen 5, PB 2000, 02044 VTT
tel. växel 020 722 111, fax 020 722 4374

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O.Box 2000, FI–02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 4374

VTT Tuotteet ja tuotanto, Tekniikantie 12, PL 1301, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 6752

VTT Industriella System, Teknikvägen 12, PB 1301, 02044 VTT
tel. växel 020 722 111, fax 020 722 6752

VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FI–02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 6752

Technical editing Anni Kääriäinen

Otamedia Oy, Espoo 2005

Tommila, Teemu, Hirvonen, Juhani, Jaakkola, Lauri, Peltoniemi, Jyrki, Peltola, Jukka, Sierla, Seppo & Koskinen, Kari. Next generation of industrial automation. Concepts and architecture of a component-based control system. Espoo 2005. VTT Tiedotteita – Research Notes 2303. 104 p.

Keywords industrial automation, control architecture, component-based control systems, local-area distribution services, information technology, domain-specific concepts, process automation, discrete manufacturing, systems architecture, implementation

Abstract

Recent trends in automation are characterised by geographical distribution and functional integration. On the technical level, the goal is to easily connect devices and software components from different vendors. Functionally, there is a need for interoperability of control functions on different hierarchical levels ranging from field equipment to process control, operations management and various Internet-based service applications.

This research report discusses current technological trends and outlines a new, component-based control system platform that supports the reuse of both system and application software. The working hypothesis is that next generation control systems will be a combination of new information technology and the domain-specific concepts found in process automation. The application area is industrial automation, including both process industries (continuous and batch) and discrete manufacturing. In addition, similar applications in other areas, such as environmental monitoring or distributed energy production, have been kept in mind. As this report is an outcome of a research project, the focus lies on concepts and system architectures rather than on hardware issues. To make it more useful however, also selected new implementation technologies have been described.

The discussion starts from business and technological trends and their implied requirements for future control systems. Then we introduce the conceptual model of a component-based control platform. Only general ideas of their implementation are given. Based on these concepts, the application needs and middleware architectures related to local-area distribution are analysed. Also a review of a few Ethernet-based communication standards is included. Next, the domain is extended to geographically distributed applications. This emphasises the role of remote monitoring, data-acquisition, and business-level services. In terms of implementation technology, service-oriented architectures and communication over public networks become more important. This report also describes a prototype implementation that was developed to clarify the suggested concepts and to evaluate some technical solutions. The final section sums up the key findings and lists areas that would need further research.

Preface

The research report at hand is a result of three consecutive projects sharing a common theme and the short name “OHJAAVA”. They were carried out by VTT Industrial Systems and the Helsinki University of Technology during 2001–2004 as a part of the “Intelligent automation systems” technology programme (ÄLY in short) organised by the National Technology Agency, Tekes. The following Finnish companies participated in the projects: JOT Automation (currently part of Elektrobit Group Oyj), Vacon Oyj, Raute Precision Oy, Vaisala Oyj and Metso Automation Oy.

The aim of the three OHJAAVA projects was to define functional concepts and implementation architectures for future automation platforms. The first one started from the basic idea of reusable “automation components” on a single computer node. The second project, called OHJAAVA-II, aimed to analyse the required interaction mechanisms and communication models in more detail. In addition, a number of emerging Ethernet-based distribution middleware solutions for local area control networks was evaluated. Finally, the OHJAAVA-III project focussed on geographically distributed applications. This scope emphasised the role of remote monitoring, data acquisition and maintenance services in addition to basic process control. Consequently, the possible applications of such systems go beyond traditional industrial automation, including for example, environmental monitoring, mobile working machines and distributed energy systems. While the Java language and related technologies were initially the key implementation technology of the OHJAAVA platform, the emphasis gradually shifted towards more generic functional principles. However, a number of limited Java demonstrations were developed during the projects for testing and clarifying the basic concepts.

This report summarises the key findings of the three projects. More detailed discussions can be found in the three master’s thesis produced during the projects. We hope that this report gives the reader some insights to the state-of-the-art and future directions of distributed control systems. We also wish to express our gratitude to the representatives of the participating Finnish companies, as well as to Tekes for making this work possible.

Espoo, May 2005,

Authors

Contents

Acronyms

API	Application Programming Interface
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial-Off-the-Shelf
CSMA/CD	Carrier Sense Multiple Access/Collision Detect
DCOM	Distributed Component Object Model
DCPS	Data Centric Publish Subscribe
DCS	Distributed Control System
DDS	Data Distribution Service
EDDL	Electronic Device Description Language
EJB	Enterprise JavaBeans
ERP	Enterprise Resource Planning
FBD	Function Block Diagram
FDT/DTM	Field Device Tool/Device Type Manager
GSM	Global System for Mobile communication
GPRS	General Packet Radio Services
HSE	High Speed Ethernet
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technologies
IDA	Interface for Distributed Automation
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISA	The Instrumentation, Systems and Automation Society
ISO	International Organization for Standardization
JMS	Java Message Service
JNDI	Java Naming and Directory Interface
LDAP	Lightweight Directory Access Protocol
M2M	Machine-to-Machine
MAC	Medium Access Control

MES	Manufacturing Execution System
NTP	Network Time Protocol
O^3NEIDA	Open Object-Oriented kNowledge Economy for Intelligent inDustrial Automation
ODVA	Open DeviceNet Vendor Association
OMG	Object Management Group
OPC	OLE for Process Control
ORB	Object Request Broker
OSI	Open Systems Interconnection
P&P	Plug&Play
P2P	Peer-to-Peer
PC	Personal Computer (IEC 61131: Programmable Controller)
PLC	Programmable Logic Controller
QoS	Quality of Service
RMI	Remote Method Invocation
RTPS	Real-Time Publish Subscribe
SCADA	Supervisory Control And Data Acquisition
SFC	Sequential Function Chart
SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
UDDI	Universal Description, Discovery and Integration
UDP	User Datagram Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
VPN	Virtual Private Network
WBF	World Batch Forum
WSDL	Web Services Description Language
XML	eXtensible Markup Language

1. Introduction

1.1 Problem statement and motivation

In the current highly competitive business environment, the industry is challenged by the demand for productivity, quality, safety and environmental protection. Tight profit margins and networked manufacturing emphasise the need for integration and global optimisation of production facilities. The role of information technology in achieving these goals has become critical. Large and complex production systems can not be efficiently and safely managed without advanced information management and process control. End users expect to get improved functionality at reasonable cost. Management of knowledge and real-time information, integration with condition monitoring and plant maintenance, high availability, flexible upgrades and life-cycle support are examples of key requirements. System integrators need efficient tools for building applications. Manufacturers face the challenge of satisfying customers' needs while still maintaining a sound and profitable product structure in a rapidly changing technical and business environment.

For many years, integrated, intelligent and dependable control systems have been the focus of standardisation organisations, industrial consortia and research groups. The solutions have, however, been hard to reach, partly due to the complexity of the issue and partly because of conflicting commercial interests. Only few efforts have provided practical results about common architectures and application objects that could make systems really "understand" each other. Technically, open control systems still focus on ways of making bits and bytes flow between devices from different vendors. There is also a gap between research results and real-life applications. Instead of elegant control theories, practical automation projects often struggle with low-level technical problems. Important issues are, for instance, how to find out user requirements, how to interface different products, and how to reuse existing application software.

As illustrated in Figure 1, earlier generations of digital control systems have been combinations of existing automation practices and advances in electronics and information technology. With the emergence of microprocessors in late 70's, faceplates of pneumatic controllers were transferred to computer screens of Distributed Control Systems (DCS). Since then, PC technology has, after a long debate, found its way to industrial applications. Current control systems are typically a mixture of many techniques.

We can expect that a similar situation will exist in the future also. Increasing processor power, consumer electronics, mobile communication networks and programming languages provide the tools to implement smart functions that have been unrealistic

before. Meanwhile, recent advances in the control domain have grown upwards from low-level programming languages and communication standards in the direction of more comprehensive data models and reference architectures. The increasing degree of horizontal and vertical integration leads to complex applications, which, in turn, calls for more powerful communication and computation models in control systems.

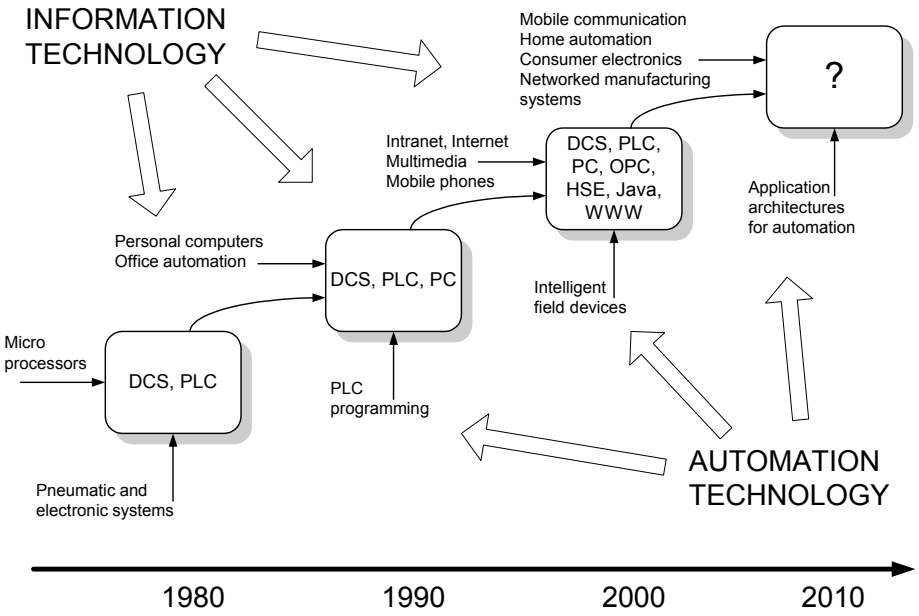


Figure 1. This report tries to define a new control system platform that combines the recent advances in information technology and industrial automation.

So, our working hypothesis has been that next generation control systems will be a combination of new information technology and the domain-specific concepts found in process automation and operations management. The focus has not been on hardware issues but rather on functional concepts related to applications and system software of control platforms. Therefore, the programming models and architectures used in software engineering have had an important role in the emergence of our future visions.

1.2 Scope and intended audience

The application area of the report is primarily industrial automation, including both process industries (continuous and batch) and discrete manufacturing. In addition, similar applications in other areas have been kept in mind. These include, e.g., environmental monitoring and distributed energy production. While the term "automation" in general has a wider scope, it refers here to rather low-level monitoring control of equipment and production processes. A *control system* is a collection of distributed devices and software interconnected by means of a communication network.

It typically performs functions like process monitoring and control, management of production operations and reporting. However, it is recognised that control systems must be integrated with higher-level business and service applications.

Recent trends in automation are characterised by geographical distribution and functional integration. On the technical level, the goal is to easily connect devices and software components from different vendors. Functionally, there is a need for interoperability of control functions on different hierarchical levels ranging from field equipment to Enterprise Resource Planning (ERP). Many customers already use, within certain security limitations, web-based "thin" clients for on-line knowledge management, remote monitoring and maintenance. Future systems will be based on an even stronger distribution to the field level, on mobility, and on co-operation with components in distant locations.

As this report is an outcome of a research project, the main focus lies on concepts and generic system architectures. The aim is to discuss recent technological trends and to outline a new, component-based control system platform on the basis of identified application requirements and enabling technologies. However, the authors hope that it is useful to practitioners and automation students, as well. To achieve this, current trends and new implementation technologies in industrial automation have been described.

1.3 Structure of the document

The thread of discussion in this report starts in chapter 2 from business and technological trends, and their implied requirements for future control systems. Then, chapter 3 introduces the conceptual models of a component-based control platform emphasising component interfaces and interactions. Only very general ideas of their implementation are given.

The focus of chapter 4 is on the application needs and middleware architectures related to local-area distribution. It also gives a review of a few Ethernet-based communication standards being developed in the control domain.

Chapters 3 and 4 together cover the scope of typical control applications. In chapter 5 the domain is extended to geographically distributed applications that are scattered over a wide area, perhaps even over the whole world. This emphasises the role of remote monitoring and maintenance, data-acquisition, and support for various business-level services. In terms of implementation technology, service-oriented architectures and communication over public networks become more important. In order to add some concreteness to the discussion, chapter 5 also describes a prototype implementation that

was developed to clarify our ideas and to evaluate some technical solutions. Finally, chapter 6 summarises our findings and lists some areas that would need further research.

This document can be characterised as a domain analysis and a suggestion for future directions. The treatment of the subject is informal. We have used common terminology whenever available, but defined some new ones to be more specific about our concepts. These key terms are printed in *italics* when appropriate. Diagrams commonly applied in software engineering, such as UML class diagrams, have been freely adopted in some sections. However, no special skills are required from the reader.

2. Trends in industrial automation

The purpose of this chapter is to introduce the key problems and requirements. The domain covers the monitoring and control of various processes and devices in the industry and public infrastructure. Applications are related to automation of manufacturing systems and environmental monitoring, for example. All these are characterised by local and geographical distribution and integration with company-level information systems.

2.1 Changes in the business environment

Industrial products, such as chemicals, publication papers or mobile phones, are becoming more complex and have higher added value than before. At the same time, the number of product variations increases and their life time gets shorter. For years, Western countries have focussed on high technology and moved bulk manufacturing to countries with lower labour costs. During the last decade, changes in the political situation and the economic development have accelerated the emergence of global markets. The growing markets in Far East and South America, for example, have created a situation where the manufacturing of high-tech products, and even parts of related R&D functions, are re-located close to major market areas. Tight competition and the increasing amount of societal regulation underline the importance of product quality, safety and environmental protection. The economics of scale and the expertise required in the manufacturing of complex products have led to co-operative networks of companies specialising in certain technologies or manufacturing operations. Along with globalisation, these networks expand from domestic to international co-operation.

These trends in industrial production have their implications for companies that develop and manufacture the required production facilities, such as process equipment, paper machines or diesel generators. Firstly, deliveries often go to foreign countries. The project staff must be able to co-operate with many other suppliers and contractors from various cultures. Secondly, customers need extensive support in the operation and maintenance of those complex systems. For other than big manufacturers, establishing an office close to the customer is too expensive. Unfortunately, outsourcing the services to a local operator may also be difficult because of the expertise needed, or because of the intellectual properties that must be protected.

Monitoring and control systems are basic ingredients of efficient, flexible and reliable production systems. Traditionally, monitoring and control applications have been purchased as separate systems and adapted to their operating environment on a case-by-case basis. There have been companies specialising in dedicated hardware and software products without necessarily having much knowledge about their application domains.

Recently, understanding the customer's business and manufacturing processes has become more important. In spite of the increasing computing power and lower hardware cost, developing and maintaining complex computer-based products is very expensive. This has led to acquisitions and fewer but bigger players in the market. Instead of starting from scratch, existing commercial solutions, such as PC hardware, operating systems and databases, are used whenever feasible. Partly influenced by the ongoing standardisation, the current Distributed Control Systems (DCS) and Programmable Logic Controllers (PLC) in the market are rather similar in their capabilities.

This trend has emphasised the role of application knowledge and customer service, whether the system is supplied by the original platform manufacturer or by an independent system integrator. The customer expects to get solutions rather than hardware and software. As a result, system suppliers shift their focus from individual projects to application expertise, extended after-sales services and strategic partnership with their key customers.

In addition to control system suppliers, information technology has become a critical issue for manufacturers of production equipment. These machines or processes are often very complex, possibly even safety critical, and can not be efficiently operated without automatic control. The know-how required for the operation and maintenance of these systems is a critical competitive factor for the equipment manufacturer, whether it is provided in the form of operating instructions, system specifications, control software, or as an actual control system embedded in the machine itself.

To summarise, control system suppliers must be able to implement more intelligent and reliable applications in shorter time and with lower cost in a global, multi-cultural environment. These remote systems must be maintained and upgraded according to changing production requirements. The business in basic data acquisition and process control has matured and is now growing rather slowly. Therefore, new opportunities are looked for in improved customer service and enhanced application functions.

2.2 Application needs in process control

The dynamic business and technological environment makes the far-sighted allocation of research and development resources very important. In several countries, national technology roadmaps have been written to give a good basis to the decisions. The Finnish discussions have identified several important development areas, such as intelligent data processing, control system architectures, human-system interaction, plant operational state management, abnormal situation management and standardisation of information flows (Ventä 2005). The following analyses some of these issues in more detail.

Intelligent machines

The increasing number of product grades and grade changes, combined with the demand for high availability and rapid reconfiguration, emphasises the need for flexibility in plant operation. To make this possible, the manufacturing equipment should be seen as intelligent resources capable of performing multiple tasks in co-operation. While the initial prerequisites are determined by process and mechanical engineers, the control system has a key role in putting the inherited potential into practice.

The physical production resources in an industrial plant consist of individual devices (e.g. tanks and pumps) and larger subsystems (for example equipment modules and process units defined in ANSI/ISA-88.00.01-1995 for a batch plant). This leads to a wholes-parts hierarchy as shown in Figure 2. Process equipment can be in different *operational states*, such as "maintenance", "starting up" or "operating". In each state, they provide a set of *capabilities* (ANSI/ISA-95.00.01-2000) that can be combined to perform the various stages of the manufacturing process.

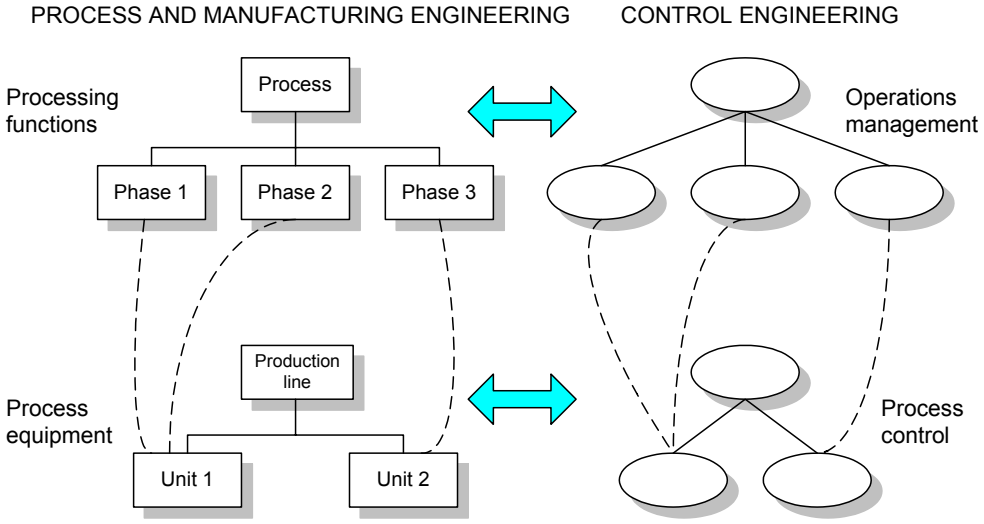


Figure 2. Process entities and automation activities are arranged as hierarchies. Capabilities of process equipment are used to carry out process phases. In the control system, production oriented functions of operations management send commands to process control functions associated with physical equipment (Tommila et al. 2001).

For example, process units in a multipurpose batch plant or machines in a flexible manufacturing system have several capabilities and can be used for different purposes. A pool of resources can even be re-configured into "virtual production lines". In the control domain, various product recipes are defined on the basis of services programmed on the equipment control level of the control system. Each process control component takes care of the process equipment it represents, including for instance

- reading and validating process measurements
- performing control actions
- managing physical resources
- controlling operational state and operating mode
- condition monitoring and exception handling
- providing services for other activities.

This arrangement makes the physical manufacturing resources behave in an "intelligent" way and to provide high-level services for operations management functions, such as product specific control recipes. In addition, a start-up sequence or a critical safety function (e.g. cooling of a chemical reactor) can be thought of a service or capability. The actual implementations may vary. For example, controls can be provided by the equipment manufacturer and embedded into the physical device (e.g. intelligent valves). The functions can also be included into the main control system.

Improved application structure

In the course of control system design, control tasks identified in co-operation with users and other engineering disciplines are allocated to the control system and human operators. The automated parts should form a structured set of control activities corresponding to the physical equipment and processing tasks. Automation activities may consist of lower-level functions or be implemented in other techniques, such as function blocks (IEC 61131-3 2003, IEC 61499-1 2005) or a general purpose programming language. A product recipe, for example, may consist of several unit recipes.

A component logically contains the components controlling the lower-level process systems even if they are allocated to other control devices. The allocation is usually static, but an activity can also be re-allocated to another computer if one platform fails or becomes overloaded. During operation, a component can acquire external resources owned by other components. Managing shared resources is essential for implementing flexibility in control systems. The hierarchical relations to owners and clients should be maintained in the run-time environment. For example, they can be used to propagate component status allowing upper levels and clients to react to device failures.

Reuse of design and software

To enhance design and to improve quality, applications should, as much as possible, be based on proven solutions. In the current design practice, this seldom happens in a systematic way. Instead, designers copy relevant material from the documentation and program code generated in earlier projects. Standard solutions have been defined only for common components like block valves and electric motors. The goal should be to reuse larger plant segments. The emerging standards, such as IEC 61131-3 (2003) provide tools

for this. In the first phase, companies can develop function block libraries for the control products they use. Later on, standardisation of portable and platform independent programming languages opens the way for open markets of control software.

With the increasing complexity of applications, the established programming models of control engineering are, however, not fully sufficient. In our vision, they should be complemented with the principles commonly used in software engineering. Control functions like sequences and control loops could be similar to objects in object-oriented programming with the difference that they, in addition to responding to external messages, have internal activities. As in the case of function blocks, the external interface of automation objects includes input and output terminals (ports), but in addition to data, the terminals should be able to handle services and event notifications. "Wiring" is the common paradigm currently used in function block programming. Also other distributed programming models, such as message queues or shared memory, should be considered for control applications. These approaches have, so far, been more common in information systems and object-oriented programming, although they are becoming more familiar to control engineers, along with standards like DCOM, OPC and CORBA.

Flexible re-configuration

Product and process changes requiring modifications to the control system are more frequent than earlier. In industrial applications, system updates must often be carried out on-line without interruptions in production. A further demand for flexibility comes from availability requirements. The control platform must support redundant hardware and software.

Therefore, the control platform must have built-in re-configurability at runtime. Control devices and automation components must be able to describe themselves to designers, human operators and other automation components. During system operation, newly installed devices and software components must have ways of looking up the rest of the application and to advertise their own capabilities. To make this easier, devices and network segments can be arranged to "system areas" in a hierarchical fashion. Root devices in each area can maintain directories of other nodes. This results in a distributed directory service embedded into the control system itself.

Exception handling

In addition to changes in products and production schedules, control systems should cope with other types of abnormal situations, namely unexpected disturbances originating from process fluctuations, failure of process equipment or faults in control system hardware and software. The scope of exception handling covers issues from fault avoidance in design to problem identification and display, diagnosis, corrective

actions and recovery during system operation and continuous improvement on the basis of problem reports.

Even if as much as 50 % of control system software and design costs are related to exception handling, only a few practical approaches and tools are available. For example, alarm floods are still a problem. While general-purpose programming languages, such as Java, have at least some mechanisms for catching and processing runtime exceptions, the application-oriented languages used in automation provide virtually no support for this. Nor are there any widely accepted ways to propagate exception-related events between different software modules.

Although actual algorithms depend on the situation and equipment under control, the control platform and application should include built-in features that make abnormal situation management easier. For example, automation components can have a "health" attribute that describes, on a qualitative scale, its performance. Intelligent, situation-aware, event generation implemented on the spot would limit the generation of nuisance alarms and the need for alarm filtering in the user interface. The designer should be able to specify monitors running in parallel with the normal actions. Depending on the situation, they could force control functions (e.g. sequences) to an appropriate exception routine. General principles of Quality of Service (QoS), like request priorities, performance figures for services and deadlines for network messages, could also be included in the basic control platform.

Interoperability

Current applications are combinations of control products from different vendors. A single supplier can not provide all the equipment and software required in a large application. In addition, the customers do not want to become dependent on one manufacturer. Therefore, there is the need for

- interfacing various control devices that execute manufacturer-specific software
- porting application software from one platform to another
- combining software components from different sources on a single platform
- integrating basic control functions with different types of local and remote user interfaces
- connecting application functions from different domains, e.g. process control and condition monitoring.

This problem area is often referred to as the standardisation of "open control systems". The next section will give an introduction to the subject.

2.3 The emergence of open control systems

Distributed control applications are implemented in many ways. Figure 3 shows one possible structure and examples of system elements, typically organised in several physical layers. Correspondingly, different types of communication networks are needed. Process inputs and outputs (I/O) populate the bottom layer. Intelligent sensors and actuators attached to a fieldbus are becoming more popular, but traditional field equipment and I/O modules still have an important role. The information processing tasks in a control application are performed by programmable controllers, operators' workstations and dedicated servers for the collection, storage and reporting of production information. A possibly redundant control network, typically based on switched Ethernet, links these nodes together. The "basic process control system" is connected to the plant/process network and to the supervisory control and information processing applications residing on the next level. As an indication of the widening scope of process control, all the tree system layers make up what can be called an "automation system". In smaller applications, however, the term only covers field instruments, controllers, user interfaces and basic data collection and reporting. An automation system connects to various office applications, and the Manufacturing Execution Systems (MES) and Enterprise Resource Planning (ERP) applications of the company. Web servers and other Internet technologies are a typical approach to connect the local automation system to the corporate network.

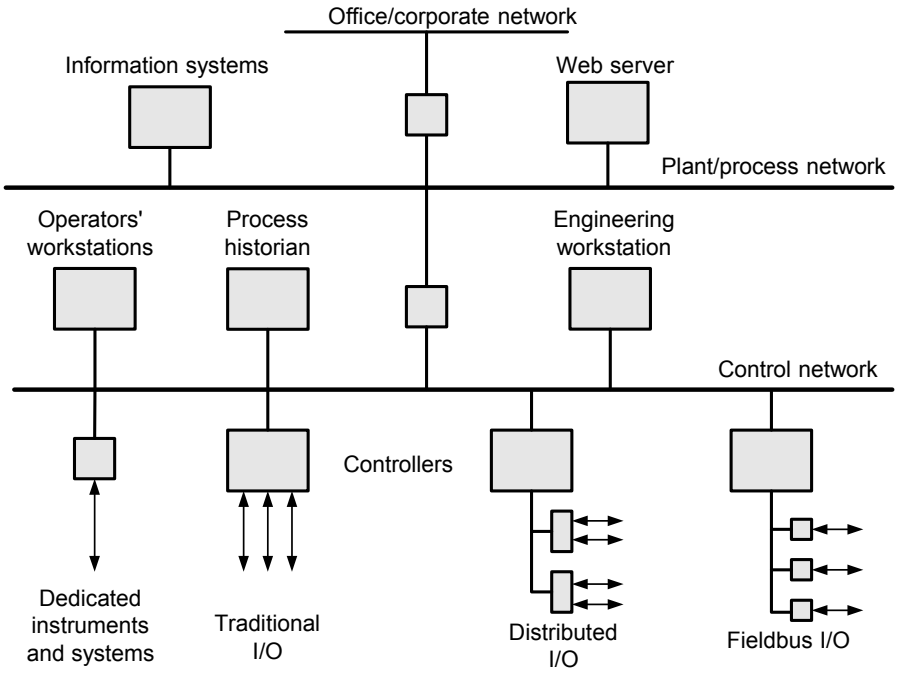


Figure 3. Elements of an industrial automation system.

Distributed Control Systems (DCS) and Programmable Logic Controllers (PLC) are currently the mainstream technologies in basic process control. DCS platforms have their background in large-scale continuous applications and regulatory control in the process industry. PLCs, for their part, were originally developed to replace hard-wired relay logic in discrete manufacturing (see e.g. Webb & Reis 2003).

DCS platforms are integrated product families usually provided by one manufacturer and include facilities for user interfaces and information management. PLCs, on the contrary, are mainly limited to low-level control functions. Therefore, personal computers and databases are required for more extensive information processing. PC-based operating panels are commonly used as the user interface. With intelligent field devices (e.g. control valves, drives and process analysers) capable of executing application software a third system architecture is emerging. Small applications can be built by connecting fieldbus devices directly to a PC.

To reduce development and manufacturing costs, manufacturers tend to adopt common hardware and (open source) software (PC's, Ethernet, TCP/IP, Windows, Linux, etc.) in their product development. In some cases, they even use the same system components as their competitors (e.g. PC-based user interfaces and information management products). In spite of this, control products and applications are not interoperable due to different data models and upper levels of communication protocols.

The requirements for integration, vendor independence and software reuse are widely recognised by the automation community. Numerous standardisation efforts are going on to solve these problems. We discuss below selected issues of interest to our scope.

Programming languages

Basically, most control platforms use similar approaches to building applications. These include form filling, instruction lists in the style of assembly languages, function blocks and ladder diagrams. General-purpose programming languages are usually available for complex control algorithms and data processing tasks. The International Electrotechnical Commission (IEC) has published several standards for programmable controllers. Even if they are widely supported by PLC manufacturers, the number of products conforming to the specifications is rather limited (see PLCopen 2004). As a result, application programs can not be easily ported from one platform to another. This means that system integrators are tied to one product or must maintain parallel versions of their code for the platforms they use.

In particular, the IEC standard 61131-3 (2003) is relevant to the scope of this report (a practical textbook has been written by John & Tiegelkamp in 2001). It defines the basic

elements of an application as shown in Figure 4. A *configuration* corresponds to a control application. A *resource* represents a "signal processing function", such as a processor on a PLC. A configuration contains one or more resources, each of which contains one or more *programs* executed under the control *tasks*. A task can be triggered on a periodic basis or upon the occurrence of the rising edge of a boolean variable. A program may consist of *function blocks* or language elements in the other programming languages defined in the standard. An *access path* defines a symbolic name of a variable, whereas *directly represented variables* identify, in a manufacturer-specified way, the physical location of data (e.g. the hardware address of an input signal). *Global variables* are visible everywhere within a resource or configuration.

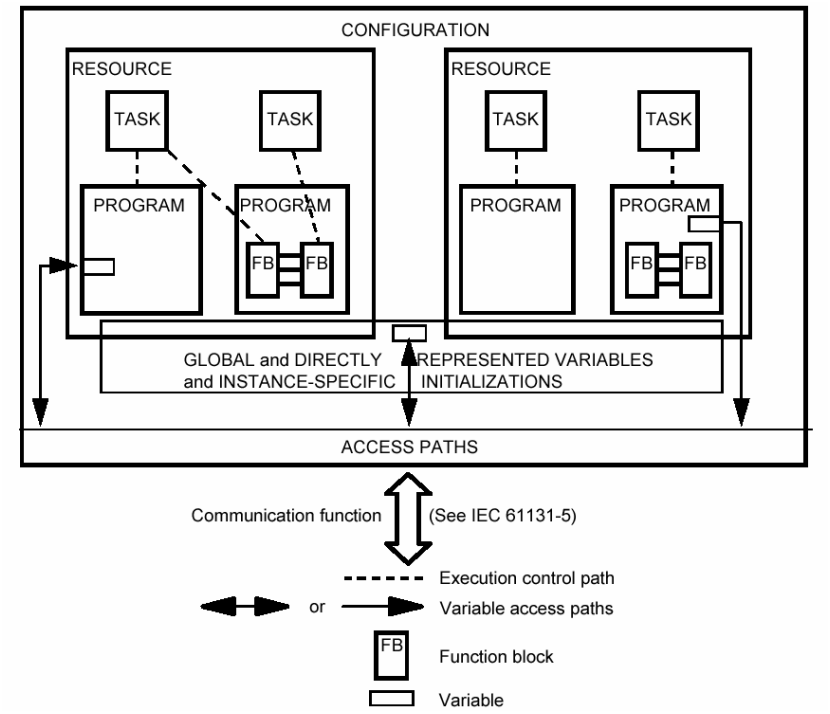
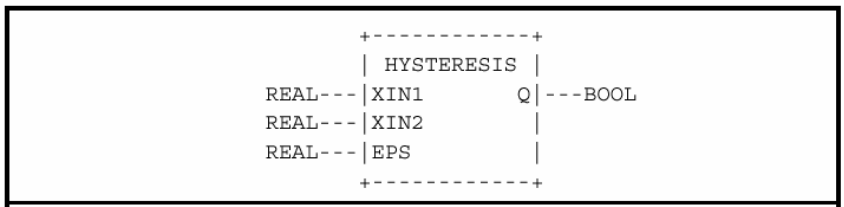


Figure 4. Software model of a programmable controller system (IEC 61131-3 2003).

IEC 61131-3 defines two textual languages, namely the *instruction list* and *structured text* (with a Pascal flavour), and two graphical languages, the *ladder diagram* and *function block diagram*. In addition, a *Sequential Function Chart* (SFC) consisting of steps and transitions is defined for performing sequential control functions. The SFC model is derived, with some changes, from the documentation standard IEC 60848 (2002), known as GRAFCET.

Functions and function blocks belong to the common elements used in all the programming languages. A *function* is a procedure that produces one (possibly multi-valued) data element, the function result, and arbitrarily many additional output elements. Functions contain no internal state information, i.e. invocation of a function with the same arguments always yields the same outputs. A *function block* consists of

input, output, and internal variables and a set of operations to be performed when the function block is invoked. So, a function block has an internal state that has an effect on the results of subsequent invocations. A *function block* is an instance of a *function block type.* There can be an arbitrary number of function block instances corresponding to the same function block type. IEC 61131-3 defines a set of standard functions and function blocks. In addition, the programmer can construct new company and application specific functions and function block types, such as the one shown in Figure 5.




```
FUNCTION_BLOCK HYSTERESIS
    (* Boolean hysteresis on difference *)
    (* of REAL inputs, XIN1 - XIN2      *)
  VAR_INPUT XIN1, XIN2, EPS : REAL; END_VAR
  VAR_OUTPUT Q : BOOL := 0; END_VAR
  IF Q THEN IF XIN1 < (XIN2 - EPS) THEN Q := 0; END_IF ;
  ELSIF XIN1 > (XIN2 + EPS) THEN Q := 1 ;
  END_IF ;
END_FUNCTION_BLOCK
```

Figure 5. A function block type that implements hysteresis on the difference between two inputs XIN1 and XIN2 (IEC 61131-3 2003).

Function block diagram (FBD) is a graphical representation of a control activity consisting of function block instances, graphically represented functions, variables and constants. It can also be used to represent the body of a function block type derived from existing function blocks. Figure 6 gives an example of a function block diagram.

IEC 61131-3 is not exact in all aspects but leaves many implementation details to the controller manufacturer. The evaluation order of program elements is an example of issues not fully specified. The technical report IEC/TR 61131-8 (2003) provides guidelines for the implementation of the languages in programmable controller systems and their programming environments. In addition, guidance is provided by PLCopen (http://www.plcopen.org), a vendor-independent association that supports the use of international standards in the field of control programming. In addition to promoting the use of IEC 61131, a significant aspect of PLCopen's work is the certification of compliant programming systems. It has also published a library of function blocks for motion control based on the IEC 61131-3 function block concept.

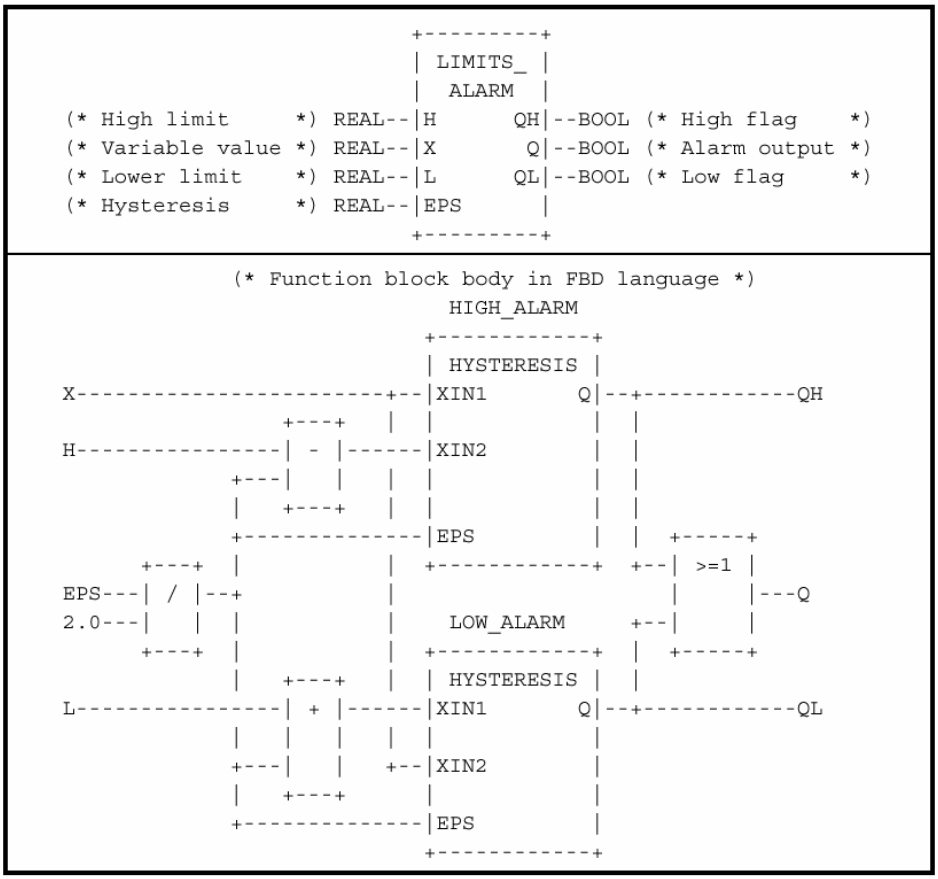


Figure 6. This function block implements a high or low limit alarm with hysteresis on both outputs (IEC 61131-3 2003).

Function blocks are a traditional and natural programming paradigm for many control applications. In addition, they provide a good basis for information encapsulation and reuse of application software. So, function blocks can be thought of as one form of "automation components". Function blocks are also an important concept in several standards and industrial specifications beyond IEC 61131-3. In particular, the new standard IEC 61499 (see Lewis 2001) specifies a general function block model for distributed control applications. In addition to data inputs and outputs, the external interface of an IEC 61499 function block consists of event ports as depicted in Figure 7. Event inputs receive events, which may trigger the execution of one or more algorithms. The algorithms, in turn, generate data and new events and deliver them through the data and event outputs to other function blocks. Specific data inputs can be associated with input events to indicate that certain data must be sampled prior to execution of an internal algorithm. In the same way, modified data outputs can be tied to an event.

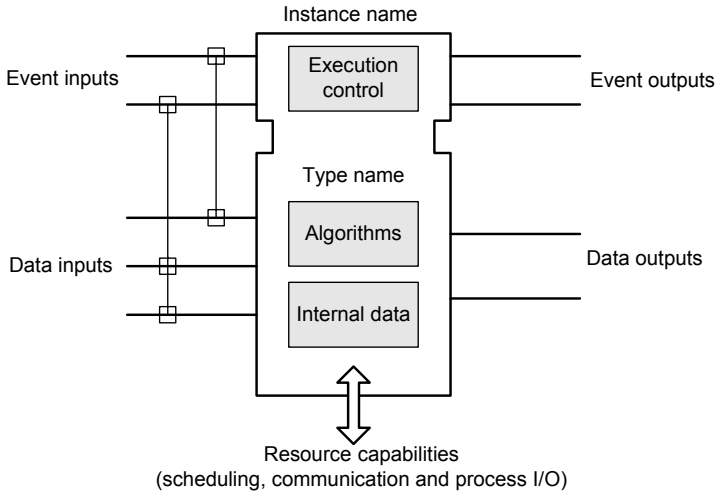


Figure 7. Basic structure of a function block (IEC 61499-1 2005).

IEC 61499 uses the programming languages of IEC 61131-3 for defining the algorithms internal to a function block. Special service interface function blocks (SIFB) are defined for accessing data and services, such as process I/O and communication, provided by the underlying resource. The main difference as compared to IEC 61131 lies in the event-driven nature of an IEC 61499 function blocks. Event connections determine the execution order of a function block diagram in an unambiguous way (Figure 8). If the blocks are allocated to distributed resources, communication function blocks must be added. In addition to synchronisation of program execution, event connections provide an implementation mechanism for the design of inherently event-driven control applications.

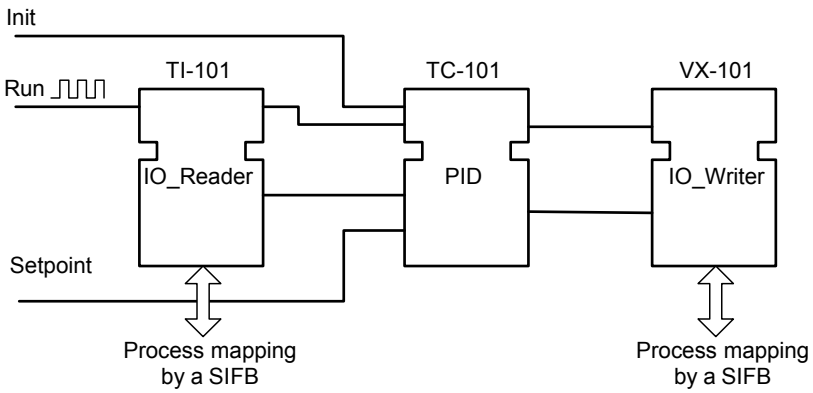


Figure 8. A simple temperature controller uses service interface function blocks for process I/O and a regular stream of events for timing.

There are currently very few commercial control devices and programming tools supporting the IEC 61499 function block model. In spite of that, the concepts have been used as a basis for several other standards. For example, IEC/TS 61804-1 (2003) defines

a set of basic function blocks that will be required for fieldbus applications in the process industries. In addition, a couple of developments in the field of industrial Ethernet, e.g. PROFInet and Modbus-IDA use IEC 61499 as a reference model for the application layer. An introduction to these can be found in section 4.3

Communication technologies

Data transfer protocols and the distribution services built on them are an essential feature of distributed control systems. The Open Systems Interconnection (OSI) model developed by the International Organization for Standardization (ISO) is often used as a reference model in structuring data communication. It consists of seven layers:

1. Physical layer: The actual wire or wireless connection, such as twisted-pair wiring, fibre-optic cable or radio link, and the components that connect the network to the devices. Basic Ethernet serves here as an example.
2. Datalink layer: A processor or a chip that tells the components how to communicate. It adds error checking and addressing information to the basic frames and controls the access to the networking media. A distinction can be made between deterministic Medium Access Control (MAC), often used in fieldbuses, and random procedures (e.g. CSMA/CD with Ethernet). This level also defines the type of communication, such as master-slave.
3. Network layer: A protocol that determines how the information is moved across the network. The layer performs network routing, flow control, segmentation and de-segmentation, and error control functions. An example is the Internet Protocol (IP).
4. Transport layer: Controls the reliability of a link. Some protocols, such as the Transmission Control Protocol (TCP), are stateful and connection-oriented. The layer keeps track of packets and retransmits those that fail. The second widely used technique, the User Datagram Protocol (UDP) on the other hand is stateless and thus faster. Packets are not acknowledged by the receiver, so possible errors are not fixed.
5. Session layer: Provides the mechanism for managing the dialogue between end-user applications. For example, this layer is responsible for setting up and shutting down TCP/IP connections.
6. Presentation layer: Relieves the application layer from syntactical differences in data representation. MIME encoding, encryption and similar manipulation of data format is done at this layer.
7. Application layer: Determines the meaning of the messages and performs services directly for the application processes. Typical examples include protocols like FTP, HTTP and POP/SMTP.

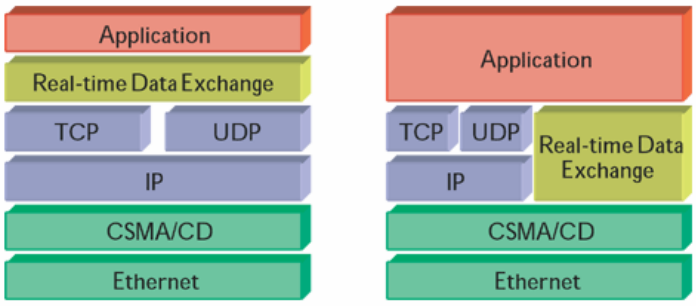


Figure 9. Possible ways of adding real-time communication to the common OSI protocol stack (Schwab 2004).

In control applications, real-time performance and light-weight solutions are an important issue. Communication protocols have been implemented in different ways. Real-time data exchange on Ethernet can, for example, be based directly on the transport layer (most often on UDP). Alternatively, layers 3 and 4 can be replaced by a dedicated protocol (Figure 9). Switches can be used to remove the problem caused by the non-deterministic medium access of Ethernet. If critical timing requirements exist, special hardware is also needed. Fieldbuses are typically resource-limited. They often drop layers 3–6 and use their own solutions at the physical and datalink and application layers.

The basic data exchange functions are not sufficient in control systems. For example, distributed applications must be able to locate the required data sources, establish the connections and maintain them in a dynamic and error-prone environment. System manufacturers and standardisation organisations have selected various approaches to the problem. Some of them utilise naming or directory services that allow dynamic configuration in the runtime environment, while in others, communication links are resolved in advance by the engineering tool. Both centralised and distributed, peer-to-peer architectures have been suggested in current middleware solutions. Section 4.2 gives a more detailed analysis of these issues.

Further, control applications need a system-wide sense of time with a resolution of a millisecond, or below. Existing protocols for clock synchronisation are not at the optimum in the control domain. For example, the Network Time Protocol (NTP) targets large systems with an accuracy of milliseconds. The new standard IEC 61588 (2004) issued by IEEE and IEC defines a high-precision synchronisation protocol for distributed measurement and control systems. It is independent of any networking technology, and the system topology is self-configuring.

Communication aspects have been the subject of extensive standardisation activities in official standardisation organisations and industrial groups. The following is a short

review of the situation in fieldbuses, control and process networks as well as in Internet and corporate networks.

Fieldbus technology was developed in the 1980's to replace wired I/O. Fieldbus systems are digital, serial, multi-drop and two-way communication networks for information transfer between primary controllers and intelligent field devices, such as sensors and actuators. Fieldbuses were designed for low-volume and real-time data transfer. Transmission of data is basically cyclic. Token passing and master-slave methods are used for deterministic medium access. Additional alarms, parameter modifications and diagnostic data are transmitted asynchronously as required. The benefits expected from digital communication include, for example, reduced wiring costs, shorter installation and start-up time, improved on-line monitoring and diagnostics, easier modifications and improved local intelligence.

Due to different industry-specific demands and proprietary solutions of large manufacturers, several bus solutions emerged. This was a problem to end-users, who would like to see one or at most a few alternatives with interoperable and interchangeable products. Extensive standardisation was started in the 1990's to relieve this problem. The International Electrotechnical Commission (IEC) and the Instrumentation, Systems and Automation Society (ISA) have the main responsibility in this work. In addition, there are numerous industrial groups and user organisations involved. However, the established fieldbus technologies already had such a strong position that the result was a number of standards instead of one. The most important solutions are specified in international and European standards like IEC 61158, IEC 61784 and EN50170. The following lists some of the most common fieldbuses.

- Controller Area Network (CAN) was developed in the early 1980's to simplify the wiring in automobiles, but it has spread to machine and factory automation. CAN is the basis of several sensor buses, such as DeviceNet. CAN in Automation (CiA) is an organisation that supports CAN and develops CAN-based higher-layer protocols, such as CANopen (http://www.can-cia.org/cia).
- AS-Interface (AS-i, see http://www.as-interface.com) is a cost effective solution for the lowest level in the automation hierarchy. AS-i connects binary sensors and actuators with the primary control level, e.g. PLCs or PCs in factory automation.
- Modbus is a widely used messaging structure developed by Modicon in 1979. It does not define a physical layer. Normally, serial lines (e.g. RS-232 or RS-485) are used as the physical layer. Modbus and its successor, Modbus TCP/IP, are currently developed by Modbus-IDA (http://www.modbus-ida.org).
- HART (Highway Addressable Remote Transducer) was originated by Rosemount in the late 1980's. Different from other fieldbuses, it allows the transfer of digital data

over the analogue 4–20 mA wires that are an old standard in the process industry. The HART Communication Foundation is a non-profit organisation that supports the application of the HART (http://www.hartcomm.org).

- DeviceNet is a low-cost link to connect limit switches, photoelectric sensors, valves, motor starters, etc. Application areas include automotive, electronics manufacturing, batch chemical processing, packaging and material handling. DeviceNet is developed by ODVA (http://www.odva.org).
- Profibus is developed in several versions by the Profibus User Organization (PNO) and Profibus International (http://www.profibus.com). For example, Profibus-DP is for high-speed data communication in factory automation while Profibus-PA is intended for process automation.
- In contrast to most fieldbuses that originated from a specific geographical area or a company, Foundation Fieldbus is a result of a merger of international standardisation efforts. It is used in process and manufacturing applications. It is developed by the Fieldbus Foundation (FF, http://www.fieldbus.org).

Simple sensor-actuator networks, such as CAN and AS-i, were designed for bit- or byte-oriented interfaces, primarily in discrete manufacturing. More complex fieldbuses provide analogue and digital support for more advanced devices. These include Foundation Fieldbus and Profibus, for example. In addition to transferring process and diagnostic data, special versions of the basic solutions have been developed for explosive atmospheres, safety-critical applications, etc. In the most advanced fieldbuses, i.e. in Foundation Fieldbus and Profibus-PA, it is possible to run control functions in the intelligent field devices. The technical specification IEC/TS 61804-1 (2003) defines a set of basic function blocks for this purpose.

In spite of the standardisation, the large number of alternatives and the non-standard, manufacturer-specific features still complicates the application of fieldbus technology. A failed device can not necessarily be replaced by another product, and several engineering tools are often needed to configure the various types of field devices. The situation is further complicated by the introduction of new bus technologies from the general-purpose information technology, such as USB, Bluetooth and FireWire. Two competing approaches have been suggested to solve this problem. In the first one, *profiles* are used to specify the standardised properties and behaviour of interoperable devices in specific types of applications and systems. An *Electronic Device Description Language* (EDDL) is the way to specify parameters and functions of a field device. EDDL is a part of the standard IEC 61804. The second approach applies the FDT/DTM concept coming from the PROFIBUS community. In it, device specific features are encapsulated by the device manufacturer as a software component called *Device Type*

Manager (DTM). These "device drivers" are then imported to a manufacturer and protocol independent framework application, the *Field Device Tool* (FDT).

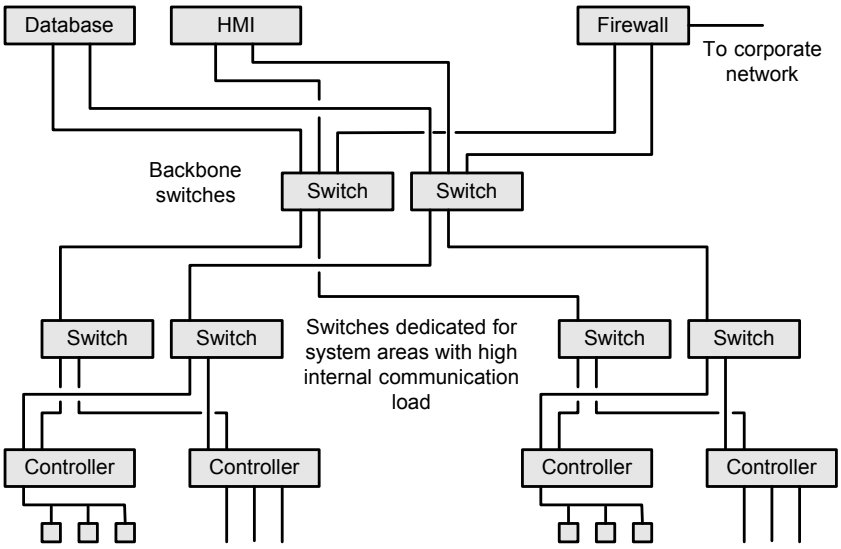


Figure 10. An example of a redundant control network based on switched Ethernet.

On the level of control network, communication protocols have typically been, and still are, specific to DCS or PLC manufacturer. While switched high-speed Ethernet, as shown in Figure 10, is currently a common data transfer technology, the communication protocols and application concepts differ. The lack of interoperability is a major inconvenience to system integrators and end users as a large number of tailored interfaces must be developed and maintained. OLE for Process Control (OPC) is an early attempt to reduce integration costs by defining a standard way for reading and writing data items from and to heterogeneous control devices (Figure 11).

Since its introduction in the mid 90'ies, OPC has become a de-facto standard for retrieving data from DCS and PLC applications to human-machine interfaces (HMI) and historical databases. In addition to the basic and most popular Data Access specification (OPC DA), OPC Foundation (http://www.opcfoundation.org) has developed a series of techniques for alarms and events, historical data access, batch process control, etc. With the new Data eXchange specification (OPC DX) introducing horizontal communication between various Data Access servers, the initial focus of OPC is extending from the client-server architecture towards peer-to-peer (P2P) communication on the controller level.

Basically, OPC is based on Microsoft's DCOM technology and consequently more or less tied to the Windows operating system. A web-enabled counterpart of OPC Data Access, the OPC XML-DA specification, has been developed to solve this problem. It uses XML and the Web service technology to achieve platform independence and to

circumvent the limitations that DCOM has with respect to Internet access and firewalls. In addition, the DX specification includes a Web service binding. As a more recent initiative, the OPC Unified Architecture working group is preparing means to integrate existing OPC specifications under a common data model. This data model should facilitate application integration especially in cases where data is distributed behind several different OPC interfaces. Also the OPC Unified Architecture utilises the Web service technology.

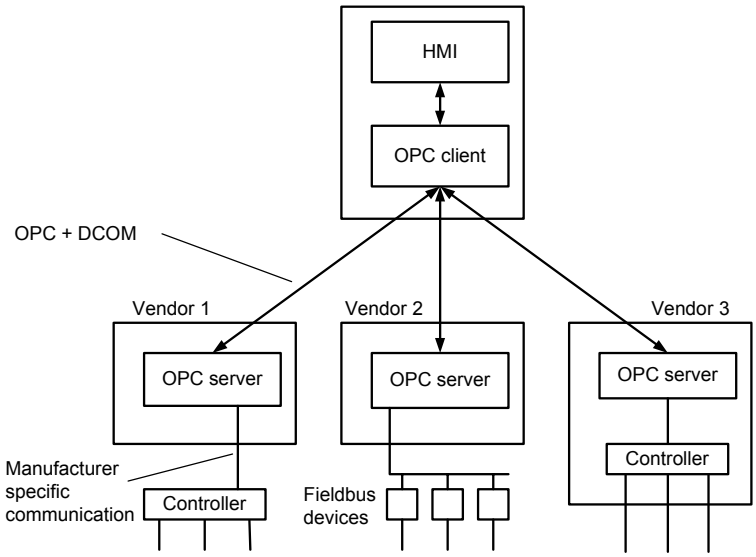


Figure 11. The basic idea of OPC is to allow standardised connections to various DCS and PLC devices based on different and often proprietary communication protocols.

OPC is an example of applying Ethernet and general purpose information technology to industrial automation. Since the introduction of OPC, several other examples have emerged. Some of these (e.g. Ethernet/IP and Modbus-IDA) are based on existing fieldbus protocols, whereas some (such as CORBA) have their background in distributed information systems. Different from OPC and CORBA, which have primarily focussed on communication aspects, some new specifications in the control domain (e.g. PROFInet and Ethernet/IP) aim at a more comprehensive model of a distributed control system including concepts and services at the application layer, too. The function blocks discussed above are a common approach for modelling control applications. Section 4.3 below discusses some of these techniques in more detail in the context of local-area distribution services.

The system supplier needs a remote access to the control system for diagnostic and maintenance purposes. Before, telephone lines and modems were often used, perhaps with a prior call asking the client to plug in the modem connector. Dedicated solutions are still sometimes needed in distant locations. However, public networks and Internet protocols are the most common solution today. Even small control devices and data

acquisition equipment can now have an embedded web server, and some of the emerging standards include specifications of basic functions of remote maintenance and diagnostics.

With the rise of service business and remote operation, the scope of information exchange has expanded. Web services and related domain specific standards, like OPC XML-DA, have become more important. Simultaneously, the number of parties having a need for remote access has also grown. This has created a security problem, especially in applications with high asset values and safety requirements. Customers are often reluctant to grant suppliers and service providers an access to the control system. General-purpose techniques, such as Virtual Private Networks (VPN) and secure versions of IP protocols, provide feasible solutions, but their efficient application is a challenge to most end-users and control system suppliers. The problem has been recognised by the automation community, and some guidelines have been prepared, for example the ones compiled by the Instrumentation, Systems and Automation Society, ISA (ISA-TR99.00.01-2004 and ISA-TR99.00.02-2004).

Domain models and architectures

It has long been the goal of the industry to integrate the operating units of a plant in order to produce products at minimum cost and at maximum performance. This has led to closely coupled production units, minimised in-process inventories, and maximum use of energy and material resources. More recently, integration has expanded to multiple plants, business processes and supply chains. To make world-class manufacturing networks possible, tight quality control and scheduling, and therefore novel process control and information systems, are a necessity.

The rather low-level technologies described above can be used in the development of such large-scale systems. However, also higher-level standards and reference models are necessary to define the common terms, information content and function structure of an application domain. The purpose of these models is to improve common understanding and co-operation between the numerous parties involved, and to provide a sound basis for computerised tools and data exchange.

Several specifications of this type have been and are being developed in various branches of the industry. In particular, the activities of the working group SP95 of the Instrument, Systems and Automation Society (ISA) are relevant to our scope. The goal of SP95 is to define the interface between control functions and other enterprise functions, including data models and exchange definitions, as shown in Figure 12 (based on the Purdue Reference Model for CIM). The committee has published the first two standards of the series. ANSI/ISA-95.00.01-2000 provides standard terminology and a set of concepts and

models for integrating control systems with enterprise systems. The second standard (ANSI/ISA-95.00.02-2001) contains additional details and examples of the object model. The information in the SP95 model falls into three main areas:

- Product definition: How to produce a product
- Production capability: What are the physical resources available at the plant
- Production: Information about actual production of the product, plan and results.

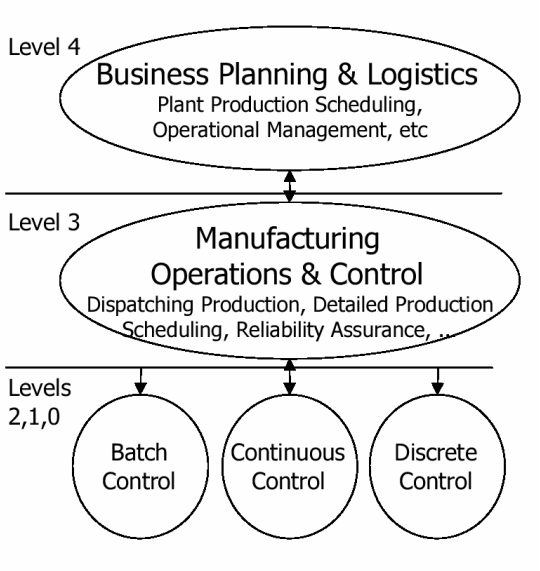


Figure 12. Functional hierarchy of an integrated system (modified from ANSI/ISA-95.00.01-2000).

In fact, the object model defines a comprehensive but abstract plant model including, for example: product properties and required processing steps; equipment structure, capabilities and status; personnel availability and capabilities; material and inventory data; production schedules and histories; and maintenance information. The basic entities in the model are classes and instances of production resources, their capabilities and associated properties (Figure 13). For example, process equipment, combined with the required personnel and materials, define a hierarchical set of resources each having certain operational capabilities (e.g. storage, transfer or processing of material or energy) that can be used for manufacturing a specific product.

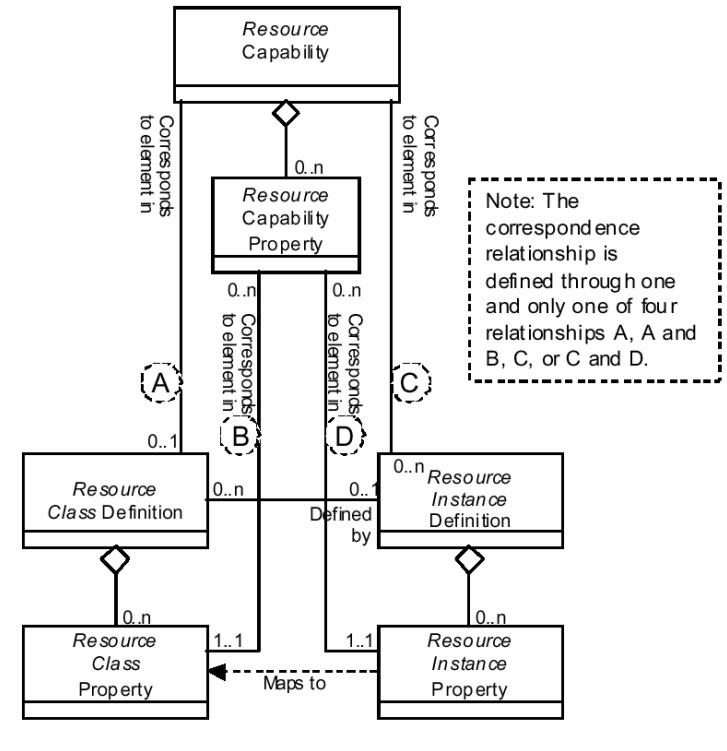


Figure 13. Basic model elements and their relationships (ANSI/ISA-95.00.02-2001).

This kind of a plant model gives a good basis for structuring the control application and its data. In this way, the plant can be seen as a population of "intelligent machines" as discussed earlier in section 2.2. The modelled pieces of equipment and functions are reusable entities and, thus, obvious candidates for automation components, the major topic of this report.

The work of ISA-SP95 was taken here as an example, but there are other similar efforts going on in other application areas. For example, SP95 has been influenced by the earlier standards of ISA-SP88 on batch process control. Based on the initial definition of functional models and terminology of batch processing and control (ANSI/ISA-88.00.01-1995), SP88 has produced a series of more formal data models and published them in the form of XML schemas in co-operation with the World Batch Forum (WBF, see http://www.wbf.org). Further examples can be found in discrete manufacturing (http://www.omac.org), in electronics industries (http://www.ipc.org), in condition monitoring, maintenance and asset management (http://www.mimosa.org, http://www.osacbm.org), etc. Many of the responsible standardisation organisations and industry groups co-operate to some extent. However, a single plant and data model can't be expected, merely due to the different application needs.

2.4 Requirements for future automation architectures

To summarise the discussion of the previous sections, there is a need for an integrated, flexible and low-cost system platform that supports design reuse and allows intelligent features to be easily implemented. To make this possible, the following general requirements should be considered:

- In order to guarantee sufficient semantic coherence and interoperability, applications should be based on the common concepts and reference architectures of the intended application domain.
- The platform should support reuse of design and software by individuals, companies and application domains. In addition to custom-made components and vendor specific libraries, it should be possible to use independently developed Commercial Off-the-Shelf (COTS) components.
- The use of common hardware and software standards is critical for reducing development costs and time to market, as well as for improving system reliability. Standards are also the way to easier integration with external systems and, thereby, larger component markets.
- New design tools should provide application-oriented programming languages with powerful interaction mechanisms and hide unnecessary implementation issues. For instance, this includes name-based addressing, location transparency and the freedom to define detailed system functions before deciding the hardware structure.
- An industrial system must be scalable from small stand-alone control and data-acquisition units to integrated applications with hundreds of nodes. The platform should apply horizontal peer-to-peer (P2P) communication instead of a centralised system architecture. Due to functional integration, it must be possible to combine different types of functions, such as real-time control, parameter and software modifications, historical data storage and reporting.
- Industrial applications must be maintained for periods longer than the life-time of their individual components. So, in addition to easy component integration, the platform must provide services for remote diagnostics and maintenance. Further, flexible plug&play mechanisms are needed for adding, updating and removing components without interruptions in system operation.
- Security and reliability are critical requirements for an industrial application. The system must be protected from unauthorised access and denial of service attacks. In addition, malicious and incompatible components must be detected before their installation. On the other hand, a mechanism is needed to protect the intellectual properties of component developers. In order to cope with the often harsh operating environments and communication uncertainties, control systems should have built-in mechanisms for detecting problems and management of various abnormal situations.

3. Concepts of a component-based control system

As explained in the previous chapter, there is an increasing demand for enhanced functionality, high quality and low cost in the control business. There are several approaches to tackle this challenge, for example the improvement of life-cycle processes, intelligent design and testing tools, and the use of commercial (or open-source) software and hardware instead of in-house development. Here, we are primarily interested in software reuse and believe that the principles of component-based software provide an appropriate technical basis. The purpose of this chapter is to outline the basic concepts and mechanisms of a future component-based automation platform. It is unlikely that this proposal will ever be implemented in practice as such. However, its goal is to give some ideas for future development in research, standardisation and industry-driven product development.

3.1 Component software in automation

In everyday life, a component is a replaceable and often commercially available part of a system that fulfils a function in some context. Most engineering disciplines introduce components as they mature. In software engineering, modularization, libraries, architectures and design patterns have been used to enhance design and software reuse. Recently, *component-based software* has emerged as an approach to similar goals.

There are a variety of definitions of what can be considered a software component. For example, Szyperski (1998) describes a *software component* 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. As there are other types of reuse (e.g. analysis and design), software components are binary units of independent production, acquisition and deployment.

Most programmers today use object-oriented languages and may think that objects and class libraries are components. However, there are differences even if a component often consists of one or more objects. For example, instead of implementation inheritance, composition of existing components by connecting their interfaces is the main mechanism of reuse in component software (Kuikka 1999).

An agreed interface definition and other design rules must exist to allow the assembly of applications from in-house and acquired components. In addition, a concrete hardware and software platform is needed to run the applications. A *component framework* is a software entity that supports components conforming to certain standards and allows instances of these components to be “plugged” into the framework. The framework

establishes an operating environment for the component instances and regulates their interactions (Szyperski 1998). So, frameworks are not specifications or patterns, but implemented software designed to be the basis for certain types of applications in certain domains (Fayad, Schmidt & Johnson 1999). Frameworks are applied by customising (by composing and subclassing) the framework's *hot spots* that are left open for modification.

We do not need too strict definitions of components and frameworks here. The most important criteria for us are: Application design through component composition; support for third-party components and open component markets and; existence of runtime platforms conforming to an agreed framework specification.

In general, the markets of commercial software components are not very well developed at the time. In the control domain, the concepts of component software have not been widely used as such. However, there is a clear need for standardised and reusable automation objects available for the engineering of measurement and control systems. This issue has been discussed within the standardisation organisations for several years. Also some examples of component features can be found. Firstly, the function block model has long been used in programmable logic controllers and distributed control systems. They are inherently reusable, and the standardisation of programming languages could, at least technically, allow commercial markets to develop. Recently, even the term automation component has been used in the context of large function blocks capable of controlling complex machines (e.g. a centrifuge with an embedded controller) instead of individual devices (e.g. a valve). Secondly, the FDT/DTM concept for managing heterogeneous fiedbus devices (see section 2.3) has a common framework application and independently developed driver components as its key feature.

As a third example, O^{3}NEIDA (http://www.oooneida.info) is a new international initiative to enable the flexible and open integration and re-configuration of embedded intelligence in industrial automation. O^{3}NEIDA plans to build and populate a publicly available repository of automation objects based on ongoing international standardisation (in particular on IEC 61499 function blocks). Manufacturers of production equipment can then use these components to implement intelligent machines and more complex systems (Figure 14). A key ingredient in the envisioned control business is a set of broker agents that enable all players to locate and retrieve the required intellectual property, and to deposit their products in searchable form accessible to other players.

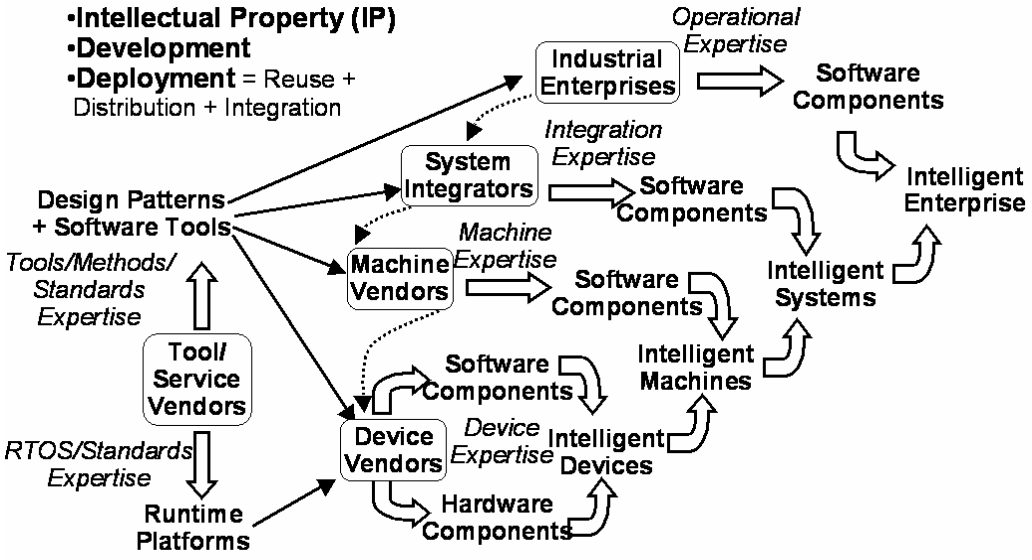


Figure 14. Integration of intelligent machines and software components in the value adding chain (source O[3]NEIDA).

As can be seen, there are signs of component orientation in industrial automation. However, they seem to be tightly coupled to the traditional function block models. In the rest of this chapter, we outline a slightly different concept that intentionally combines function blocks and the principles of general-purpose component software.

3.2 Equipment model

Knowing the current trends in computing power, data transmission and functional integration it is reasonable to suppose that future control systems are highly distributed. There is a mixture of computing resources varying from intelligent field devices and embedded controllers with limited capabilities to high-end process controllers and workstations. These computing nodes are connected by a wired or wireless communication network. Obviously, there is a need to combine different types of hardware and communication techniques. Logically however, they should all provide a homogenous computing environment to enable full exploitation of the component paradigm.

In the following, the term *control system* refers to a set of hardware and software designed and installed to serve a specific application, such as a production line in an industrial plant. As a distinction, *control products* are pieces of commercial hardware, software, services or combinations of them, provided by a manufacturer. When appropriately put together, control products from one or more suppliers form the basis for the customer's application, i.e. an *automation platform* in our terminology.

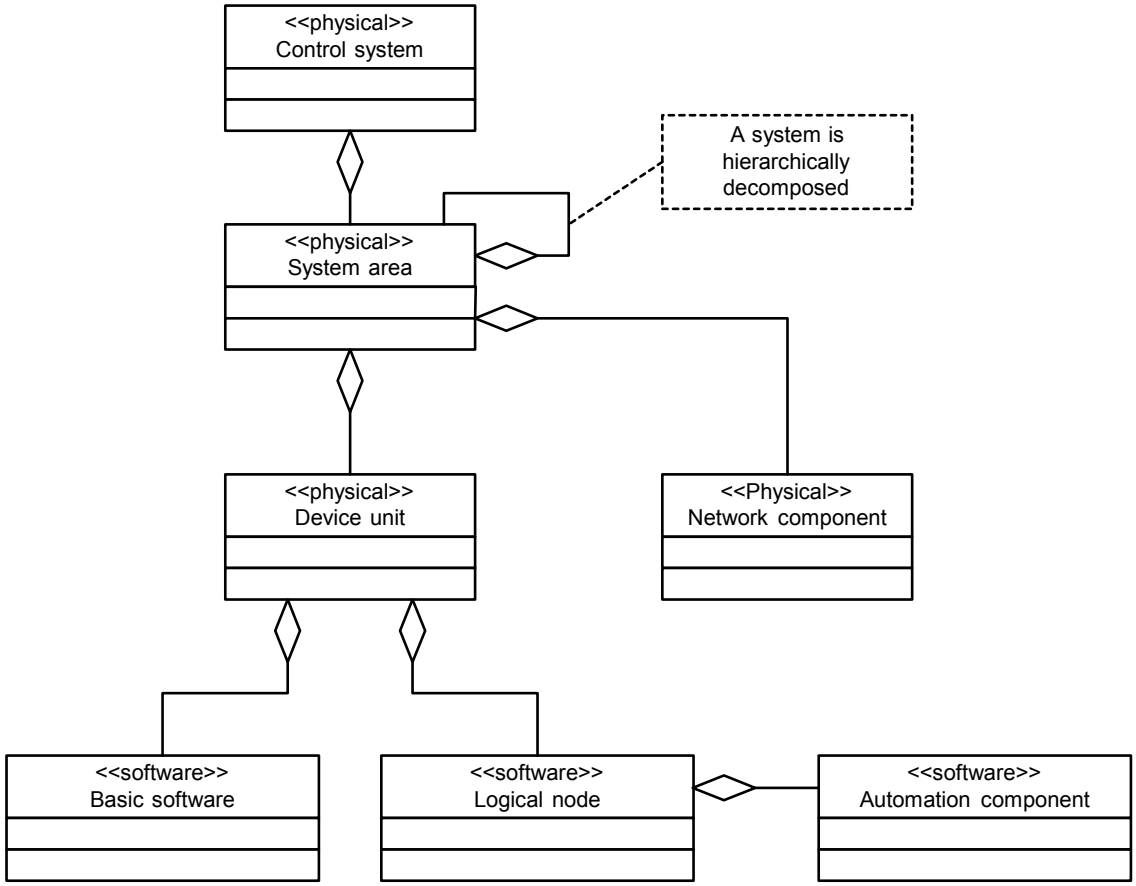


Figure 15. Physical components of a control system.

Figure 15 illustrates the concepts related to a *control system*. The total control system domain is hierarchically decomposed into *system areas* each consisting of a set of *device units* (control computers) and the necessary network components (e.g. switches and firewalls). The hardware and basic software (e.g. operating system) of a device unit, in turn, provide the computing resources for one or more *logical nodes* that finally act as the runtime environment of *automation components*.

In our typical scenario, a control system consists of a rather large number of computers. The purpose of the system itself and the subsystems defined by *system areas* is to provide well-defined domains for system management and protection. For example, a control engineer can be made responsible for a given system area, such as the "wet end" section in a paper machine control system. This makes it possible to report system-related events like device faults as alarms directly to the named person and to suppress events from other parts of the system. In addition, system areas can be used to specify the authorities of each control engineer to make modifications.

Normally, the boundaries of a control system are clear, and all the components running in the logical nodes belong to the same application. However, the allocation of

components to various nodes should be very flexible, which means that two applications might be overlapping in some part. In particular, this may be a future scenario in service-oriented and geographically distributed systems as discussed later in chapter 5.

3.3 Automation components

In our vision, *automation components* are self-contained pieces of software that can be combined and placed for execution in the *logical nodes* of a *control system*. To make this possible, there should be widely accepted standards for both the runtime environment and the engineering tools used for adapting and connecting components available from various suppliers. The following discusses, disregarding any technical details, the abstract functionality of automation components as seen by a future application designer.

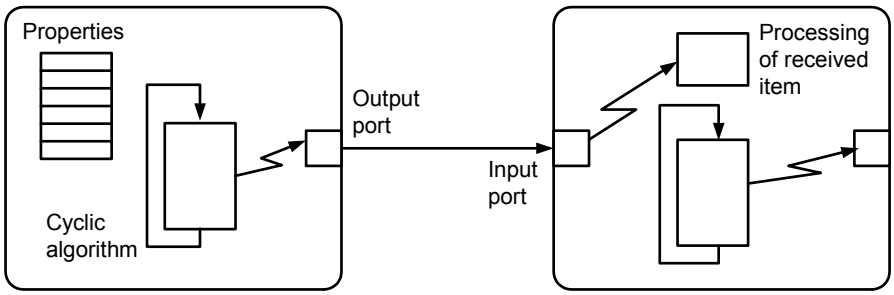


Figure 16. Basic elements of automation components: Ports, algorithms and properties.

Automation components are both reactive and proactive entities, whose behaviour is defined by their responses to external events and their internal activities. The external component interface is modelled as input and output *ports* that are "wired" to each other during design stage (Figure 16). An algorithm can be specified to process new messages received at an input port. The proactive behaviour of a component refers to internal algorithms that are cyclically executed. If no handler routine is attached to a port, the incoming message is just stored for later use. As a result, the model gives the designer the possibility to build both event-driven and cyclically running applications. An additional way for engineers to adapt library components and to control their behaviour is provided by component *properties*. Properties can also be used to store product information, runtime statistics, etc.

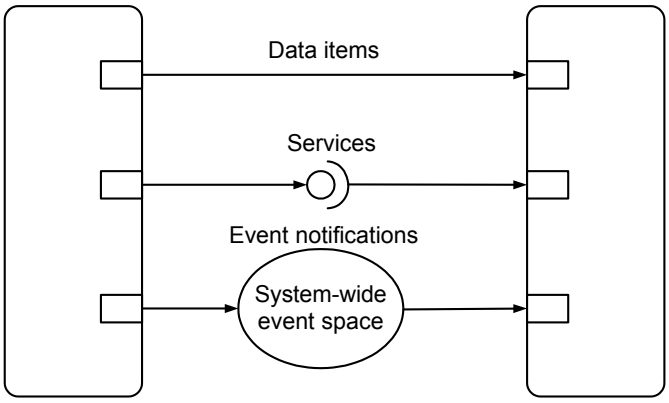


Figure 17. A minimal set of interaction mechanisms.

To build useful applications, the designer must have a way to make the components collaborate. In our approach, she or he forms connections between compatible input and output ports of component instances. In control engineering, connection endpoints are usually uniquely identified by their names, sometimes even by their "hardware" addresses. Another and a more dynamic approach is to establish component associations at runtime according to the required properties of data sources and services[1].

A number of different mechanisms for component interactions can be identified from software engineering and current practices of industrial automation. Section 3.4 below will discuss this issue in more detail. It is clear that different principles are needed for various types of functions. However, unnecessary complexity should be avoided. Therefore, Figure 17 shows a minimal set of interaction mechanisms that might be sufficient for the applications we are working with:

- Wiring *data input* and *output ports* of components is familiar to all control engineers that have designed function block networks for current DCS and PLC platforms. Normally, a data connection is an abstraction of copper wire, meaning that the value at the receiving input port is (practically) equal to the value of the data source at any point of time[2]. In our component model, however, the designer has an option to use the wire in an event-driven mode by attaching a handler routine to the input port.
- Services refer to methods provided by components through their *service output ports*. The *service input ports* of client components are wired to appropriate service outputs to accomplish the intended functionality. Different from data connections, a

1 This principle is often called 'contextual composition' and used, for example, in Enterprise JavaBeans (EJB), CORBA Component Model and Microsoft Transaction Server.

2 So, the way of thinking comes from electrical circuits.

service connection carries two-way data traffic, i.e. the client's request in one direction and the server's response in the other.

- The purpose of *event notifications* is to inform interested parties about events occurring within a control system. A fundamental difference from the two previous interaction mechanisms is that information exchange is anonymous, i.e. based on message content rather than on the identity of the sender. *Event output ports* act as *event sources* and issue *event notification messages* to a system-wide common space. *Event listeners*, in turn, subscribe to specific types of events by defining an *event filter*. This principle is often called *content-based publish/subscribe* in the literature.

There are various kinds of automation components as illustrated in Figure 18. Firstly, we make a distinction between *platform components* and *application components*. Platform components are pieces of "system software" running on top of the operating system, databases, and communication protocol stacks etc. Their purpose is to provide a runtime environment for the application. Depending on the case, the platform may consist of components for web access, directory and naming services, data communication, database access, security, user interfaces, etc.

Application component *container* is a special type of platform component. Its role is to provide a controlled environment for domain-specific *application components*. In process control for example, application components take care of sequential control tasks, regulatory control, interlocks, etc. Automation components are rather similar to the *function blocks* defined by current standards like IEC 61131-3 (2003) and IEC 61499-1 (2005). In other application areas, other types of functions, such as data acquisition, statistical analysis, and reporting can be found.

The *container* manages the life-cycle of application components and gives them access to the different platform services (e.g. communication) and to the resources of the underlying hardware. In fact, the container should even take care of the execution of and interactions between application components. The rationale behind this is safety. As applications are easily and rapidly constructed from commercial components, there is a risk of incompatible, misbehaving, deadlocking, or even malicious components. Therefore, the container should have tight control of their execution. Platform components, on the contrary, are usually acquired, tested and assembled by one platform manufacturer. So, more freedom (e.g. creation of internal threads) can be allowed to them.

In the control domain, applications are aggregated by combining library elements like *function blocks*. This leads to a hierarchical structure with the whole control application (e.g. "plant X") on top. Of course, the smallest pieces (leaves of the tree) must be

defined in another way. As shown in Figure 18, basic automation components can be implemented in general-purpose programming languages like Java and C++, or in the control-oriented languages of IEC 61131-3 (2003). A second approach is writing a proxy component that interfaces with external devices or software. The proxy pattern is particularly valuable as it allows reuse of existing control software in a new environment.

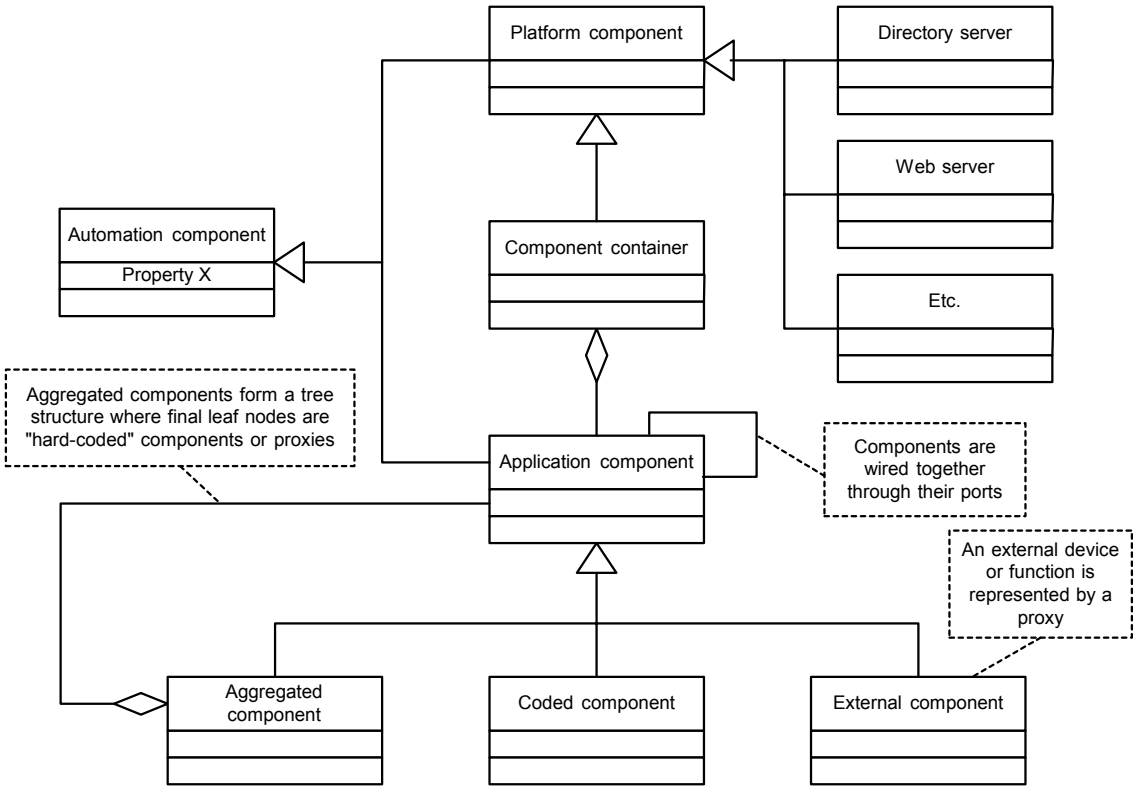


Figure 18. Subtypes of automation components.

Aggregated application components form a layered structure in the engineering environment as shown on the left side of Figure 19. In principle, only those ports of sub-components that are brought to the boundary of the parent are visible to the rest of the application. In practice however, this would mean that upper-level components should export a large number of ports. So, the designer needs a way to specify the visibility properties of a port.

At runtime, the components defined in the design tool are allocated to one or more *logical nodes* and their embedded *containers*. The distribution services provided by the platform take care of the necessary communication links between components. In some approaches, only the leaf components are downloaded to the target systems, while upper-level components are component repositories and serve the management of design information only. In our model, also parent components are real entities and have

their counterparts in the runtime environment, e.g. C1 in Figure 19. The component tree is maintained during runtime so that a component "knows" its children and vice versa. This is needed for system maintenance, for example. Shutting down C1 will automatically shut down also C1.1 and C1.2. In the other direction, C1.1 can report its health problems to its parent component.

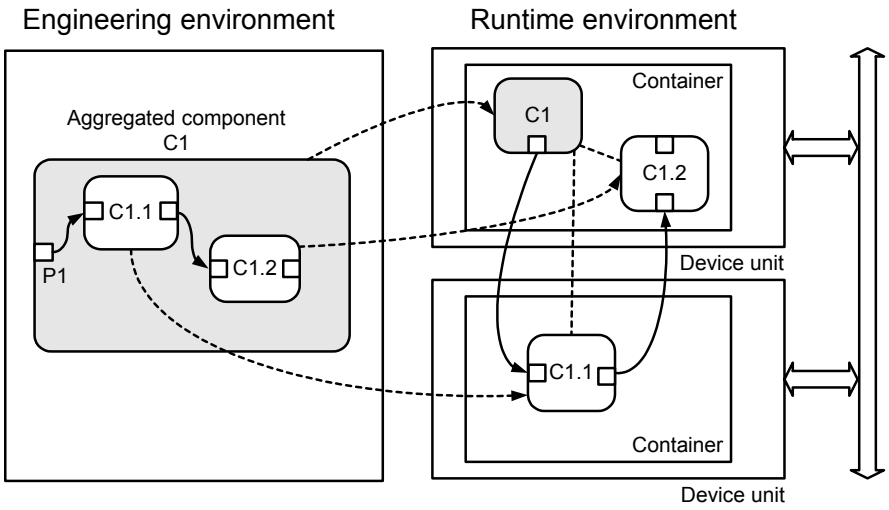
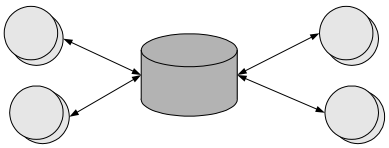


Figure 19. Parts of an aggregated component can be allocated to several device units.

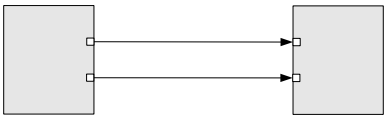
3.4 Interactions between automation components

This section describes the most important component interaction mechanisms and their intended uses in a distributed environment. The material in this section is generic in the sense that it does not make assumptions about application design or implementation. The issues discussed include general interaction patterns, i.e. continuous data distribution, event-based data distribution, services and notifications. The focus is on the functional views seen by the application designer.

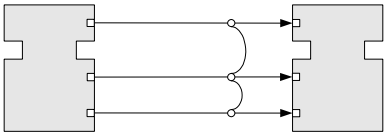
A number of interaction principles can be found in computer systems and automation technology. Figure 20 lists examples of the most common mechanisms. It is desirable to minimise the number of mechanisms. Yet it is not possible to make any strict recommendation here, since the need for communication mechanisms varies depending on design practices, standards and technologies used. The following sections give our suggestion that is limited to three basic mechanisms, data distribution (continuous and event driven), content-based event notifications and request-reply services. No existing product satisfies these requirements in full. This is discussed in section 4.3 when the implementation level mechanisms of individual middleware standards and products are described.



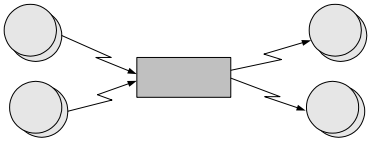
SHARED (DISTRIBUTED) MEMORY
- read and write operations on a common data repository
- typically a centralised database
- data can be distributed and replicated
- notifications can be sent about new or modified data



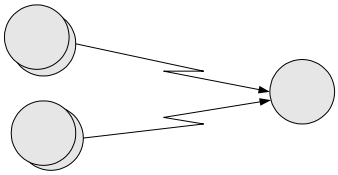
CONTINUOUS DATA DISTRIBUTION
- “wiring” from a named source to a named receiver
- receiver maintains an up-to-date copy of the data
- any data structure can be used
- data transfer can be done cyclically or when value changes
- compare to IEC 61131-3



EVENT DRIVEN DATA DISTRIBUTION
- “wiring” from a named source to a named receiver
- every message is a unique event
- received events can be stored or process immediately
- events can be used to control execution
- new data values can be associated with events
- compare to IEC 61499



MESSAGE BROKERING
- messages written to and read from a named channel or queue
- channel can provide services like message filtering
- blocking or non-blocking read operations



EVENT NOTIFICATION AND ACKNOWLEDGEMENT SERVICES
- message sources are not specified
- receiver subscribes to interesting messages with a filter
- local event triggers sending a message to all listeners
- data structure is an event message
- some messages must be acknowledged by application
- compare to OPC Alarms & Events

Figure 20. Examples of interaction mechanisms.

3.4.1 Continuous data distribution

This mechanism is familiar to anyone who has programmed PLCs using function blocks or designed intecrated circuits. A connection is made between the output and input ports of two blocks by “wiring” them. From application perspective, the signal value on both ends of the wire should always be the same. In the basic case, the wire is connected to one receiver, but the continuous data distribution mechanism should be able to handle any number of receivers.

When computer technology is used, an illusion of wiring is created by making the producer transmit a fresh data value frequently. Another way to implement the mechanism is to transmit new values only after the value has changed enough (exceeding an application specific threshold). The required frequency depends on the dynamics of the process, and the shortest cycles in modern systems are usually around 5–10 ms, or even shorter (in motion control for example). However, in process automation there are many tasks whose period is hundreds of milliseconds, several seconds or even minutes.

Cyclical data transfer is typically used for the continuous, real-time process control at the level of control algorithms. These have been designed to operate with a certain period, so the untimely delivery of data can result in unstable behaviour. Therefore, continuous data distribution places the most stringent requirements on latency and *jitter* (jitter is a measurement of how much the latency deviates from the mean latency on the average). However, a slightly longer (e.g. several ms) latency is usually acceptable in order to minimise the jitter. The delivery should be reliable, but missing a delivery is typically better than delaying everything with retransmissions.

3.4.2 Event-driven data distribution

The event-driven programming paradigm is another approach for developing automation applications (Lewis 2001), but it presents new requirements for reliability and synchronization. Data ports are wired as in continuous data distribution, but the transmitted message is an event instead of some signal value. An incoming event will trigger the execution of an algorithm that is associated with the input data port. This algorithm will usually process some input data; this might be carried by the event message, or it might be wired to other input ports.

Since the execution of the application is controlled by events, missing even one of them might have serious consequences (in the worst case, some critical task will never be executed since it will wait forever for an event that was lost). The mechanism should therefore guarantee the delivery of every event to all receivers. Even so, application designers should give some thought to the consequences of missing an event, because the network might be down for a long time.

3.4.3 Event notification and acknowledgement services

Although this mechanism is also used to transmit events, its nature is different from event-driven data distribution. The latter is used to control the normal execution of the application. In the case of event notifications, we are sending alarms and other notification messages to all interested receivers. These events are generated when the state of a component changes in some significant way.

Usually, the messages, such as alarms in current control systems, are only sent to inform an operator, or they might be stored in a history database. A notification might also trigger some operations, e.g. force the receiver to stop its normal operation and embark on a special course of action (such as open an emergency valve or stop the process safely). So, event notifications could be used to implement purely message-oriented applications.

The reliable delivery of messages is again required, since missing even one alarm can have serious consequences. We can see that there is a separate domain of events that are not involved in controlling the application's normal execution, but can nevertheless interfere with it. This design approach has proven its usefulness with automation systems, since they are large and complex entities, which must be prepared to cope with a great number of potential problems.

Another feature of event notifications is that they must in some situations be acknowledged from the application level. This will become complicated when there are several receivers. Many acknowledgement models exist; for example, it might be enough for one operator to acknowledge the event, or then a response from some or all of the receivers might be required.

The designer needs a mechanism for event notifications, because the event-driven data distribution mechanism is insufficient in two ways. First of all, there must be a flexible mechanism for subscribing to a large number of events from many different components. For example, we might ask for all temperature-related alarms from a certain process area that have priority "medium" or "high". Subscribers to the event-driven mechanism will specifically name the desired types of events. Secondly, if there is inadequate support for application level acknowledgements, the system developer will have to handle these by adding much functionality into his design of the automation system. For example, a considerable number of dedicated communication links might need to be created solely for carrying acknowledgements. If the acknowledgements are hacked into existing communications, there can be serious maintainability, reusability and reconfigurability problems.

Event notification services apply message-oriented communication and de-couple senders and receivers. Messages are sent to a mailbox or channel, so the sender does not need to know anything about the receivers. Receivers will in turn retrieve the messages from a channel, or they will filter them with some criteria.

3.4.4 Request-reply services

Request/reply can be used to obtain some data or to start some service. The reply might be asynchronous, in which case a call-back routine is invoked when the reply is received. A typical use of this would be to read or modify a parameter of a controller. This kind of a mechanism is easier to implement because of its simplicity and the looser real-time requirements. Reliability is desirable, though. The name or location of the service might not be known at compile time, so some of the server components might advertise their interface in some naming or directory service.

Remote read/write is a simple version of request-reply, since no other functionality is performed beyond writing or reading a parameter. Automation designers might make the assumption that the operation is always successful. In the future however, an exception handling mechanism should be required, and many implementations of request-reply in information systems already provide this.

3.5 Software architecture

The sections above focussed on the abstract and functional aspects of automation components. In the following, we introduce some general ideas about their software implementation. A more detailed discussion will be given in chapter 5 in the context of geographically distributed monitoring and control systems.

A distributed automation application is preferably developed from various components without constraints regarding on how they are allocated to several nodes or how the communication links between them are established. This is especially important, since it should be possible to deploy the same application for several clients and to modify or extend existing installations.

The goal is to let an organisation encapsulate its intellectual property into a library of reusable components. New deliveries are then simply a matter of selecting the necessary components and configuring them onto a standard platform. Depending on the needs of the application and the available hardware resources, it is possible to include only those parts of the platform that are really needed.

A component-based application needs an execution environment that provides services for the lifecycle management and interaction of components. Using existing components from a library and defining their distributed configuration and communication links should be as automatic and simple as possible. Every effort must be made to avoid “hard coding” that limits reusability.

As described in the previous sections, the *logical nodes* of our platform host two types of *automation components*, platform components and application components. Both should be able to run on top of common hardware and (real-time) operating systems, utilising general-purpose communication and data storage techniques. This leads to a layered architecture shown in Figure 21. On the level of system software, we only need a loose framework that enables combination of existing *platform components* like web servers and data distribution services. Their purpose is to provide a safe and controlled environment for *application components*.

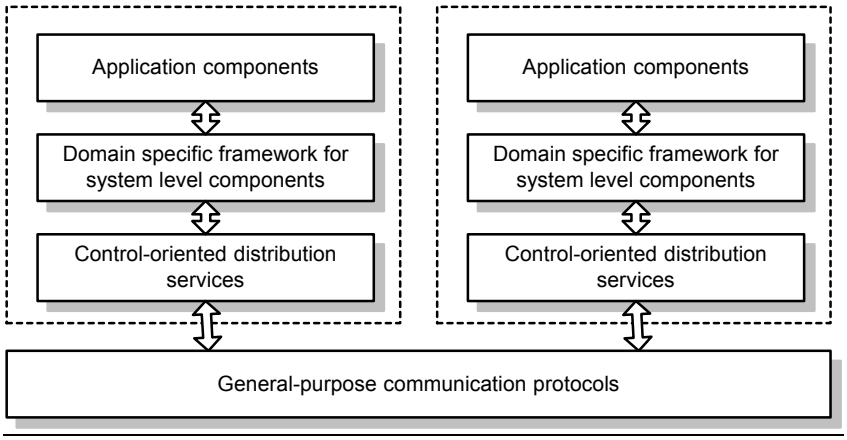


Figure 21. System levels: Basic software, platform and application.

As there are security and configuration control issues in industrial applications, it's a natural and common approach to put the components (written by various vendors) into a "sandbox" where they can be easily managed. In a *logical node* the component *containers* take this job (Figure 22). Firstly, the logical node itself is a container for platform components. The node, represented, for example by a platform component called "root", should provide a set of interfaces that serve the other, contained platform components. Accordingly, all platform components should implement certain interfaces to qualify as platform components. Secondly, it is possible to implement one or more inner containers for application components, for example in order to execute IEC 61131 function blocks as a "soft PLC". Again, there is an agreement about interfaces between the container and application components. However, it is probably different from the interfaces of platform components.

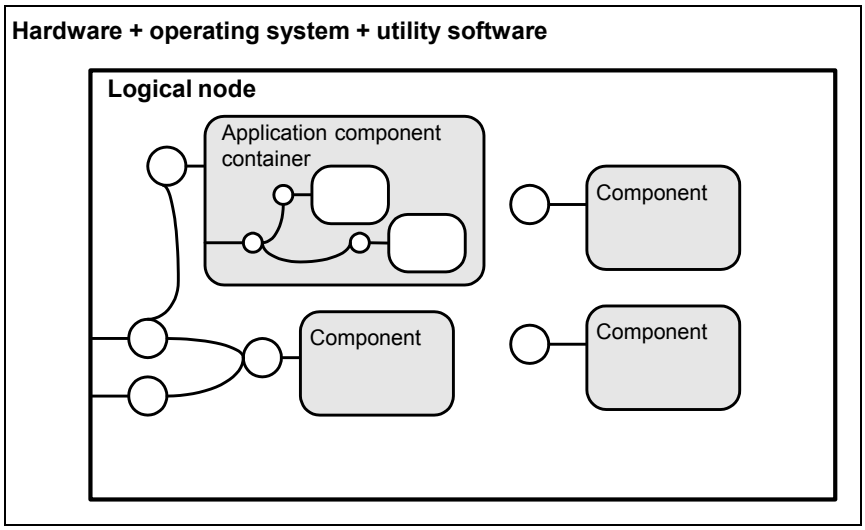


Figure 22. Component layers and interfaces in a logical node.

Containers and components should be independent of the implementation details of other components. For example, components should be portable between different container implementations. Therefore, interfaces are the key issue in the architecture. They form the central part of the contract between the container and components.

In our case, the components in a logical node typically represent system software and are developed and tested by an experienced vendor. Therefore, the platform components can be allowed to have some freedom, such as starting their own threads. The node should, however, be able to monitor and manage their execution, if this doesn't lead to excessive complexity and overhead.

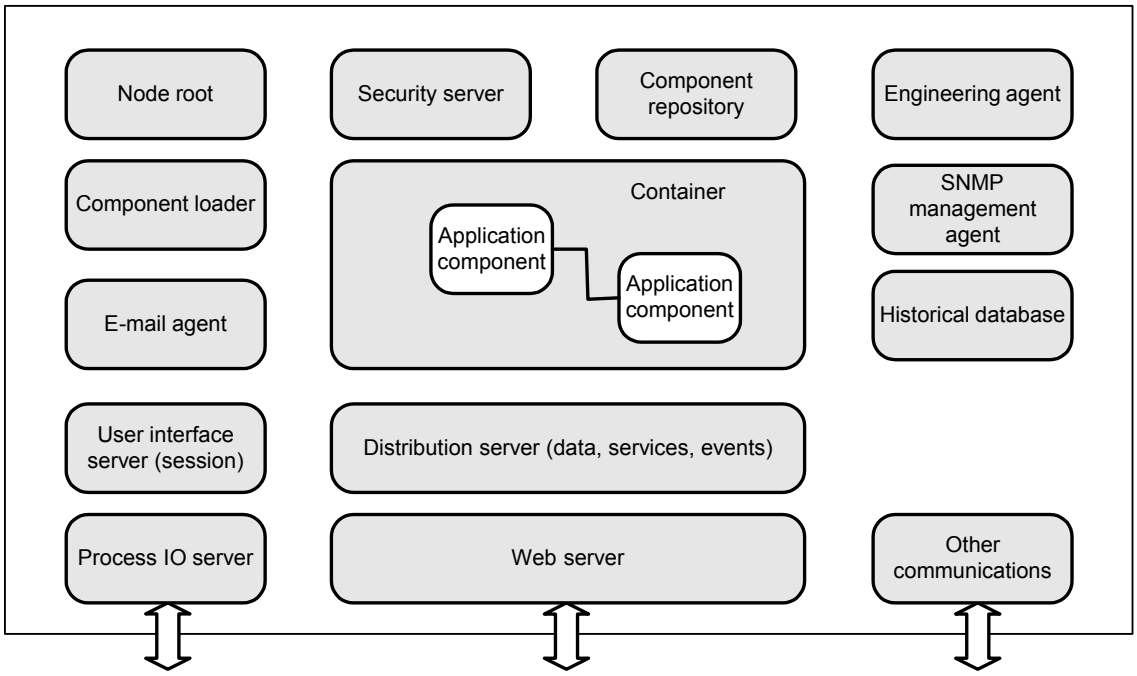


Figure 23. Examples of potential platform components in a logical node.

Figure 23 gives some ideas about platform components that could be installed in a logical node. The components and containers should provide following services:

- management of component life-cycle
  - installation, removal
  - starting, stopping, updating
  - propagation of configuration events to interested listeners
- searching the properties and optional services implemented by the container
- directory services (publish/register, find, update)
  - components
  - services provided by components (service ports)

- data published by components (data ports)
- event notifications (event ports)

- access to the underlying operating system and hardware and their properties
- receiving propagating events generated by components and the container itself.

To enable adaptation to application requirements and available hardware resources, the logical node itself should be modular and configurable. Some basic services are mandatory but others might be optional. In addition, some services may consist of a group of interfaces, only some of which need to be implemented. For example, Quality of Service attributes might be provided as an extension to the basic data distribution. This creates the need for some sort of negotiation mechanisms and compatibility checks during component installation and operation.

3.6 Mechanisms for configuration management

The discussion above represented interactions found in typical control applications. However, something is still needed to build robust and flexible distributed systems. For example, the system should have mechanisms for flexible addition and removal of devices and components (Plug & Play). To enable on-line configuration (which is a "must" in most industrial systems), replacement of an active component should happen without interrupting its functions. To do this in a safe way, the compatibility of the new version should be checked in advance. When downloaded into the target node, the component should first be started in parallel with the old one. Only after its internal state is synchronised with the old version, the replacement component can be connected on-line.

The platform should be fault-tolerant and support replication of data, as well as redundant devices and components in a way that is preferably transparent to the application. A robust system should adapt to varying service levels. For example, it should tolerate temporary unavailability of services and data sources (a typical problem during system start-up). Data buffering and message re-routing should be automatic in case of communication problems. All components should be aware of current Quality of Service in communication links etc. To make this operational, all components should report their own problems and react to problem reports they receive from others. Automatic garbage collection (such as leases in Jini) is a possible approach to manage misbehaving components that fail or quit without releasing their resources.

These requirements hold for both application and platform components. The detailed solutions are beyond the scope of this report. Yet, the demands for flexibility, robustness and dynamic system configuration have some implications for component interactions. Firstly, the wired connections between components' ports are not

inherently flexible since both ends of the wire are uniquely identified by their names. One way to relax these tight couplings is to define connections in terms of data or service contents instead of identity of the producer. A common approach is discovery, i.e. the use of directory services as a place for the producers to advertise themselves. These "lookup servers" can then be accessed by consumers to discover most suitable data sources and services on the basis of their properties (such as refresh rate and accuracy of a measurement). The concept allows several producers to coexist. A second approach called *content-based publish/subscribe* was already discussed above in the context of event notifications. This would mean that data ports are not connected by their names but by the name of the information they provide (e.g. "column top temperature"). Content-based communication allows several redundant producers to run in parallel. The consumer or distribution middleware must select one (first or best within a time window) from the received items.

Secondly, the platform should have services for components to report and receive notifications about changes in their states and properties (Figure 24). For example, a component might subscribe to events indicating communication faults or degraded quality. Or, a component blocked at start-up because of a missing service could be later activated by a notification telling that the service has become available. So, in addition to normal interactions, the platform should have dedicated notification services to propagate any changes in the configuration and state of local and remote nodes. Fortunately, event notification and acknowledgement services can be used as such for this.

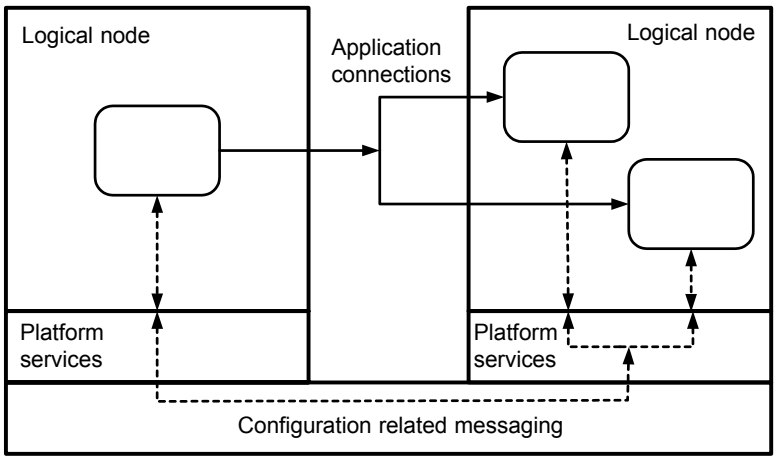


Figure 24. In addition to normal application interactions, the platform should have services that report and propagate notifications about changes in system configuration and status.

4. Local-area distribution services

Modern control systems are inherently distributed. Control functions are allocated to several controllers and servers within an industrial plant and connected by a communication network. Three network levels can usually be identified in today's plants: fieldbuses connecting intelligent sensors and actuators; control networks between programmable controllers and operators' workstations; and finally plant networks on top. In the future, wide-area networks will also be an essential part of the system.

Consequently, distribution services are a key ingredient in our architecture. The purpose of this chapter is to discuss the distribution of data, services and event notification between logical nodes and physical device nodes. We are primarily interested in the level of control networks. Fieldbuses can be integrated by proxy components as explained in section 3.3. Information systems and office applications, on the other hand, see the control system as a set of Internet enabled services. These issues will be discussed later in chapter 5.

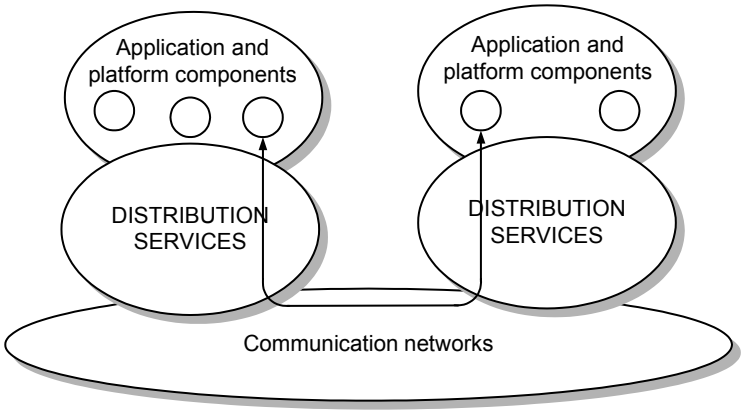


Figure 25. The role of distribution services is to provide location transparent connections between automation components.

For transparent interactions of components residing on different device nodes the control platform must provide distribution services. As scalability and reliability are key issues in control applications, these services can not be allocated to a single computer but distributed to each logical node. Figure 25 shows the position of distribution services in connecting automation components over a communication network. Their responsibilities include, for example

- advertising and searching available data sources, services and events
- establishing and maintaining links between client components
- buffering, routing and receiving messages

- monitoring the performance of communication links and generating notifications about significant events
- detection and management of exceptions and device failure.

In fact, distribution services are part of the platform and implemented as one or more *platform components*. Because of their complexity and performance requirements, well-known middleware solutions are a preferred basis for the development. The following two sections discuss application requirements and functional principles of distribution services. The third and last section in this chapter introduces a few new Ethernet-based middleware solutions being developed in the control domain.

4.1 Requirements for middleware over the application lifecycle

In software engineering requirements are typically elicited with use cases, which provide a solid foundation for development and validation (Fowler & Scott 1997). The approach is not perfectly applicable to evaluating middleware against the communication needs of process automation, so we developed "communication cases", which capture the main communication scenarios in a process automation application along with their functional and non-functional requirements. The following summarises the main categories of communication cases.

Process control

The communication cases in this category involve interaction mainly between controllers, sensors and actuators. They cover cyclic processing as well as one-shot operations such as parameter changes event-based communication. These typically require reliable transmission, but in the case of cyclic transmission such as sensor measurements for a PID control loop, it might be acceptable to rarely miss a value if the next one will be sent soon in any case. The frequency and regularity of the traffic varies greatly. Cyclic processing is typically frequent (usually the period is 10–1000 ms in process control). Event notifications and parameter changes are quite unpredictable and can cause a sudden load in unexpected or problematic situations.

Production control

The communication cases in this category are not directly involved with the plant-floor equipment control, but with receiving production-related information and setting parameters. Latency requirements are quite loose, usually depending on the patience of human users (e.g. < 2 s). However, the messages can be large and reliability is required.

User interfaces

This includes both the control room equipment and portable terminals that are used by operators. The nature of the data varies from cyclic trend information to event notifications and alarms. Reliability is required, but for a trend it is permissible to occasionally loose one value. Latency requirements are always similar, preferably less than 1 s, so that the performance is satisfactory for a human user. The communicating parties are the control room or terminal and the process controllers.

Management of production data

Great quantities of data must be stored in databases and reports must be generated from this. The communicating parties are the field devices, controllers and history-databases. Cyclically generated data as well as events are stored. Latencies are not a big issue as long as the system can handle great volumes of data, especially in exceptional situations. However, the resolution of timestamps should be on the order of 1 ms. In many applications, it is important that all data is eventually transferred and stored in a database. Reports should be generated in a few seconds.

System maintenance

This involves managing the lifecycle of a component-based application, for example loading, starting and updating components from a component server. Unfortunately, the middleware products we evaluated did not support this much.

The demands of the application's lifecycle place requirements on the automation platform. The deliverables of one phase should be the expected inputs of the next one. If conceptual and structural changes are needed, the original design idea may be lost, and extra work is needed in the implementation. The specification of the application's functionality should be done in terms of application-oriented communication mechanisms. Quality-of-Service (QoS) requirements and redundant data sources are also specified. The communication among components should be based on logical names that correspond to entities in the process, such as measurements and control signals.

The programmer should not be engaged in coding workarounds for technical obstacles. Only the automation platform directly uses middleware layer, which should be hidden from application components. The distribution services are used to establish the required communication paths, for example by automatically adding the service interface function blocks of IEC 61499 (Lewis 2001).

It is common that an application must be modified to satisfy newly discovered requirements. A good automation platform encapsulates technical details, so that application components can be modified or moved with minimal reconfiguration work. If communication links are based on logical names, it is possible to move existing components to different locations without making modifications.

However, since current middleware standards have different backgrounds, they also assume somewhat different ways of modelling the application. If developers use middleware that does not support their model, the implementers might be forced to "break" their paradigm. This is always error-prone and will complicate issues in later phases of the lifecycle.

4.2 Design principles of middleware services

Although designers are directly interested only in application level communication mechanisms and their QoS, the design principles of the underlying middleware services will have a considerable impact on these issues. Any middleware design will typically support some communication mechanisms better than others. It is also often the case that a middleware standard or product has not been designed especially for industrial use, so its design might not support the desired level of QoS even on powerful processors and networks.

Figure 25 above shows the place that is occupied by middleware (distribution services) in the system. Distribution services are seen as a black box, since application developers do not need to understand their internal architecture, only their service API. However, this internal architecture will have an impact on QoS, so the consequences of a few major design decisions are described in the next subsections.

4.2.1 Communication models

Section 3.4 described communication mechanisms from the point of view of the functionality and Quality of Service properties that designers are interested in. The discussion was intentionally de-coupled from any implementation principles or details, since these often vary among different standards and products. The requirements for middleware services should be expressed strictly from the design and application perspective. If standards and tools are in place, designers might even be altogether relieved from the responsibility of understanding implementation details. It certainly is a major reason for using middleware that the responsibility for understanding this complexity can be delegated to the middleware vendor. Since there are currently many

competing standards and products, users will benefit from some insight to implementation level principles. This section discusses the most salient issues and their implications.

In continuous data distribution, the actual transmission can be done cyclically or when the value changes significantly. The latter is preferable if the data structures are large, e.g. above 100 kB[3], but this requires guaranteed delivery. On the other hand, rapidly changing values of a few bytes are transmitted most efficiently on a cyclic basis. If the sending rate is high enough, the overall application performance can be improved by using a light-weight but unreliable communication mechanism (such as UDP), since the next value will arrive very soon if the network loses a transmission.

Another problem that middleware designers have solved in many ways is how multiple consumers of the same data are serviced. In a local-area context, it might be possible to broadcast or multicast the produced data, so that middleware services on all nodes determine if there is an interested consumer; otherwise, the sender can use point-to-point transmission to all registered consumers. The optimal solution depends on the application and the number of consumers, and some middleware products allow the application programmer to select the implementation mechanism (RTI 2002). It is also possible that messages are not transmitted straight from the producer to the receiver, if some special node is responsible for keeping track of the producers and consumers and routing the messages. However, this approach is inefficient and not very fault-tolerant.

If event-based control is used as defined in the IEC 61499 (Lewis 2001), it is important that all of the input data that accompanies an event is coherent, e.g. from the same sweep of an earlier algorithm. Whichever way the event-driven data distribution mechanism is implemented, it should guarantee the coherence of the data. Existing solutions often do not solve the problem satisfactorily, since they might force the developer to bundle the related input data tightly into some data structure. This causes problems when applications are reconfigured.

4.2.2 Centralised vs. decentralised

There are two common middleware architectures: peer-to-peer and centralised. In the centralised architecture, a server keeps track of all the communication links and routes the data and messages to the correct receivers. This approach is neither scalable nor efficient, because the server will easily become a bottleneck and there is no direct

3 In this case only the changed parts of the data structure might be transferred.

communication between the senders and receivers. Fault-tolerance is also a problem, because a failure of the server will bring down the whole system. The peer-to-peer solution is based on having middleware services on every node. These maintain up-to-date information of communicating parties on other nodes and are thus able to directly route messages to the node hosting the receiver. A failure of one node will only affect communication links with an endpoint on that node. The pure centralised solution is considered unacceptable for automation systems. The peer-to-peer architecture is in many ways ideal, but partially centralised solutions can also be considered.

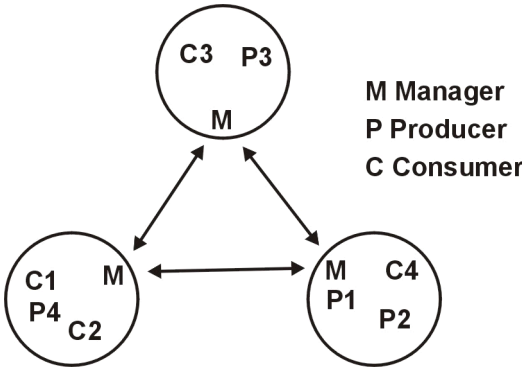


Figure 26. A generic peer-to-peer architecture.

Figure 26 shows an example of a decentralised peer-to-peer architecture. On each node there is some kind of manager or other middleware entity, which keeps track of the producers and consumers on each node. For this purpose the managers maintain local databases and communicate any changes in local producers and consumers by exchanging information with the other managers. Note that consumers are completely de-coupled from producers, since they only receive data from the local manager. This can be contrasted to centralised client-server solutions, or to the channel architecture in Figure 27. A representative example of the latter is the CORBA Notification Service, where there is a channel residing on one node and all communicating parties (i.e. the producers and consumers of data) can connect to it using proxies (OMG 2002). The two horizontal lines in the figure show the node boundaries between the server that hosts the channel and the other nodes on which the producers and consumers are running.

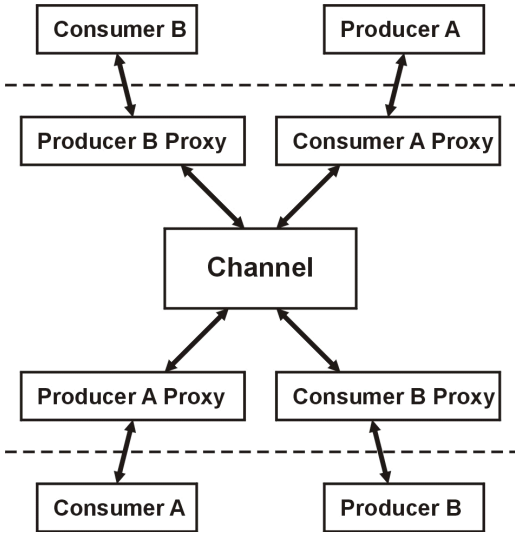


Figure 27. Typical centralised channel architecture.

4.2.3 Dynamic vs. static communication link formation

Middleware design will be considerably easier if application designers can be expected to specify all the communication links before the application is started. However, sometimes it is necessary for a component to form links to others at runtime, for example if something is fixed or added without shutting down the entire automation application. The trend towards flexible reconfiguration also places greater requirements on automatic communication link formation. From the perspective of middleware, this means that using a logical name, the middleware must be able to form a link to the other communicating party, although the name should not contain any information about the network address of that party. Some products do not support logical names and this will limit the reconfigurability of the entire application.

An alternative to name-based identification of communication link end-points are content-based communication links, where data is published to some "signal space" and consumers can subscribe to data based on certain filters or expressions regarding the properties of data. If very dynamic service discovery is desired, lookup mechanisms such as Jini can be used (Coulouris et al. 2001, pp. 372–374), but this already assumes a radically different approach to application design.

The middleware's strategy for dynamic communication link formation will depend on whether or not a centralised architecture is chosen. In a peer-to-peer design, each node must maintain databases about the sending and receiving entities in the distributed system. The middleware must therefore have some protocol by which the middleware services on each node can communicate this data in real time but at a lower priority than

the more time-critical communications such as the data used control loops. In a centralised solution, there is usually some name service at a well-known location, from which it is possible to retrieve information about the signals, events and services in the system. There might also be some centralised channel, so that senders and receivers simply connect to that and can therefore be de-coupled from each other.

4.3 Distribution middleware solutions

As pointed out before, functional integration combined with physical distribution is a major trend in industrial automation. There is a need for both vertical and horizontal information exchange, including local area networks and remote access over the public Internet. Currently, the distribution solutions are vendor-specific, and while numerous standardisation efforts are going on at fieldbus level and above, no convergence can be seen so far. This section discusses a few of the emerging middleware solutions in this area. The emphasis is not on fieldbuses but rather on the level of control networks, which seems to be a feasible place for Ethernet technologies.

Following the success of Ethernet and IP-based protocols, it is often suggested that Ethernet might be the only solution needed in future control systems ranging from field devices to manufacturing operations. The strength of Ethernet and IP is based on high throughput, low cost of standardised hardware and software, and on the possibility to use several application protocols at the same time. It looks attractive to mix real-time process control and support of remote maintenance on the same cable. However, there is a multitude of open questions to be solved:

- Physical compatibility and common communication protocols are only a part of the solution. For interoperability, the communication partners must have common concepts and protocols on the application layer also. Initially, the application layer was not in the focus of protocol development. Now, many specifications in the control domain include application concepts, but there are many suggestions for different purposes instead of one. Different application layers make the emerging Ethernet solutions incompatible.
- Even if Ethernet has high communication speed, it is inherently non-deterministic, which is in contradiction with the requirements of real-time control. This is, however, not a big problem anymore for most control applications running on switched networks.
- Current Ethernet networks utilise switches and star topology to balance communication load in different network segments. This is good for response times, but adds hardware and cabling costs. The harsh environments often encountered in industry limit the use of standard cables and connectors. So, industry grade

equipment is needed. Even if standard techniques are used, it seems unreasonable to extend Ethernet to all field devices.

- Normally, the Ethernet cable can't supply power to network nodes. This is an issue on the field level in particular, and new cable and connector types have been suggested. Explosive atmospheres and the requirements for intrinsic safety impose further limitations on power distribution.
- Security problems are the dark side of connecting control systems to corporate networks and external service providers. The possibility to access a web server embedded in a process controller for diagnostic and maintenance purposes creates a threat to confidentiality and system integrity. While solutions exist, there is no single approach. In particular, many of the standard techniques can not be applied to control systems with limited processing power.

There are numerous industrial consortia and standardisation groups in various application areas trying to overcome these obstacles and reach an agreement about a distributed control architecture. The following list of development efforts hopefully demonstrates the fuzziness of the situation:

- Ethernet/IP was introduced by the Open DeviceNet Vendor Association (ODVA, http://www.odva.org) towards the end of 2000. IP stands here for "industrial protocol". EtherNet/IP takes advantage of standard Ethernet communication and extends the stack for industrial purposes. Like ControlNet and DeviceNet, EtherNet/IP uses the Common Industrial Protocol (CIP) at the application and user layers. Different from many other techniques, also application objects (e.g. analogue input points) and profiles for widely used devices in discrete manufacturing (e.g. pneumatic valves and AC drives) have been defined.
- Modbus/TCP is a derivative of the old Modbus protocol. It is currently developed by the Modbus-IDA group (http://www.modbus-ida.org), which was formed as Modbus joined IDA (Interface for Distributed Automation). In addition to managing the Modbus protocol, Modbus-IDA targets at more comprehensive interfaces and architectures for distributed automation. The previous work of IDA included an application model based on the function blocks of IEC 61499 and a distribution approach called Real-Time Publish/Subscribe (RTPS). RTPS will be shortly described below in section 4.3.2.
- HSE (High Speed Ethernet, see http://www.fieldbus.org) is based on standard (100 Mbit/s, switched) Ethernet technology. It operates as a backbone of one or more underlying Foundation Fieldbus segments using the older H1 protocol. Bridges are used to connect the considerably slower H1 buses to the HSE network.
- PROFInet has its background in PROFIBUS (http://www.profibus.com). However, it is not a PROFIBUS ported to Ethernet, but a more comprehensive control

architecture based on IEC 61499 function blocks and component-thinking. A short introduction is given below in section 4.3.3.

- OLE for Process Control (OPC, http://www.opcfoundation.org) is the current de-facto standard for linking heterogeneous control products. It consists of a set of implementation-oriented specifications based on Microsoft's DCOM technology. Section 4.3.4 includes a description of some new developments in OPC.
- The Common Object Request Broker Architecture (CORBA), developed by the Object Management Group (OMG, http://www.omg.org), is a vendor-independent infrastructure that allows the integration of distributed object-based applications. CORBA consists of a rather large number of specifications and services, mainly intended for general-purpose information processing systems. However, CORBA has been considered as a control platform also, and some newer developments in the CORBA technology suite are, in fact, directed to distributed control applications. These issues are shortly discussed in section 4.3.1 below.
- The Association Connecting Electronics Industries (IPC, http://www.ipc.org) is an example from another type of industry. IPC develops standards for printed circuit board (PCB) fabrication and assembly. The CAMX standard series defines computer frameworks to move data through the web. In particular, IPC-2501 (Definition for Web-Based Exchange of XML Data) establishes the semantics and an XML based syntax for shop floor communication between electronic assembly equipment. This standard outlines a purely message-oriented communication architecture whereby a single logical (centralised) middleware server, the Message Broker, exchanges messages among clients in a domain.

Many of the emerging Ethernet networks in automation are based on a previous fieldbus technology and use the same application layer, i.e. the same data types, objects and functions. PROFInet is an exception; it is not a PROFIBUS sitting on Ethernet (Berge 2004). Some techniques, such as CORBA and OPC, have their background in information technology with limited concepts on the application level. A few (IDA for example) are more comprehensive suggestions to combine novel information technology and automation.

To be more specific, the following sections give a short introduction to CORBA, RTPS, PROFInet and OPC[4]. The discussion includes application level concepts and interaction mechanisms (data distributions, event notifications and services), as well as performance characteristics (real-time issues, security) and availability of commercial products.

4 These four techniques were considered particularly interesting for our research and were therefore studied in more detail than the others.

4.3.1 CORBA

The Common Object Request Broker Architecture (CORBA) is intended for creating, distributing and managing distributed objects in a network. CORBA is best known for its Remote Method Invocation (RMI), but many other specifications add to this basic functionality. The specifications are developed by the Object Management Group (OMG, http://www.omg.org).

CORBA integration requires CORBA middleware in both ends of the connection. All applications need at least an Object Request Broker (ORB). This provides an API for basic remote method invocation. Stubs and skeletons hide the actual request and reply processing over the network. The ORBs themselves communicate via an Inter ORB Protocol, which must be implemented for the transport technology used (e.g. IIOP for TCP/IP). The more advanced CORBA Services (e.g. Notification Service) use the ORB and the basic RMI to provide higher-level communication functionality. Applications access these functions through various service specific APIs.

CORBA's suitability for control systems can be evaluated against our classification of desirable interaction mechanisms (see section 3.4). As CORBA is basically a collection of specifications, we need to find the appropriate ones. A remaining task is to find a suitable CORBA implementation supporting the selected services. The following discusses shortly how various CORBA specifications might contribute to some of our communication scenarios.

As described in section 3.4, the request-reply mechanism is used for accessing a service provided by some other component or for performing a remote read or write action. Remote method invocation, which is at the heart of CORBA, matches well for this purpose. It was the first functionality specified, and many other CORBA services use it as the basic mechanism. It is an object-to-object, two-way synchronous remote method invocation service, which involves getting a reference (stub) to a remote object, calling the stub's method and waiting until the method returns with related return values. References to remote objects can be retrieved from naming services, if one is needed.

Continuous data distribution is a distributed version of the traditional "signal wiring" concept, familiar to the majority of automation designers. CORBA's basic RMI service has some properties reducing its applicability for continuous data distribution. Continuous data distribution does not require return values, and blocking a call made by a real time control application is seldom acceptable. The ORB's Dynamic Invocation Interface (DDI) allows one-way method calls, which do not involve any return values. This could be used for sending signal values periodically to a target node. Just like the plain CORBA RMI, DDI uses object references to the target object. In one-to-many

data distribution, this causes major reference and name management issues. Although the CORBA Naming Service may provide some help in locating object references by name, the application still has to manage and communicate with low-level software constructs.

The new CORBA Data Distribution Service (DDS) attempts to standardise a high-level API for a real-time publish subscribe. It includes the Data Centric Publish Subscribe protocol (DCPS), which encapsulates the complexities of distributed naming and reference management and network layers behind a developer-friendly interface. DCPS interfaces provide the developer with methods for naming signals ("topics"), creating communication objects, configuring communication flows (e.g. publish, subscribe) and for the actual data transfer (e.g. send_issue, receive_issue). The links and details of the transmissions are processed behind the scene by the middleware implementation. The full specification version 1.0 is available from OMG (OMG 2005).

Compared to some other CORBA specifications with tens or even hundreds of implementations, the new DDS is implemented by just a small group of companies (RTI 2004, OIS 2005, Thales 2005). The US company Real-Time Innovations (RTI) has been the driving force behind DDS, and its latest NDDS middleware version implements the specification. RTI was also involved in developing the IDA Group's RTPS protocol.

For event-driven communication, the CORBA Messaging Specification includes a model for Asynchronous Messaging Invocation (AMI). The AMI call-back model can be used for event-based applications that use asynchronous calls (OCI 2002). Such calls relieve the client from having to wait for a call to return, which may sometimes take an unpredictable time. The answer is propagated back to a separate reply handler. This has code, which is invoked by the client ORB when the reply is received. The server needs not to be aware that AMI is used; the client ORB will notice asynchronous requests and map them to the appropriate reply handler.

In event-based applications, an event consumer registers at an event source, after which it receives events via a call-back method. The supplier should not need to keep track of consumers and how they are contacted. The CORBA Event Service provides a convenient way of developing event notification systems (Henning & Vinoski 1999). Its flexibility is achieved by using an event channel as shown in Figure 28 (OMG 2002). There can be many suppliers and many consumers, and no direct link is made between them. For example, consumers do not need to know the object references of the suppliers and suppliers do not need to keep track of consumers. This leads to robust code for the client and server applications, since it does not matter if some new object will take the role of a supplier or if new consumers register. The mechanism is similar to the real-time publish-subscribe protocol of IDA (see section 4.3.2).

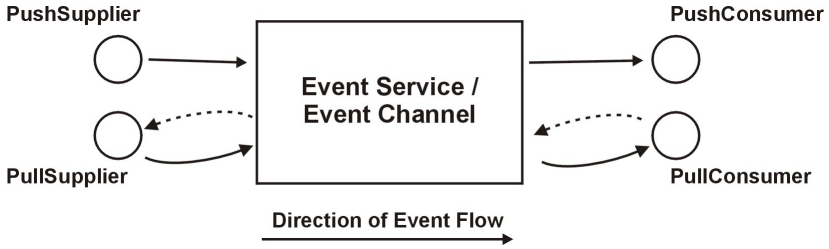


Figure 28. Event delivery in the CORBA Event Service.

Figure 28 illustrates different models for event delivery. A push supplier actively gives events to the channel, while the channel is responsible for obtaining events from pull suppliers. A pull-consumer will explicitly request events from the channel while the channel delivers events to push-consumers. Consumers and suppliers with different models can be attached to a single event channel.

The event channel does not try to interpret the event data in any way. This means that all events from any supplier to the event channel are delivered to all consumers attached to the same channel. The channel does no event filtering. The CORBA Notification Services, however, provide advanced filtering mechanisms, which can be used to limit the incoming message flow to the ones the consumer is interested in.

4.3.2 Real-Time Publish Subscribe

IDA (Interface for Distributed Automation) (IDA 2001) has proposed a new way of developing and managing automation applications that is very similar to IEC 61499 (Lewis 2001). The communication aspects of the proposal rely on the RTPS (Real-Time Publish Subscribe) protocol, which is one middleware solution.

RTPS is only a communication protocol, so the implementers have much freedom in designing the architecture. This description concentrates on the standardised communication mechanisms; architectural decisions were discussed in section 4.2.3. A more detailed analysis of RTPS and its real-time performance can be found in (Sierla 2003).

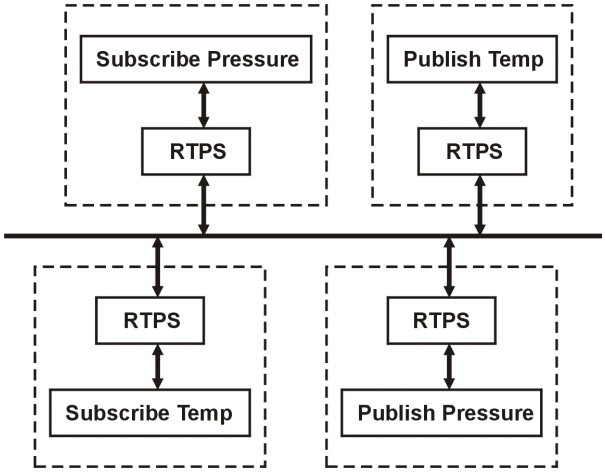


Figure 29. RTPS as an example of a peer-to-peer architecture.

RTPS supports continuous data distribution, event notifications and request-replies. With every mechanism, the programmer is given an opportunity to set parameters that control the Quality of Service (QoS). In this way, reliability, memory usage and determinism can be optimised according to the needs of the application.

Figure 29 illustrates the publish-subscribe mechanism of RTPS. For continuous data distribution, the "best-effort" version of this mechanism is the most appropriate. It has been designed for minimal latency, so messages that are lost by the network are not retransmitted. This is usually the best approach for transmitting periodically or frequently sent measurement and control signals.

The flexibility of the publish-subscribe mechanism derives from the fact that publications and subscriptions do not need to know anything about each other. A signal is identified by a topic name, such as "temp" or "pressure" in the figure. A producer of measurement data needs to create a publication for a topic, and after this individual measurement values can be sent as issues by making one method call. A consumer of data can create a subscription as long as it knows the name of the topic it is interested in. The middleware will then interrupt the consumer application every time it receives new data. Therefore, producers of data do not need to know anything about the number and location of possible consumers, because the middleware maintains the necessary databases. Reconfiguring and updating the application becomes easier, since names are independent of network addresses.

The request-reply mechanism of RTPS resembles the remote procedure call that is widely used in object-oriented programming. The middleware makes services available by name (clients do not need to know the location of the server), and there is support for QoS and redundant servers.

The reliable version of publish-subscribe is well suited for transmitting event messages. The implementation uses retransmissions and acknowledgements very much like TCP, but better QoS support is provided. These messages might belong to a sequence of commands, so they must arrive in order and none may be lost. However, alarms can also be transmitted, and the pattern subscription's of RTPS enable a programmer to subscribe to all topics whose name fields satisfy the specified filter criteria. Reliable publish-subscribe also satisfies our requirements for event-driven data distribution reasonably well. However, there is no support for operator-level acknowledgements of events and alarms.

The RTPS protocol was introduced a few years ago. Since then IDA and the Modbus Organization have merged to form Modbus-IDA. Towards the end of 2004, the RTPS and the Modbus protocol were published by IEC as the pre-standard IEC/PAS 62030 (2004). Recent developments on the Modbus-IDA website (http://www.modbus-ida.org) indicate that its attention has shifted to Modbus (IDA 2004). Further development and commercialisation of RTPS is carried on actively by RTI (RTI 2004), whose NDDS implementation is currently the only commercially available product.

4.3.3 PROFInet

PROFInet is an effort of the PROFIBUS User Organization (PNO) and PROFIBUS International (PI, see http://profibus.com) to define a standardised architecture for distributed control systems. It integrates existing fieldbus systems (in particular PROFIBUS) and PLC applications, etc. (Figure 30). Regardless its background in a specific fieldbus technology, PROFInet is not a fieldbus ported to Ethernet, but a more comprehensive approach to control system integration.

One goal of PROFInet is to secure long-term investments by allowing the easy integration of existing installations and the reuse of application software (PNO 2002). Single devices and more complex machines are modelled as *technological modules*, i.e. as intelligent and reusable production units consisting of mechanical and electrical parts, and control software. The automation functionality of each module is encapsulated as a *PROFInet component* in the control system.

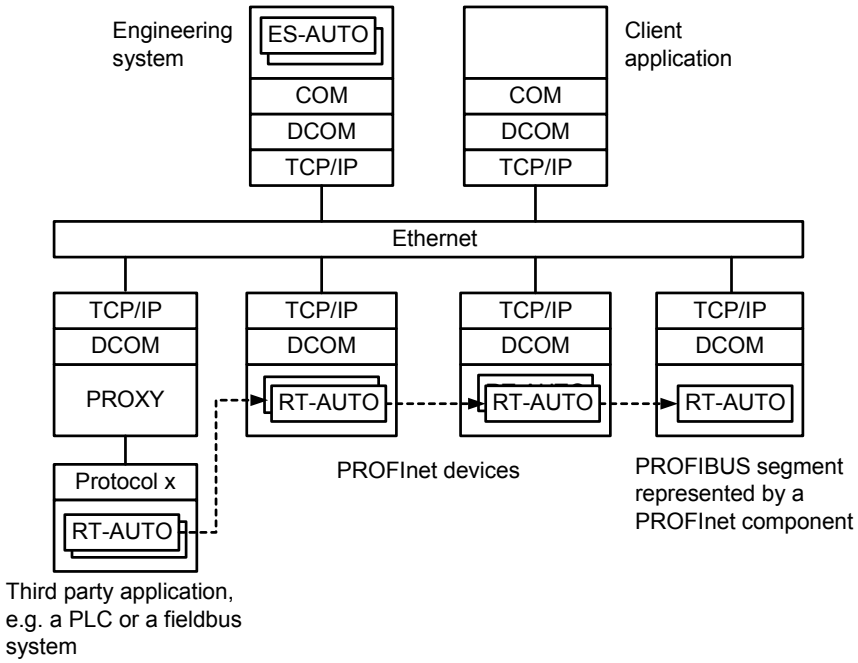


Figure 30. A PROFInet system integrates various types of controllers and fieldbus systems by interconnecting runtime components (RT-Auto) over the Ethernet. Applications are configured by connecting components' (ES-Auto) interface variables in a manufacturer independent design tool.

The focus of PROFInet is on the interface of components leaving the implementation to the manufacturer. So, there is no requirement for the actual control software to run on the PROFInet device as far as the component interface looks the same to the rest of the system. In particular, the use of PROFInet components as proxies to external PLC applications or fiedbus systems is an important migration strategy for existing installations.

Programming of control software for the production devices is performed as before with vendor-specific tools. For integration in the PROFInet system, the software modules are encapsulated as PROFInet components (ES-Auto) and imported in XML format into the library of a vendor-independent engineering tool. This is usually done by the manufacturer of the machine or plant.

The application can then be assembled without programming by graphically interconnecting the inputs and outputs of the components. The function blocks defined in IEC 61499-1 (2005) have been used, with some differences, as a reference model for the application level (PNO 2003). So, PROFInet is basically focussed on continuous data distribution (cyclic or triggered by change) between the input and output variables of PROFInet components. Also generation of alarms is supported. Event inputs and

outputs included in the function block model of IEC 61499 are planned to be available in a future PROFInet version.

Finally, the program code, connections, and other configuration data are downloaded into the PROFInet devices for execution. Communication links are established automatically on the basis of this information (e.g. connections and their Quality of Service attributes defined by the control engineer).

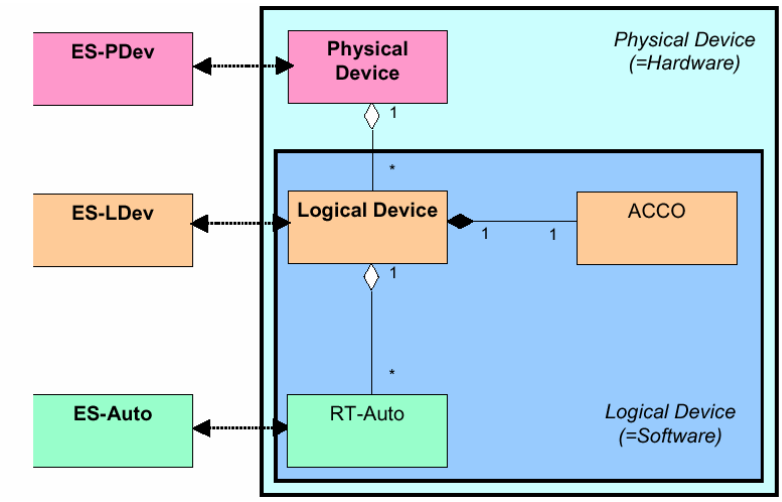


Figure 31. Object model of the PROFInet runtime environment (PNO 2003).

A PROFInet system consists of *physical devices*, each of them hosting one or more *logical devices* (Figure 31). A logical device, in turn, is a container for the runtime automation objects (RT-Auto). The objects in the runtime environment correspond directly to the objects (e.g. ES-Auto) in the engineering environment. The Active Control Connection Object (ACCO) establishes and monitors the configured connections between the devices (PNO 2002).

The DCOM Wire Protocol is used for communication between the PROFInet components. PROFInet components are thus COM objects tailored for automation applications. This has made the integration of PROFInet with OPC rather easy. Additionally, optimised protocols, utilising both software techniques (soft real-time in PROFInet version 2) and dedicated hardware (version 3), are available for applications with real-time requirements.

PROFInet is seems to be actively developed by several European manufacturers. According to the developers, tens of products and services are available from more than 20 vendors. PROFInet supports reuse of function blocks as automation components. Different versions support applications from real-time control (e.g. robot arms and cars)

to less critical process control and data acquisition. The use of DCOM does not mean that PROFInet it totally confined to Microsoft's operating systems (PNO 2003). In particular for the runtime environment, also independent implementations of DCOM are available. With regard to the engineering systems however, PROFInet is primarily based on Microsoft technologies and operating systems.

4.3.4 OLE for Process Control

The aim and motivation of the OPC (OLE for Process Control) family of specifications is to enable open connectivity between software components in an enterprise, e.g. between PLC, DCS and SCADA (Supervisory Control And Data Acquisition) systems, databases and business management software. Some of the specifications, especially OPC DA (Data Access) and OPC A&E (Alarms and Events) have reached a quite wide acceptance, which makes them attractive as a communication middleware.

Basic communication paradigm in OPC DA and OPC XML DA is a client-server-based request/reply interaction. Request/Reply style operations can be used to read and write data between the client and the server. Synchronous read operation is a good example of this kind of behaviour.

OPC DX (Data eXchange) is quite new and not as widely used as OPC DA. The aim of the DX specification is to standardise the horizontal data exchange between various data sources. Thus far, the way to configure the data exchange has been specific for each OPC Client. Data is transferred between several OPC DA servers. This is logically quite close to ideas of continuous data distribution, although in practise data is sent using either event-based client server communication or using synchronous functions. As Figure 32 shows, the DX connection is to one direction only. Bi-directional communication requires that both participants conform to the DX interfaces. Connections are thus distributed. Furthermore DX specification defines both COM-binding and Web services binding. Configuration of connections can be done using Web services interface and data can be transferred using XML-DA interfaces.

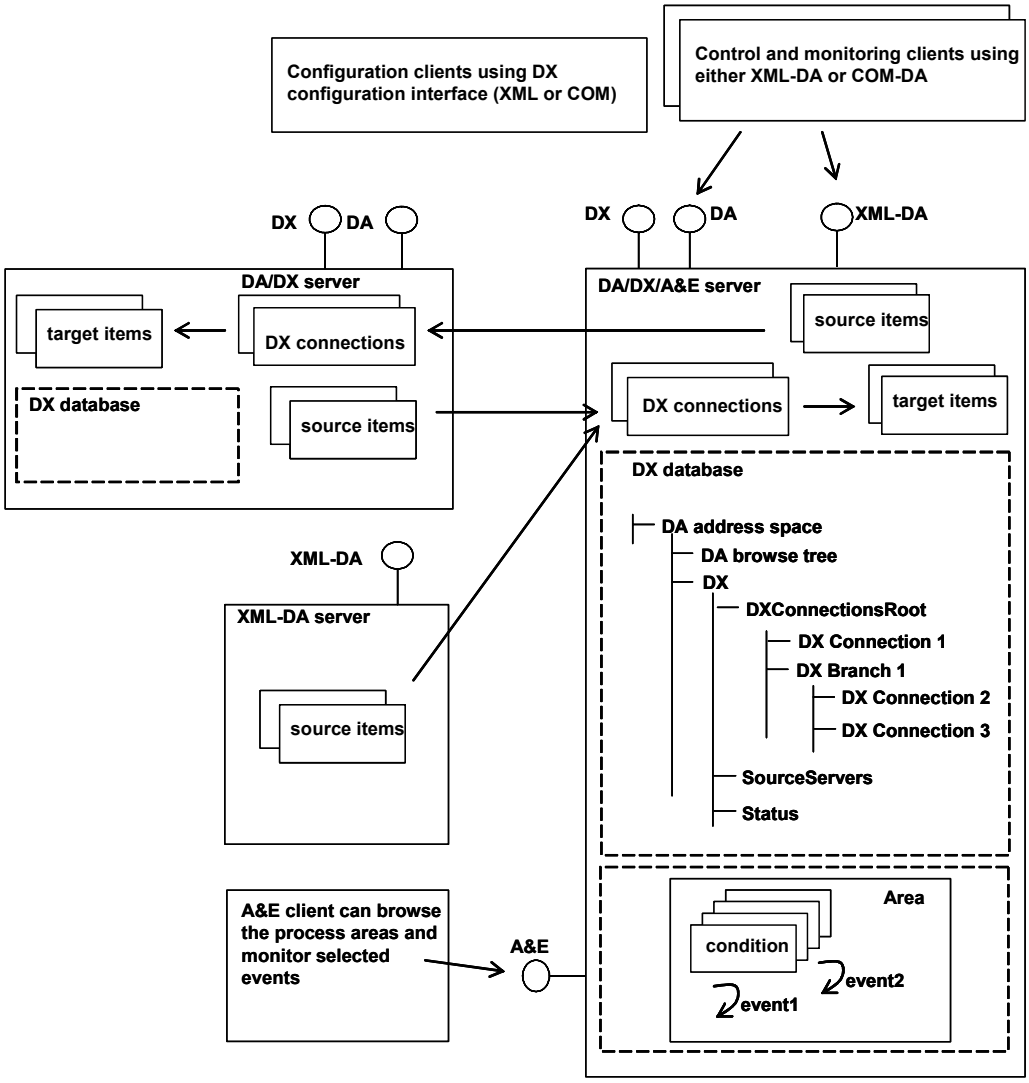


Figure 32. An example that consists of several different OPC components.

DX doesn't change the fact common to all OPC specifications, that physical locations of connected signals are defined during design-time.

For event notification and acknowledgement OPC A&E provides means to marshal event information to interested clients. A full-featured OPC A&E server provides all functions necessary for event notification and acknowledgement service. A&E specification defines the means to classify events in server-side and acknowledge alarms when such behaviour is required. Filtering of events can also be done.

The OPC Foundation is still active. The most interesting work is now ongoing in the area of Web services. The OPC unified architecture working group is working to make XML-based alternatives for those specifications that have only COM-binding. The purpose is to harmonise the set of specifications to form a more "unified architecture" than currently is available when different specifications are combined.

5. Extending the concepts to wide-area distribution

The discussion in the previous two chapters was limited to more or less local issues in one computer node or inside an industrial plant. The common trend is, however, towards geographical distribution of computer systems. In a few cases, such as in energy distribution networks and environmental monitoring, the "process" itself is geographically distributed. More often, the control and data acquisition systems in various remote locations must be integrated with a company's business applications. This adds a number of complicating factors, like communication uncertainties, security issues and remote management of applications. In addition to familiar functions, new services and techniques for business applications must be introduced. With the recent developments in networked manufacturing, virtual organisations and outsourcing of information technology from external service providers the situation has become even more complicated. An application might be used or managed by several organisations and possibly run on hosts owned by a third party. This blurs the concept of a "system" and raises questions about trust and responsibilities of each partner.

Therefore, this chapter actually reflects the most essential situation in the control business and combines the challenges and concepts related to future automation architectures. In the following sections we try to describe the business needs and technical requirements implied by wide-area distribution and business integration. On the basis of our functional concepts a demonstration has been implemented to clarify the ideas and to test some of the enabling technologies. The suggested implementation architecture will be presented at the end of the chapter.

5.1 Applications and requirements

Monitoring and control systems can be supplied and maintained as such or, more often, as a part of a production facility. In both cases, the trend is towards global markets. From the system supplier's point of view, control systems are typically installed at remote locations. For example, we can identify the following situations requiring wide-area communication (some of them are illustrated in Figure 33):

- In a typical case, when a separate monitoring or control application has been delivered to a distant customer, the system supplier often provides services related to customer support, remote diagnostics and software updates.
- An intelligent process control valve acts as an actuator connected to the local basic process control system. In addition, it generates advanced diagnostic information to the customer's maintenance staff. As analysing this data may be difficult, the valve

manufacturer provides remote services for monitoring the performance of all valves installed at the plant.

- A more complex machine or a process section delivered together with an embedded or a separate control system leads to a similar situation (not to mention moving vehicles and ships). The manufacturer having the detailed knowledge about the design may take the responsibility for maintenance, performance monitoring and continuous process improvement. This includes both the mechanical equipment and the control system. In addition to providing services to the customer, the manufacturer is interested in collecting operational experiences about their products. This information might be used for product development and for benchmarking similar plants operated by one or more customers.
- In some cases a machine, a plant section or a total plant is operated from a remote location, either by the equipment supplier, the owner or by a third party. For example, small (diesel and hydro) power plants are normally under remote control. A second case can be found from delivery of industrial gases. A local gas storage or a small air separation unit can be installed next to a customer's factory and operated remotely by the supplier (sort of vendor managed inventory).
- Some processes, such as energy distribution networks, oil and gas pipelines and district heating networks, are inherently distributed. The dependencies between process sections create the need for (centralised or distributed) co-ordination. This, in turn, calls for real-time communication between local controllers. This horizontal communication is fundamentally different from the otherwise vertical information flows required for remote operation and data acquisition.
- There are systems that are merely installed for data acquisition and monitoring purposes. For example, these may be condition monitoring systems and storage level measurement systems (e.g. tank farms in oil refineries) used in the industry. On the other hand, environmental monitoring systems (weather, air quality, radiation, etc.) collect measurements from larger geographical areas. The data can be used locally, but in many cases it is transferred to a central location for further analysis.

Figure 33 illustrates some systems and parties typically found in geographically distributed applications. The remote devices and systems are shown in the lower part of the figure. In this case, the examples include weather stations, a typical process control system, a PLC controller embedded to process equipment (a centrifuge) and, finally, an intelligent process control valve.

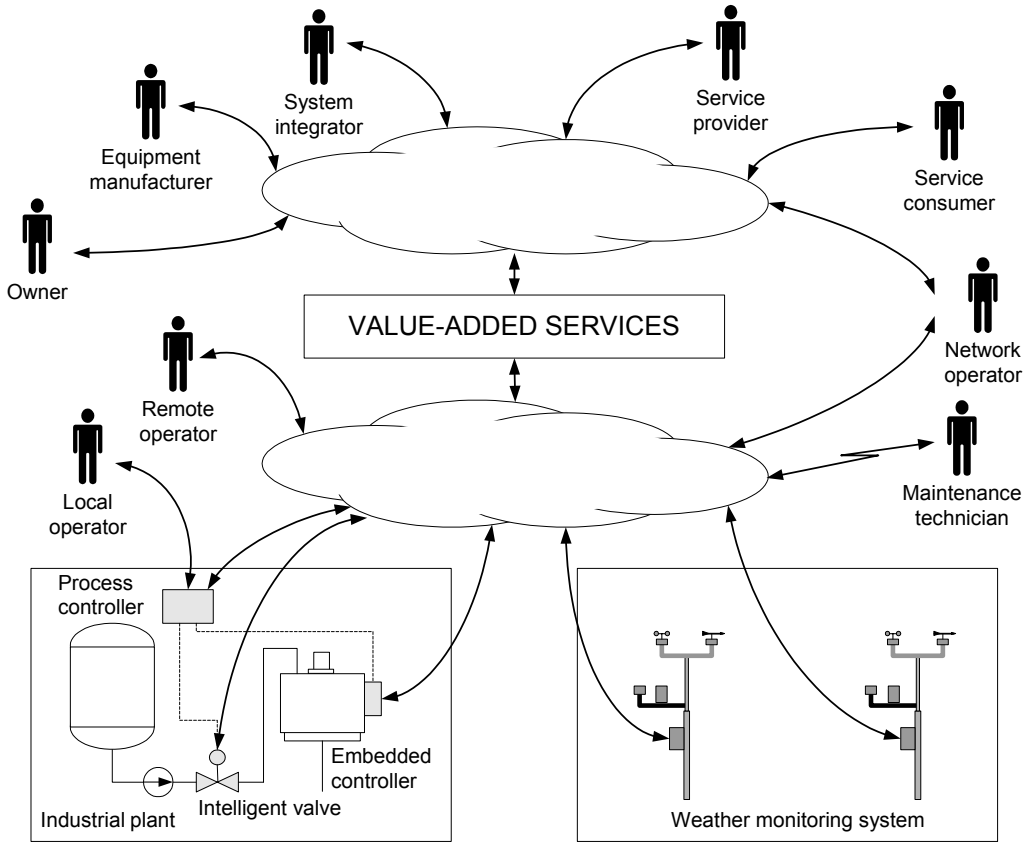


Figure 33. Some examples of systems and stakeholders participating in geographically distributed applications.

The role of "value-added services" shown in the figure is to collect, store and send forward the raw data generated by remote systems. In addition, they may perform more complex data analysis, mediate data and commands down to remote systems, or take care of system security. In practice, these functions might be carried out in a server or a network of servers.

Value-added services typically use some type of existing communication infrastructure (Internet, GPRS, satellites) to reach the remote systems. In addition, the users connect to the servers through the Internet or the company Intranet. However, the data content and the character of communication are different. That is the reason why Figure 33 shows two "clouds of communication infrastructure".

In the upper sections in Figure 33 there are various local and remote user groups. They are normally working for an organisation and acting in one or more roles with respect to the remote systems. In addition to working as individuals, users are often members of a collaborative team trying, for example, to analyse the causes of a process disturbance. This co-operation should be supported by the value-added services.

Of course, there are many variations of this generic pattern. Sometimes services run in a gateway machine installed at a remote location. Sometimes users are allowed to connect directly to the remote devices, usually for low-level diagnosis and maintenance tasks.

5.1.1 End user roles

The roles of various stakeholders can be shortly defined as follows. (It should be noticed that an organisation or a person may actually have several roles.)

- A local operator monitors and controls the process and the monitoring or control system from workstations in a centralised control room or via operating panels close to the actual equipment. The operator usually works for the owner.
- A remote operator has tasks similar to a local operator but with a limited authority. He might use a mobile terminal or a normal workstation connected (directly or through a value-added server) to the remote system.
- Maintenance technicians focus on system maintenance but may use same types of user interfaces as local and remote operators, i.e. workstations or mobile devices.
- The term owner refers to the organisation that owns and usually operates the plant or the system. The primary business interests are related to areas like asset management and production planning. As large companies run several plants in different locations, they often need to monitor and compare the performance of the plants to enable continuous improvement.
- The equipment manufacturer supplies the production machines and process equipment like a robot, a control valve, or a power plant boiler. In some cases the delivery includes a control system. As a third case, a control system vendor can be in the role of an equipment manufacturer. In addition to selling as many devices as possible, the goal of an equipment manufacturer may be to provide after-sales services to their customers, and to collect feedback information for their product development.
- A system integrator typically delivers turnkey systems, such as monitoring, control and information systems, or industrial plants and plant sections. Basically, a system integrator is independent of any equipment manufacturer and uses commercial components best suited for the application and plant location.
- Service providers focus on helping plant or system operators in maintaining and developing their business processes, personnel and production resources. Maintenance, condition monitoring and process studies are typical examples. Running a value-added service on a computer is a further example not directly related to the remote systems. A service company might be independent of the other parties, but often equipment manufacturers and system integrators are willing to take this role.

- Service consumers are people and organisations external to the application but connected to it in some way. For example, average citizens may use localised weather forecasts provided in the Internet by a service provider that collects data from a weather monitoring system. In some cases, for example in the nuclear industry, regulatory bodies need access to some information.
- A network operator provides services related to the communication network.

5.1.2 Communication needs

In the business case described above following types of communication needs can be identified:

- Buffered data acquisition is perhaps the most typical case. More or less raw data is collected and analysed by a value-added server or an end user. Usually the data to be monitored and the frequency of data transfer are statically configured. In addition, it should be possible to define the data collection on an ad-hoc basis. As real-time performance is not required, communication costs can be optimised. However, it is critical that the data can be received sooner or later.
- Reporting refers to the remote system sending a larger data-set that may consist of raw measurements or locally pre-processed information. These reports can be read by the server or sent by the remote system, either cyclically or triggered by local events.
- Operators, maintenance technicians and (possibly) service providers are informed about interesting state changes through event notification. For example, alarms on unwanted process conditions or equipment failures should be routed to the mobile phones of responsible persons. Depending on the delays that can be tolerated recent events can be periodically queried by a server or sent actively by the remote system.
- Control actions and parameter updates require that a remote communication partner is allowed to write data to the remote system. A remote operator might want to start a pump, or an instrument technician would need to adjust a control parameter. Usually, these actions take place irregularly and relatively infrequently.
- Continuous monitoring is needed only in a few situations, for example when a remote operator or a serviceman wants to observe process and machine behaviour on a graphical display. Here, “real-time” communication may be critical. Tolerable delays depend on process dynamics and characteristics of human users.
- Software downloads imply the need to update potentially large data-sets like files and configuration databases to remote systems.
- In some applications, such as in intelligent logistics, two remote systems in different locations may have the need for machine-to-machine (M2M) communication over the public network.

- Finally, it should be remembered that remote systems often consist of several computers connected by a communication network. Therefore, local distribution services are needed to support this peer-to-peer (P2P) communication.

5.1.3 Technical requirements

Geographical distribution implies a number of technical issues and requirements for system architectures and communication infrastructures. These are summarised below.

- Due to the special expertise required for the operation and maintenance of complex systems, various after-sales services are now a growing business for equipment manufacturers and system integrators. More generally, there is a need to more efficiently utilise the great amount of information generated by existing process control and data acquisition systems. For example, large industrial companies collect performance data from their plants for benchmarking purposes. As a further example, environmental measurement data can be processed for providing focussed and localised services to the public. To support this type of business remote monitoring and control systems must have information handling capabilities beyond those found in traditional applications.
- For improved customer service and for lower travelling costs remote systems should be monitored, diagnosed and updated over communication networks instead of sending a technician to fix the problem. When software updates are in concern, there may be the need to make the same modification to a large number of systems. For this purpose, there should be a modular software architecture allowing individual components to be easily added and removed, parameterised and connected. Of course, flexible configuration is essential during the initial application development also. While true ad-hoc configurations are not common in industrial applications, the demand for flexibility is growing. Reconfigurability and plug&play mechanisms are, therefore, desirable features.
- Industrial systems must be kept under strict configuration control, and modifications must be done in a safe way. Therefore, information security becomes an issue. The data transferred must be kept confidential and consistent. In addition, only authorised people should be allowed to make control actions or modifications. In particular, parameter changes and software updates should have no way to jeopardise the process under control or to make the computer application crash. For this, the correctness and compatibility of new software should be checked and tested in advance. In addition, the local platform should have some means to prevent installation of incompatible or malicious software. This calls for configuration controlled life-cycle management and certified software components (cf. signed Java applets). To enable the analysis of possible problems all modifications should be logged.

- The vision of an open, standardised platform and global markets of commercial components in Internet-scale applications also raises the question of software licensing. There should be mechanisms for preventing components from being illegally copied and installed.
- Remote applications are often installed at distant locations having less developed infrastructure services. Instead of a single device, a remote system may consist of a group of devices, perhaps only partially connected by a wired or wireless communication network. This results in unreliable data communication and considerable variations in the quality of service. The systems should be robust enough to cope with these problems. For example, they should have redundant communication channels and sufficient data buffering capacity. Loosely coupled systems based on asynchronous, message oriented communication are more tolerable than software models that use synchronous procedure calls extensively.
- Due to the lacking infrastructure, wireless communication (GSM, GPRS, satellite link) is often the most feasible solution. Unfortunately, these techniques are characterised by limited bandwidth and high communication cost. Insufficient power supply may further limit the communication capabilities. Therefore, the amount of data transfer must be minimised. Possible approaches include local pre-processing of raw data, sending the necessary information only, data compression and sending larger data sets at a time. Due to power saving (or unreliable connections, or high connection costs) a remote device may be "off-line" for rather long periods.
- In addition to communication bandwidth and power supply, remote devices often have limited processing power and memory capacity. Usually there is no hard disk, and the amount of working memory may be limited to a few megabytes. Techniques such as flash memory are used to provide permanent data storage. As a result, lightweight software should be preferred. However, the rapid development of microelectronics allows us to focus more on new functionality rather than technical constraints or manufacturing costs.
- The rapid development mentioned above also implies shorter product lifetimes. Manufacturers must maintain several software and hardware versions of their products. Sometimes even the basic platform must be changed. So, the reusability and portability of the architecture and software modules become important. For example, the system might have an independent communication layer supporting various communication protocols. Or, use a programming language, such as Java, that is less dependent on the underlying operating system.
- So far, the majority of applications is built and managed by one or just a few companies. While outsourced components are used, the manufacturer or system integrator has full control on system composition. The use of open international standards makes this easier and saves money, but is not necessarily a precondition for the business. In the future however, the number of players involved in running a

system will presumably increase. The role of commercial off-the-shelf components gains importance. Consequently, international standards and open system architectures become a critical issue.

This rather lengthy list of requirements actually boils down to an open control platform supporting reusable software modules, flexible configuration and rich, location transparent communication. Our working hypothesis has been that these requirements can be fulfilled by modular and reusable automation components characterised by intelligent internal activities and a powerful set of interaction mechanisms. To a large extent, these concepts have emerged as a combination of the tradition in industrial automation and recent advances in information technology. It is recognised that this proposal will probably never be fully implemented in practice. However, its goal has been to give ideas for future development in research, standardisation and industry-driven product development.

5.2 Concepts and architecture of a geographically distributed system

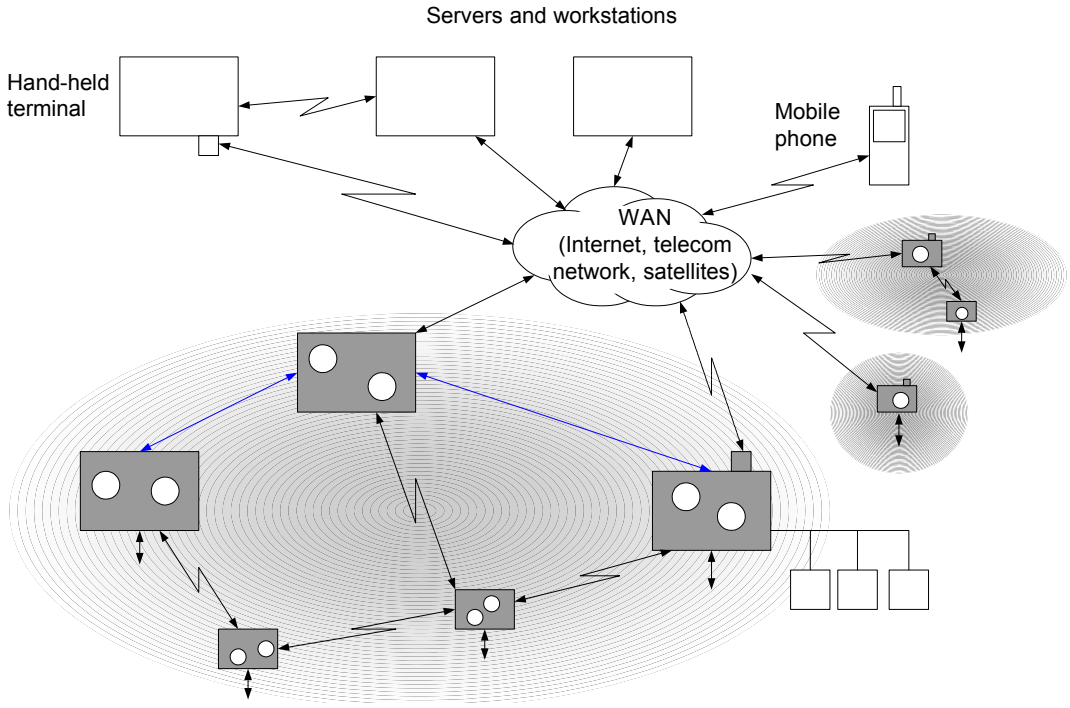


Figure 34. Typical devices in a geographically distributed system.

Figure 34 illustrates typical structure of a distributed system consisting of individual devices and local islands of computers spread over a wide geographical area. In the simplest case, a single device node with a few sensors makes up a local subsystem. In

more complex applications, a full-fledged data acquisition or control system may be needed. Communication takes place over wired and wireless connections. Because of the uncertainties, redundant routes within the local system and out to wide-area networks may be needed. However, it can be foreseen that networks are not always fully connected. So, device nodes should have the option to serve as routers for their neighbouring nodes. The sections below discuss the architecture and possible implementation techniques in more detail.

5.2.1 Overall system structure

Due to its distributed nature and remote operation and maintenance, possibly from several locations, the concept of a "system" may become obscure. Firstly, we have a set of physical computers and the necessary basic software (e.g. operating system) owned and maintained by one or more companies. Secondly, there are software components running on those nodes. Thirdly, components co-operate to create distributed applications.

In principle, components could be spread all-over the world, and applications could be wildly overlapping. However, industrial systems require clear responsibilities and strict configuration control. In addition, it would be convenient to work with sets of components instead of individuals. For example, maintaining security settings of hundreds of nodes would be rather troublesome. So, basic concepts like "application" and "system" should be clearly defined.

It is possible to see three layers of platforms as illustrated in Figure 35. The lowest layer consists of physical computers owned and maintained by one organisation (owner 1 and 2 in the figure). *Logical nodes* include *containers* capable of maintaining and running software components. It is possible to have several logical nodes on a single computer. A set of distributed logical nodes form a system *platform*. This platform might be provided by another actor (provider 1 and 2). Within these platforms, various *applications* can be defined. It might be possible for an application to spread out to several platforms. On the other hand, a platform may host more than one application.

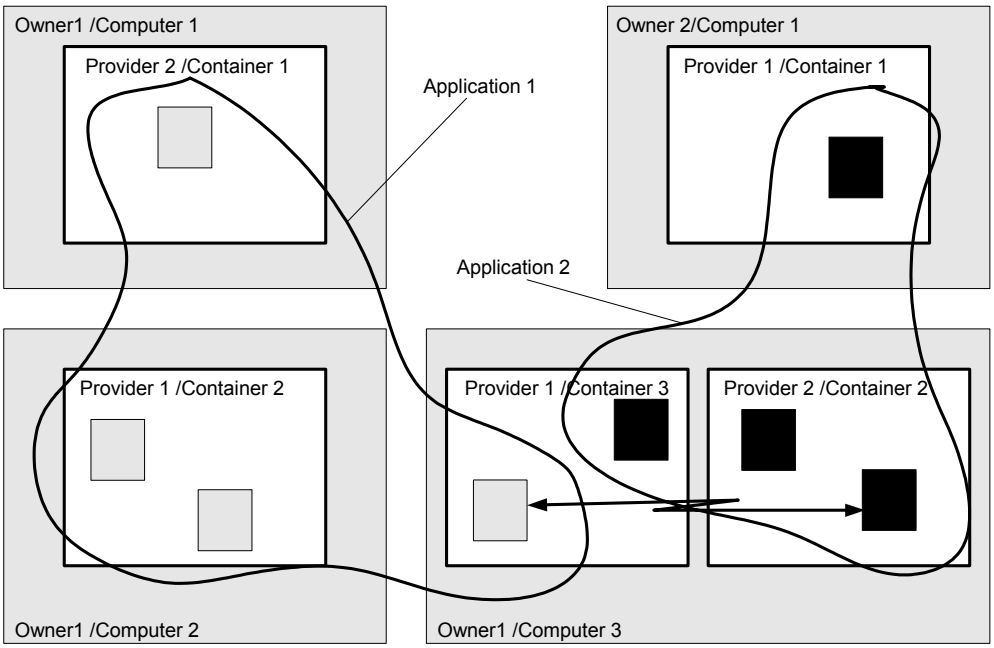


Figure 35. Three possible system layers: a) computers, b) platforms and containers, and c) applications and application components.

For management and security purposes, the boundaries of an application must be clearly defined. So, each component should preferably be part of exactly one application. However, there is the need of applications or components to interact with other applications. Perhaps, selected components might be exposed (via interfaces) to other applications. Or, shared components might be members of more than one domain or group. An alternative could be to use general-purpose techniques such as Web Services, especially when the communicating applications are based on different technologies.

In this context, a number of different business roles can be identified, for example:

- provider of the hardware platform
- network operator
- platform provider
- application provider
- application user
- provider of various management services
- etc.

Obviously this situation seems complicated. Fortunately, practical applications are (at least currently) much easier. Often one organisation owns, maintains and perhaps operates the computers, platforms and applications. Some computers (e.g. workstations)

may run also other types of software, but we can assume that the components inside a container belong to the same application.

With these restrictions we can define that an *application* consists of a set of *automation components* located in a set of distributed *logical nodes*. Every component should be able to find the data, events and services published within the application. On the other hand, this information should not be (automatically) visible outside application boundaries. So, some structure is needed for distributed directory services. The responsibilities and rights regarding system updates and maintenance should also be clearly defined.

To make this possible, the nodes of the platform are organised into a hierarchy of *system areas* (Figure 36). Each node knows its parent and child nodes. For improved reliability nodes should (learn to) know their siblings also, and replicate their information. In addition, root nodes should be allocated to redundant computers. Basically, all the nodes in a platform area are equal. The difference comes from the fact, that one (or some) of the node(s) has a special role in maintaining the system topology.

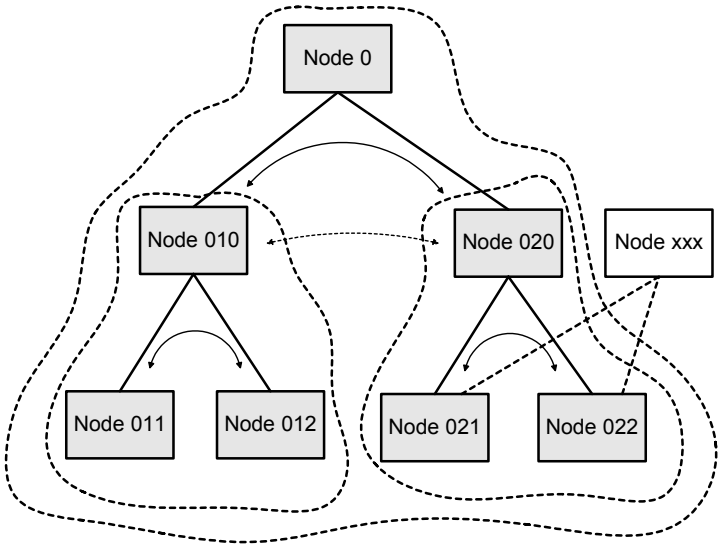


Figure 36. The system is hierarchically organised as system areas.

For future flexibility, nodes might be allowed to join more than system area (dashed lines in Figure 36). In this case, the name of the context, such as application name, should be attached to the parent-child associations between the nodes.

5.2.2 Node architecture

Each *device unit* in a system hosts one or more *logical nodes* where various types of software components can be installed. The concept of a *container* is essential for enabling the use of commercial off-the-shelf and company-specific components. The interfaces implemented by the container and by the contained components define the exact rules for component developers.

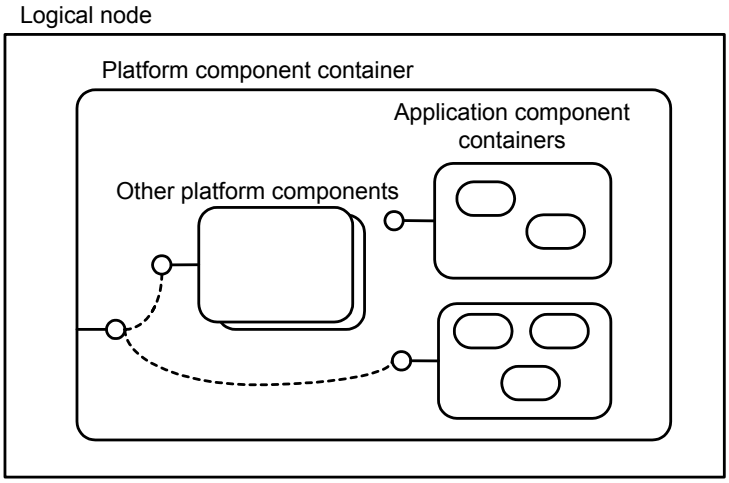


Figure 37. Layered containers in a logical node.

As explained in section 3.5, a logical node can contain several layered containers within each other. In addition, an outer platform component container might host several application component containers for different purposes (Figure 37). For example, one container might be dedicated for process control components and another for application components related to remote diagnostics and maintenance. A reader interested in control components can find a more detailed discussion in Jukka Peltola's master's thesis (Peltola 2002).

This section focuses on the platform component container. However, no strict distinction is made between platform and application components. In an environmental monitoring for example, is would be quite natural to implement application-oriented functions, such data collection and reporting, directly as platform components.

While logically seen as a "centralised" source of services, the *container* itself consists of a set of *platform components*. Potential candidate components of the container are

- root: plays the role of the container itself; helps in searching available container properties and services of other components (below)
- directory server: for local and distributed name services

- data distribution server: takes care of continuous and event-driven data distribution
- event notification server: distributes event notifications
- service distribution server: distributes services
- security server: helps other components in checking identity and authority of incoming requests.

Different implementations of these components are possible. For example, only local data distribution may be supported. To make this configurable, the server might consist of separate sub-components for intra-node, local area and wide area communication (Figure 38).

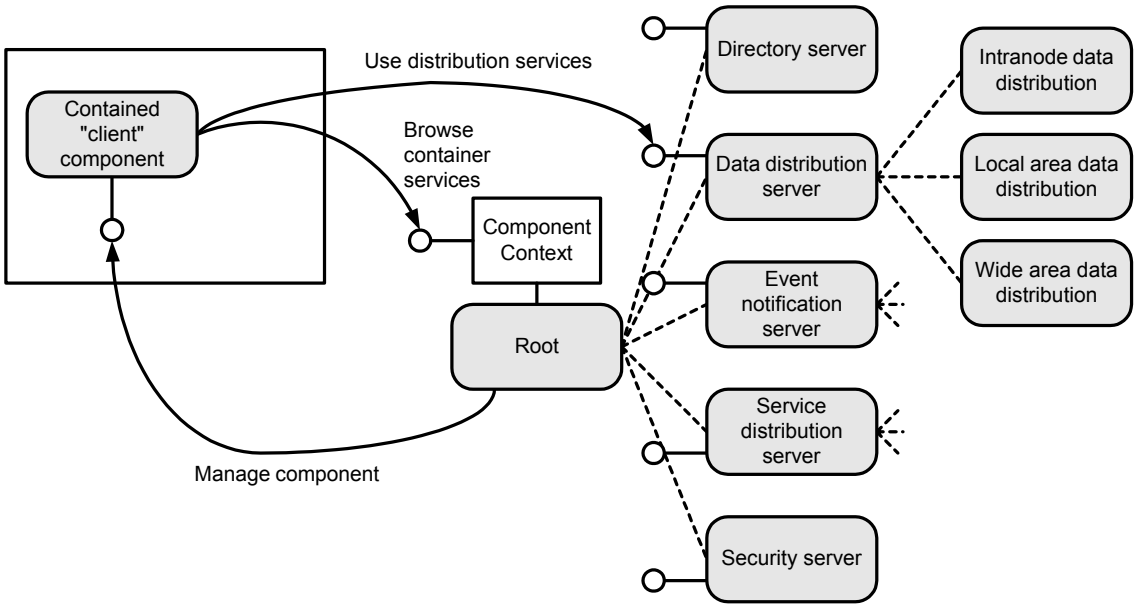


Figure 38. Container actually consists of several platform components configured according to application needs and hardware resources.

The primary connection between the client component and the container is a *component context* object created and maintained by the container for each component. The implementation of this object is private to the container, but it must implement a specific interface called ComponentContext. This object is the representative of the container to the component and provides it access to the container services.

In a dynamic situation, both the container and the contained components must be kept informed about the changes in system state, i.e. about new and removed components and services, changes in component states, etc. To achieve this, components must issue appropriate events that are collected by the container. On the other hand, components

can subscribe to relevant events using a suitable event filter. The container forwards incoming event to other interested local and remote components.

In a fully distributed application there is no difference between local and remote events. Therefore, the common event notification services might be used for this purpose also. Of course there must be an agreed encoding of event attributes to enable appropriate event routing and interpretation.

5.2.3 Essential platform components

This section describes shortly some of the most important components of the platform.

Node root

Every *logical node* in the system needs a root component that manages the components and other resources in the node. As indicated above in Figure 38, the platform components can be seen as a hierarchical structure with node root on the top. So, root is a representative of the whole logical node and its platform component container to the other components. While platform components, in fact, are parts of the container, they can use its services as if they were normal "client" components.

Node root creates a ComponentContext object for each new component and hands it over to the contained component at start time. This reference to the component context interface gives the component an access point to all the services provided by the container. Basically, the component context interface includes methods for searching the services registered by other platform components. For example, a client component might look up for a "DataDistribution" service as shown in Figure 38. Once got a reference to the service implementation it can use its methods in order to connect to a remote data source.

Directory server

As explained earlier, our main principle of building applications is to link data and service ports of components. A registry or lookup model is used here for its flexibility. A producer of data or a service advertises itself to the local distribution server as shown in Figure 39. The consumer, in turn, connects to the source by issuing a connect request to its distribution server. The distribution and directory servers running in each node create the necessary registry, connection and subscription objects and propagate new data to peer nodes and finally to consumers.

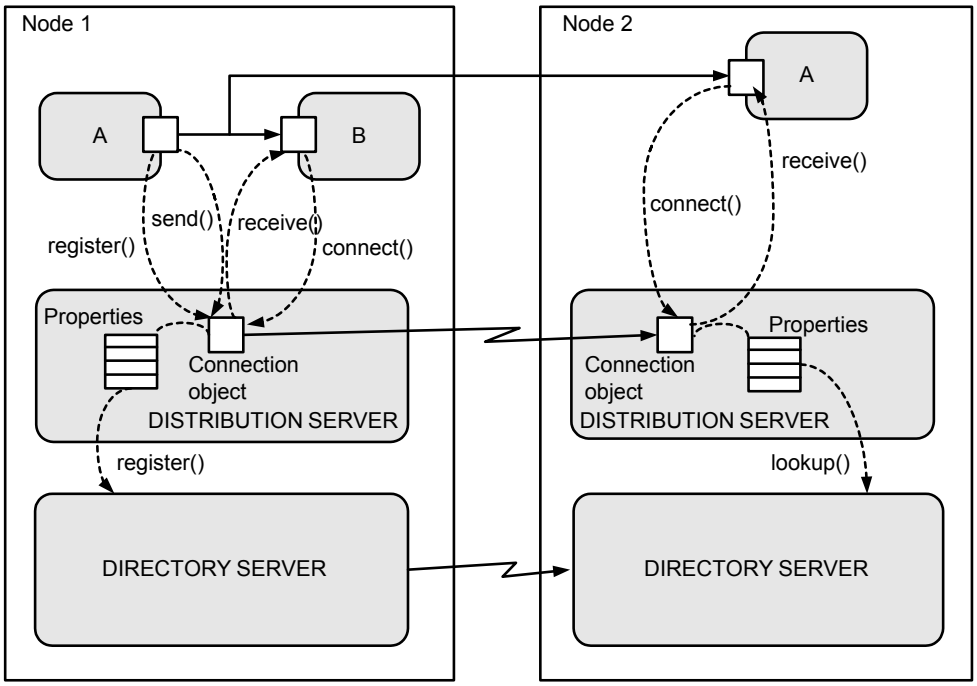


Figure 39. Use of directory services for connecting local and distributed terminals.

Separate interfaces must be defined for distributing data, event notifications and services. When registered, a source terminal is provided with a set of properties. Correspondingly, the consumer announces itself as the connection target and gives another set of properties that define a filter or various QoS parameters for the request. Only the generic and minimal information is defined as method parameters, all others can be added as properties.

A distributed implementation of directory services is an essential part of the platform. Its detailed description is outside of our scope. However, it can be expected that suitable solutions have already been developed in various areas of information technology. So, the basic registry and lookup operations should be specified according to the common principles of LDAP, JNDI etc. to enable easy implementation.

Data distribution server

By using the *ComponentContext* object a component can query the local node for available services, including for example data distribution services. Once got a reference to the distribution service, the component can ask for a connection to a local or remote port by its logical name (Figure 39). If successful, an *ItemConnection* object is returned as a representative of the "wire" to the other terminal (Figure 40).

ItemConnection is an interface implemented by the connection objects maintained internally by the data distribution server. In this basic communication pattern, the data

consumer connects to the source effectively subscribing to the data items sent by the source. Various control parameters can be specified in the properties, such as data accuracy (dead-band) and minimum update period. The *ItemConnection* objects at both ends present the logical connection ("wire") between the input and output ports.

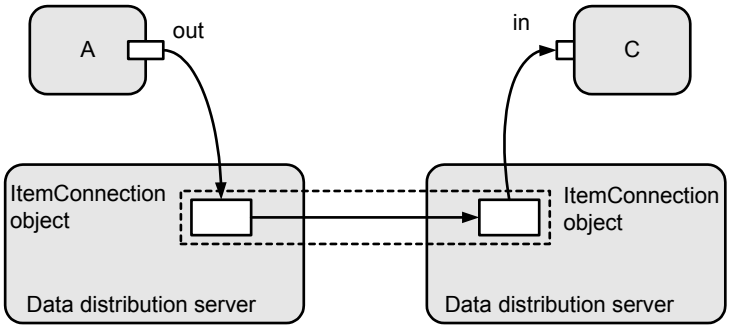


Figure 40. Connection is a "wire" between to ports.

In addition, the item connection should provide a synchronous read operation for the consumer. This can either return the most recent value stored in the local item connection object, or it can perform a (remote) read to the data source. Further, an item connection can be used in the opposite direction. Once connected to a remote item, the component writes an object to the connection in order to update the item value. To make this possible, the initial registration and the connect-request must specify an access property (read, write, or read/write).

Usually data sources and destinations are thought of as observed items and item observers as described above (the general "observer pattern"). This generic approach can simplify the application components. However, application components have the option to use directly the methods provided by item connections.

In this model, event-driven data distribution is rather similar to continuous data distribution. In both cases, data items (objects) are transferred from a uniquely identified port to another. In the event-driven approach, however, no continuous transfer semantics is available, but the transfer is always triggered by an application event.

Service distribution

Similar to data distribution, the service output ports of a component, and their properties, are registered to the local directory server. Available services are looked-up when a service consumer component wants to connect its service input port to the registered service. Similar to normal remote method invocation, the connection objects at both ends serve as a proxy and a stub to the application objects.

Event notification and acknowledgement server

In contrast to the previous two connector-based "point-to-point" communication mechanisms, *event notifications* are broadcast to the whole application and picked up by interested listeners on the basis of an *event filter*. So, this is an example of the content-based publish/subscribe model introduced in section 3.3. In content-based messaging there are no connections, but the publications and subscriptions maintained by the event notification server present the application's entry-points to and from the common data space (Figure 41).

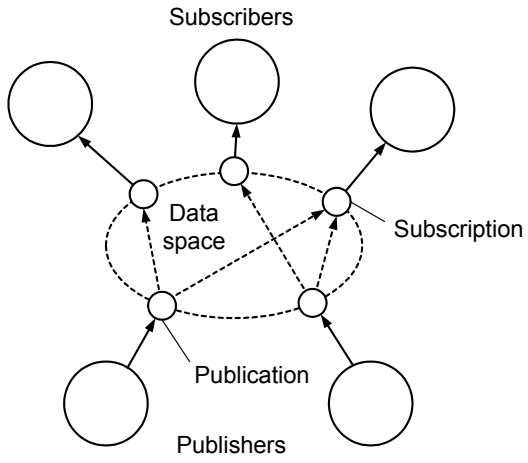


Figure 41. Publications and subscriptions are entry points to the common data space.

The transmission and subscription methods must, therefore, be richer with suitable component and event or notification classification "masks" (filters). The subscription methods should also be able refer to specific attribute values of the original event, such as process area, event type or alarm priority. Moreover, with one subscription it is possible to obtain many kinds of notifications. The sources of the messages are, in principle, not significant. However, it is practical to include that information as a message parameter.

The event notification server should be able to create routes for the notifications by the content of the event notification according to the publications of the event sources and subscriptions of the interested listeners. Moreover, the event notification server should keep track of the required application level acknowledgements. Application level acknowledgements can be a required feature of event notifications. To keep track of the acknowledgements, the event source should implement a listener for these. Thus, the event notification server acts as a local event handler.

In geographically distributed applications it is sometimes advantageous that the subscriber can also define the contents and protocol of the event notification message. E.g. a service person on duty could get a SMS message to his mobile phone, whereas the same event can be more thoroughly indicated on a browser display.

5.3 Enabling technologies

An architecture outlined in the previous section calls for technical solutions that support loose coupling of components, functional integration of heterogeneous applications and the management of uncertainties and threats encountered in geographically distributed communication networks. Two techniques of particular interest to our concepts are shortly introduced below.

5.3.1 Web services

Web services are self-contained (i.e. component-oriented), self-describing, modular applications that can be published, located and invoked across the web. Once a web service is deployed, other applications (and other web services) can discover and invoke the deployed services. Web services is also the most widely used technology to implement Service-Oriented Architecture (SOA), a more general approach to business applications which is becoming increasingly popular.

Web services use standard Internet protocols: the Simple Object Access Protocol (SOAP) for transferring request and reply messages (on top of TCP/IP), the Web Services Description Language (WSDL) for the definition of service interfaces, and the Universal Description, Discovery, and Integration (UDDI) for advertising and searching available services at runtime (Figure 42). These techniques can be used in various combinations, over the public Internet or in a private network. In particular, OPC XML-DA is an application of Web services to industrial automation.

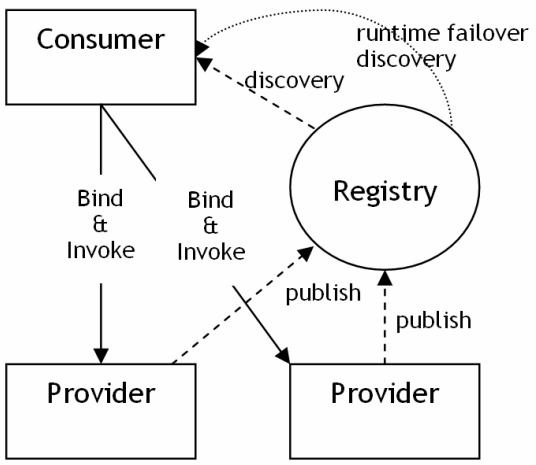


Figure 42. Service registry (modified from Wilkes 2004).

Web services is seen as a potential paradigm that could be applied to create intelligent applications spanning over large geographical areas and organisational boundaries. Web services are generally considered to be most useful in cases where (W3C 2004)

- applications communicate over the Internet, and there are no guarantees about reliability or latency
- all the service consumers and providers cannot be upgraded at the same time
- the system entities use different hardware platforms, operating systems and products from several vendors.

Ideally, solutions based on Web services yield profit in the form of lowered software development costs (software and service reuse, interoperability), faster time to market (standards based solutions), and economics of scale (integrated systems).

Standards and specifications

The original Web services specifications (i.e. SOAP, WSDL, and UDDI) are still the foundation of the Web services technology. During the recent years, major IT-companies (e.g. IBM, Microsoft, BEA) in co-operation with smaller companies and industrial consortia (e.g. OASIS, http://www.oasis-open.org) have authored a variety of new specifications, which have been designed to be interoperable and extensible. This is further promoted by the Web Services Interoperability organisation (WS-I, http://www.ws-i.org), an open industry organisation that aims to promote interoperability by defining profiles (e.g. the WS-I Basic profile) that are comprised of a set of Web services specifications. Profiles also include guidelines for implementing interoperable Web services. In general, the specifications are gradually reaching a level, where they meet the requirements of industrial customers, e.g. reliable integration of internal business processes and establishment of virtual organisations among business partners (Estrem 2003).

As TCP/IP and small embedded web servers are entering the plant floor, it is expected that Web services will eventually expand also to low level devices. The Devices Profile for Web Services (WS-DP, Schlimmer 2004) gives suggestions on what subset of the Web services specifications is applicable to lightweight devices. WS-DP might be feasible in geographically distributed automation systems that wish to utilise message-based communication. Still, verbose message formats (e.g. XML) always create communication overhead and require computing for message processing. Some efforts have been made to overcome those problems by introducing models that slightly limit the features while providing enhanced performance with a binary message format (e.g. Fast Web Services, see Sandoz et al. 2003).

Security

The W3C Web Services Architecture states that the security of a system is primarily determined by the level of authentication, authorisation, confidentiality, integrity, non-repudiation, and accessibility. There are two principal mechanisms for Web services security: transport level (e.g. the Secure Sockets Layer, SSL) and message level security, where the messages headers are utilised for security. The latter is provides better end-to-end security, which is important to the success of Web services in industrial automation.

The traditional network security models, such as firewalls and Virtual Private Networks (VPN), are not alone adequate when using a Web services based architecture. There must be mechanisms to control security in the application and operational level (e.g. user permissions) to achieve end-to-end security. The Web services security specification (WS-Security, see OASIS 2004) addresses those issues, and for example Gartner, Inc. recommends using it in all Web services implementations that transmit data through firewalls.

For small devices the WS-DP claims to define a minimal set of implementation constraints and specifications that are needed to implement secure Web services messaging, discovery, description, and event handling in a resource-constrained environment (Schlimmer 2004).

WS-Notification

Basically, the Web services technology is intended for invoking remote procedures over the net, but some of its principles are feasible also in event-driven (i.e. notification-based) communication. There are two Web services specifications, namely WS-Eventing and WS-Notification, which address this issue. The WS-Eventing only includes the basic operations and interfaces of notification producers and consumers. The specification is closely similar to WS-BaseNotification that in turn is the core specification of the WS-Notification Framework (IBM 2004a).

The WS-Notification Framework (IBM 2004b) provides functionality to implement robust and scalable event-based architectures that may include message brokering and topic-based subscription management. Altogether the specification suite (including WS-BaseNotification, WS-BrokeredNotification and WS-Topics) defines a model to implement a topic-based, direct or brokered, publish/subscribe pattern by using Web services technologies. The model of an event-based architecture includes

- a standard message exchange for service providers and for notification brokers
- operational requirements for notification senders and receivers
- notification message topic expression dialects

- an XML model for topic metadata description.

It is recommended to apply WS-Security to enhance security. Currently there are efforts to define WS-NotificationPolicy and other specifications that would standardise the way Web services implementations make use of the notification pattern.

5.3.2 Internet-scale notification services

The event-driven, or notification-based, interaction pattern is a commonly used pattern for communications in Internet. The components that wish to consume information (NotificationConsumers) are registered dynamically with the components (NotificationProducers) that are capable of generating information. As part of this registration process the NotificationConsumer may provide to the NotificationProducer an event filter, i.e. some indication of the nature of the information that they wish to receive.

Generally speaking, all wide area notification services described shortly below fall under the term *Internet-Scale Event Notification Service*. Almost all of them are based on some form of publish/subscribe architecture. Another common factor in these approaches is the fact that none of them contains user level acknowledgements. The user level acknowledgement must be distinguished from e.g. protocol level acknowledgement. Thus reliable communication model does not solve the problem of user level acknowledgement. In the following we describe some of the most relevant (with respect to automation) event notification approaches.

Java Distributed Event Model (see e.g. JINI 2004) is based on Java Remote Method Invocation (RMI) and is the basis for Jini Event model. The subscriptions are made according to the general Jini principles. RMI is quite heavy a protocol and cannot usually penetrate firewalls.

The Java System Message Queue Platform is an enterprise message server designed for small-scale deployments and development environments. It supports standards-based messaging by implementing the Java Message Service (JMS) specification and supporting the Simple Object Access Protocol (SOAP) specification. The platform supports various messaging options such as synchronous and asynchronous messaging, and messaging models such as point-to-point and publish/subscribe. The software is free. Subscriptions are made by a SQL like query to message header. The subscription can not reference the contents of message, which makes the application of JMS difficult.

Siena, Scalable Internet Event Notification Architectures (see e.g. SIENA 2004) is general purpose Internet wide notification service. Essential parts of Siena are the versatile subscription and event filtering methods. Thus, message passing is done according to matches of message contents to subscriptions. Furthermore, the Siena

servers build the message transporting network according to advertisements and subscriptions. Figure 43 presents interfaces and other parties involved in message passing in the context-based network of Siena.

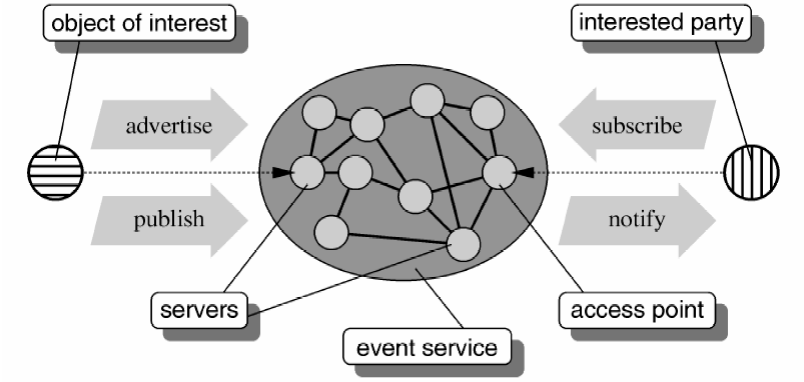


Figure 43. Siena Event notification service (Carzaniga et al. 2001).

CEA, Cambridge Event Architecture, (see e.g. CEA 2004) is based on publish-register-notify approach. This means that the subscribers register to the event sources and by registration give to the source an event filter. Thus this architecture is quite close to that specified earlier.

YANCEES, Yet ANother Configurable Extensible Event Service, (see e.g. YANCEES 2004) is an interesting configurable event service. It is based on typical publish-subscribe kernel and downloadable extensions by which advanced event filtering can be done. The approach is directed towards resource-constrained devices. In the Yancees architecture (Figure 44) events, filtering, subscriptions, and used protocols can be extended by downloadable plug-in modules.

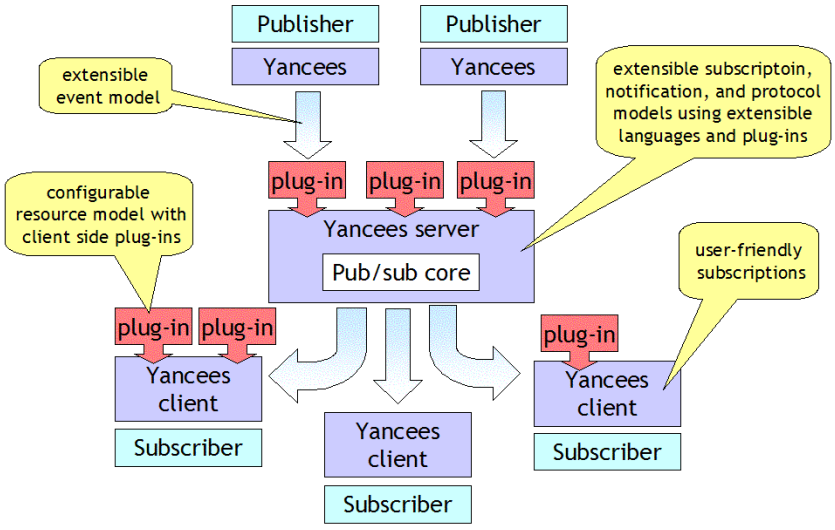


Figure 44. YANCEES event notification system (YANCEES 2004).

Elvin (ELVIN 2004) has been developed at Distributed Systems Technology Centre since the beginning of 1990's. The protocols are standardised by IETF and the research group is one of the proposers and developers of the CORBA Notification Service. The events are basically name value pairs and subscriptions are done by language similar to C. The evaluation a match of the event to a subscription is done by three valued logic (true, false, don't care). An interesting addition to Elvin is its extensions to mobile equipment. Even though Elvin is stateless by nature, the proxy server for mobile equipment is designed to store notifications in case of failures in communication links or for short occasional connection times.

To summarise, none of the approaches is satisfactory as such due to the lack of support for user level acknowledgements. Furthermore, event and notification filtering should be considered more closely even though some approaches introduce quite advanced event filtering possibilities. For example, the Web Services Notification introduces a federation of message brokers, which could be used for more intelligent filtering.

5.4 A prototype implementation

This section presents a prototype implementation that partially demonstrates the architecture described in section 5.2. A detailed description of the prototype and an introduction to Service-Oriented Architecture (SOA) can be found in (Jaakkola 2005).

One of the goals was to develop low-cost nodes, since their number in a system may be high. Use of Commercial-Off-The-Shelf (COTS) software could raise the total costs significantly. So, we decided to make use of open source projects and supplement their outcomes with our own software components. The primary objectives for the prototype implementation were

- demonstration of service-oriented communication between the nodes
- implementation of a proof-of-concept software framework that uses only open source and our own software components
- presentation of the benefits of configurable and component based software.

The basic idea of the prototype implementation was to create a test environment where nodes can exchange information using the message formats defined in the OPC XML-DA specification (OPC 2003). In addition, the idea was to test the content-based publish/subscribe mechanism for event notifications between distributed nodes.

Java technology was initially chosen for the prototypes, because it is portable and offers several development tools and configurations for different types of applications (e.g. the

Connected Device Configuration, CDC, profile for consumer and embedded devices). The architectural design is primarily based on SOA design patterns and Web services specifications. Special emphasis was put on integrating resource-limited devices into the system.

To minimise the required programming effort, available component-oriented frameworks were studied. The Open Service Gateway initiative (OSGi, see http://www.osgi.org) and its standard services (e.g. HTTP server, logging, and security) turned out to provide many features expected from the nodes in our architecture. The OSGi (2003) specification defines a standardised, component-oriented, computing environment for networked services. OSGi Service Gateways enable remote management of software components installed in local devices from anywhere in the network (Figure 45). So, it was applied as the core framework for the prototype implementation.

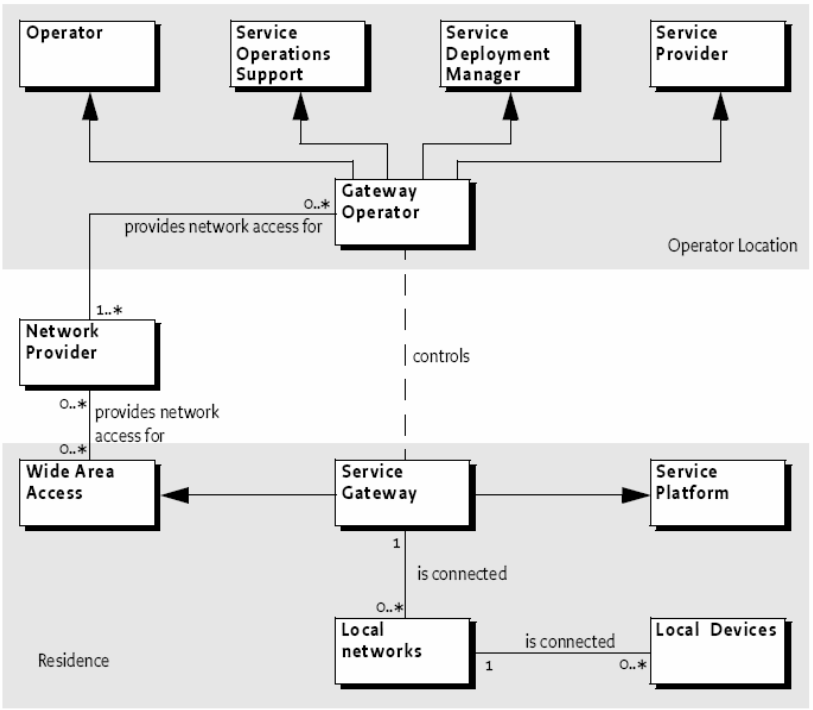


Figure 45. The basic model of an OSGi-based architecture (OSGi 2003).

However, we did not want OSGi to be the only possible operating environment for our automation platform. Therefore, we took an approach where there are two levels of software frameworks (i.e. OSGi framework and OHJAAVA framework, see Figure 46). The prototype implementation utilises some of the standard OSGi services, but most of the components had to be written by ourselves, as the current OSGi specification does not concern itself with eventing or data distribution.

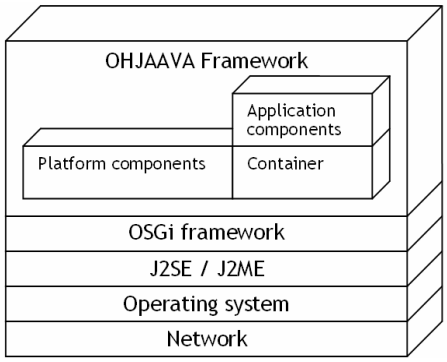


Figure 46. The implementation architecture. Application components are isolated from other layers by the container.

The test configuration was derived from typical usage scenarios that include process supervision, acquisition and presentation of process information, process control, event handling, and process history management. The configuration contained two nodes (Figure 47), but there could have been an arbitrary number of computers. Node A (Figure 48) acts as a service provider and data consumer that collects information from other nodes and makes the data available through a web server. In this configuration other nodes are represented by Node B that can be considered as an information producer. Data and events are generated by a simulator component, which simulates the process of a virtual batch process unit. The simulator utilises DataDistribution and OpcXmlServer components to distribute item values (i.e. signals), while the generated events are distributed by the EventNotification component (based on the Siena architecture, see section 5.3.2). Our OPC XML client component (a platform component on the Node A) has been successfully tested also with various OPC interoperability servers (http://www.opcconnect.com/interop.php), for example with .NET implementations.

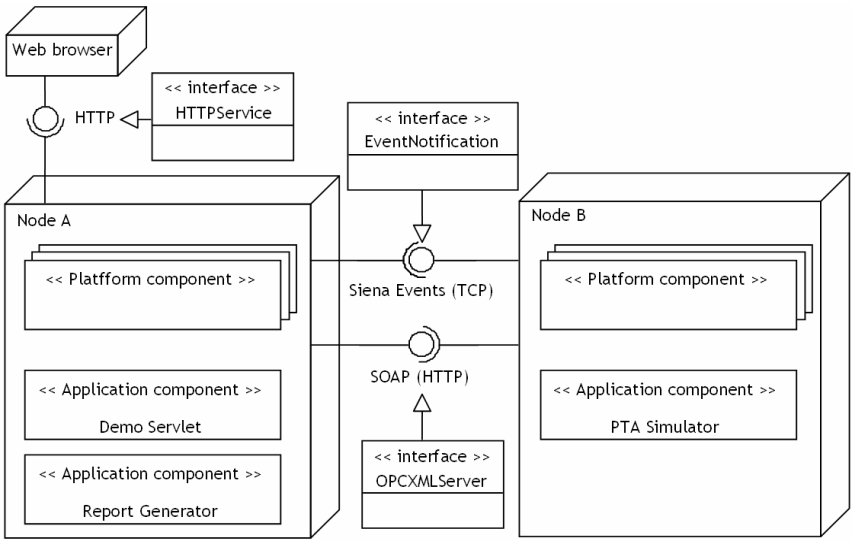


Figure 47. The demonstration configuration (some of the platform components are shown in the next figure).

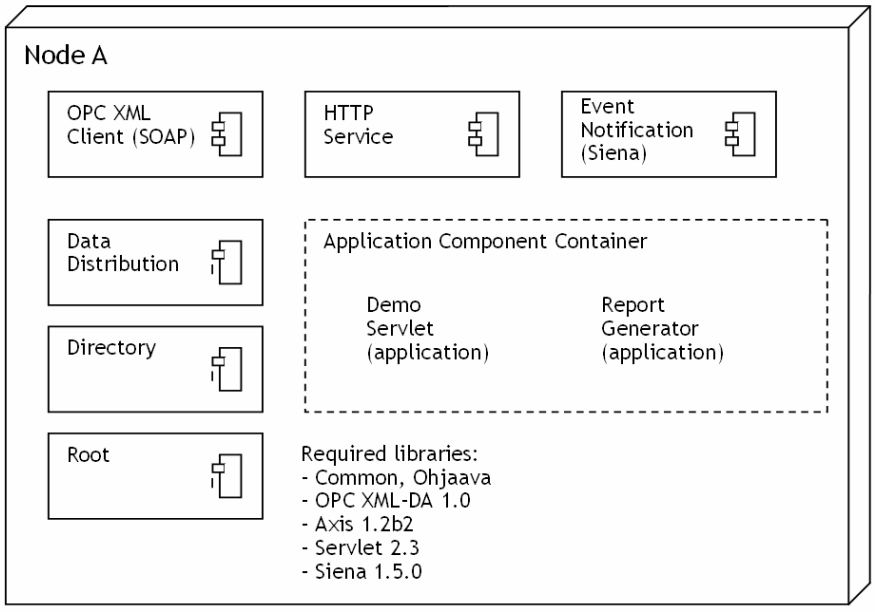


Figure 48. Node A, some of the platform and application components.

As much functionality as possible was built into the library components (e.g. OPC XML-DA and Axis). Application components were isolated from the underlying OSGi framework by the application component container represented by the ComponentContext object (see section 5.2.3). The adapter design pattern was used to wrap OSGi-specific methods to the OHJAAVA architecture (Figure 49).

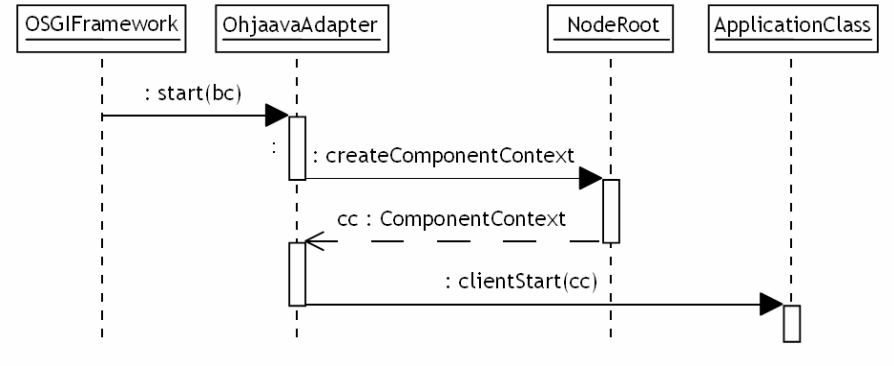


Figure 49. Sequence diagram for starting an application component in an OSGi environment.

6. Summary and conclusions

The goal of this report was to outline the concepts and architecture of a new platform for various types of applications, e.g. in industrial automation, environmental monitoring and energy distribution. These systems are today characterised by the distribution of intelligence to devices close to the processes and equipment under control. At the same time, there is a strong tendency towards vertical and horizontal integration of all functions within a plant, an enterprise or a manufacturing network.

These trends have their implications also to automation business. Instead of building familiar DCS and PLC applications, the suppliers of manufacturing equipment and control systems must be able to provide more comprehensive solutions including, for example, information management, optimisation, remote diagnostics, maintenance and continuous process improvement. In addition to a physical control system, the customer expects to get long-term services that have a clear added value to the business. Rapidly changing technologies and standards, and the requirements for low cost, short delivery times, safety and overall quality make the design, implementation and maintenance of control systems a demanding task.

In our vision, the application of modern information and communication technologies (ICT) in industrial automation provides a good basis for the future. This is, in fact, already taking place. Most controllers and data communication solutions are based on general-purpose hardware and software. Typical software tools, such as relational databases and object-oriented programming, cover an increasing portion of control applications. However, there is a conceptual gap between the two disciplines. Control applications are based on traditional paradigms like variables, signals and function blocks, while software engineering has produced a multitude of new architectures, programming models and design methods.

We suggest here an application model, in which function blocks are enriched by the common interaction mechanisms found in computer programming. This results in the idea of *automation components*, that are active, distributed entities interacting through their data, event and service ports. In addition, the content-based publish-subscribe model and event filters are used for the propagation of alarms and event notifications within the whole system. Application design is carried out without programming by linking ports of components and by assigning values of components' properties. The design should be independent of the allocation of components to physical platform nodes. However, various Quality of Service (QoS) attributes, such as the required latency, should be available to the application engineer.

To be useful, automation components need an appropriate runtime environment. Therefore, the control platform provides a *component container*, where components can be installed and managed. The control platform itself is typically composed of commercially available hardware and software products, such as PCs, databases and communication middleware. Consequently, the pieces of system software can also be treated as automation components. The result is a two-layer software architecture, in which a general framework first defines the rules for writing and composing the control platform from existing *platform components*. At the second layer, one or more *application component containers*, that are platform components themselves, provide an operating environment to the actual *application components*.

In other words, we suggest that two component-oriented framework standards should be developed; one less restrictive for platform components, and another for strictly managed application components. This would enhance the reuse of software and pave the way for open markets of automation components. In such a situation, component developers would have a better opportunity to profit from their development efforts. Control system manufacturers, on the other hand, would have an easier way to acquire and use external software in their product development. In addition, components would allow the incorporation of only the required features into each platform node, an essential issue for small, resource-limited devices. For the end-users, the benefits of the component architecture would mean better quality and functionality, lower prices and independence of a single platform manufacturer.

There are obstacles in the way to this goal. Technically, we have only been able to outline the basic concepts of a future platform, and many details should still be agreed upon. We have made certain design decisions, but also totally different solutions to the same problems could be imagined. In addition, adopting new solutions is an investment, where the existing software and current installations must be considered. Even if the commonly used proxy pattern is a way to include existing applications to a new environment, it takes time to change the current design practices. This is an issue that we have totally omitted, so far. Ensuring the compatibility of independently developed components and their version can also be a challenge. Finally, as has been the case in open control systems, the competition between vendors can delay or even prevent the introduction of new standards and technologies.

However, our goal has not been to give exact solutions but to find possible directions for the future. In this sense, we have hopefully laid a starting point for further discussions. There are a number of obvious development needs, for example:

- Plug&Play features and automatic reconfiguration
- support for redundant functions, data sources, devices and communication channels

- new paradigms and tools for component-based application engineering
- applications of loosely coupled message-oriented systems in sensor networks and geographically distributed automation systems
- integration of security services into each controller node
- integration of control systems to service-oriented business applications
- implementation technologies for low-cost resource-limited applications.

References

ANSI/ISA-88.00.01-1995. Batch Control Part 1: Models and Terminology.

ANSI/ISA-95.00.01-2000. Enterprise-Control System Integration Part 1: Models and Terminology.

ANSI/ISA-95.00.02-2001. Enterprise-Control System Integration Part 2: Object Model Attributes.

Berge, J. 2004. Fieldbus, Ethernet and the reality of convergence. The Industrial Ethernet Book, November 2004, issue 23, pp. 10–14. http://ethernet.industrial-networking.com/ieb/articledisplay.asp?id=37. [31.1.2005.]

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. 2001. Design and Evaluation of a Wide-Area Event Notification Service. ACM Transactions on Computer Systems, Vol. 19, No. 3, August 2001, pp. 332–383.

CEA 2004. http://www.cl.cam.ac.uk/Research/SRG/opera/projects.

Coulouris, G., Dollimore, J. & Kindberg, T. 2001. Distributed Systems, Concepts and Design. Harlow, England: Pearson Education Limited.

ELVIN 2004. http://elvin.dstc.edu.au/index.html.

Estrem, W. A. 2003. An evaluation framework for deploying web services in the next generation manufacturing enterprise. Robotics and Computer Integrated Manufacturing, 19 (December 2003), pp. 509–519.

Fayad, M., Schmidt, D. & Johnson, R. 1999. Building Application Frameworks – Object-Oriented Foundations of Framework Design. New York: John Wiley & Sons Inc. 664 p.

Fowler, M. & Scott, K. 1997. UML distilled: applying the standard object modeling language. Reading, Mass.: Addison Wesley Longman.

Henning, M. & Vinoski, S. 1999. Advanced CORBA Programming with C++. Addison Wesley.

IBM 2004a. Web Services Eventing (WS-Eventing). August 2004. Available at: http://www.ibm.com/developerworks/webservices/library/specification/ws-eventing/. [3.3.2005.]

IBM 2004b. Web Services Notification (WS-Notification). March 2004. Available at: http://www.ibm.com/developerworks/library/specification/ws-notification/. [11.2.2005.]

IDA 2001. White Paper. [18.4.2001.]

IDA 2004 Interface for Distributed Automation. http://www.ida-group.org/default.htm.

IEC 60848 2002. GRAFCET specification language for sequential function charts.

IEC 61131-3 2003. Programmable controllers – Part 3: Programming languages. Ed. 2.0.

IEC 61131-8 2003. Guidelines for the application and implementation of programming languages, Ed. 2.0.

IEC 61588 2004. Precision clock synchronization protocol for networked measurement and control systems.

IEC 61499-1 2005. Function blocks – Part 1: Architecture.

IEC/TS 61804-1 2003. Function blocks (FB) for process control – Part 1: Overview of system aspects.

IEC/TR 61131-8 2003. Programmable controllers – Part 8: Guidelines for the application and implementation of programming languages.

IEC PAS 62030 2004. Digital data communications for measurement and control – Fieldbus for use in industrial control systems – Section 1: MODBUS® Application protocol Specification V1.1a – Section 2: Real-Time Publish-Subscribe (RTPS) Wire Protocol Specification Version 1.0.

ISA-TR99.00.01-2004. Security Technologies for Manufacturing and Control Systems. ISA – The Instrumentation, Systems, and Automation Society. 82 p.

ISA-TR99.00.02-2004. Integrating Electronic Security into the Manufacturing and Control Systems Environment. ISA – The Instrumentation, Systems, and Automation Society. 90 p.

Jaakkola, L. 2005. Applying Service-Oriented Architecture to Geographically Distributed Industrial Information Systems. Master's thesis, Helsinki University of Technology, Telecommunications Software and Multimedia Laboratory. 92 p.

JINI 2004. Jini(TM) Technology Core Platform Specification. http://java.sun.com/products/jini/2.0/doc/specs/html/event-spec.html.

John, K.-H. & Tiegelkamp, M. 2001. IEC 61131-3: Programming industrial automation systems – Concepts and programming languages, requirements for programming systems, aids to decision-making tool. Berlin: Springer Verlag. 376 p.

Kuikka, S. 1999. A batch process management framework – Domain-specific, design pattern and software component based approach. VTT Publications 398. Espoo: VTT Technical Research Centre of Finland. 215 p. http://www.vtt.fi/inf/pdf/publications/1999/P398.pdf.

Lewis, R. 2001. Modelling distributed control systems using IEC 61499. IEE Control Engineering Series 59. London: The Institution of Electrical Engineers. 192 p.

OASIS 2004. Web Services Security v 1.0 (WS-Security 2004). March 2004. Available at: http://www.oasis-open.org/specs/. [7.3.2005.]

OCI 2002. OCI: The TAO Developers Guide. OCI 2002.

OIS 2005. Objective Interface Systems. http://www.ois.com/. [20.1.2005.]

OMG 2002. The Notification Service. http://www.omg.org/cgi-bin/doc?formal/2002-08-04. [21.10.2002.]

OMG 2005. OMG Data Distribution Service v.1.0. http://www.omg.org/technology/documents/formal/data_distribution.htm.

OPC 2003. OPC XML-DA specification 1.0. OPC Foundation, July 2003.

OSGi 2003. OSGi Service Platform, Release 3, March 2003. http://www.osgi.org/osgi_technology/download_specs.asp?section=2.

Peltola, J. 2002. Distributed control systems – Execution environment of a component based automation application. Helsinki University of Technology, Information and Computer Systems in Automation, Report 6. (In Finnish.). http://www.automationit.hut.fi/index.php?group=Labra&target=julkaisut.

PLCopen 2004. http://www.plcopen.org. [30.12.2004.]

PNO 2002. PROFInet Technology and Application – System Description. Karlsruhe: PROFIBUS User Organization e.V. (PNO). 26 p.

PNO 2003. PROFInet Architecture Description and Specification, version V 2.01, August 2003.

RTI 2002. NDDS User's Manual Version 3.0., February 2002. 330 p.

RTI 2004 Real-Time Innovations. http://www.rti.com.

Sandoz, P., Pericas-Geertsen, S., Kawaguchi, K., Hadley, M. & Pellegri-Llopart, E. 2003. Fast web services. Sun Microsystems, July 2003. Available at: http://java.sun.com/developer/technicalArticles/WebServices/fastWS/. [17.1. 2005.]

Schlimmer, J. (ed.) 2004. Devices profile for web services. August 2004. Available at: http://msdn.microsoft.com/library/en-us/dnglobspec/html/devprof.asp. [21.3.2005.]

Schwab, C. 2004. Looking behind the automation protocols. The Industrial Ethernet Book, November 2004, Issue 23, pp. 16–19. http://ethernet.industrial-networking.com/ieb/articledisplay.asp?id=38. [12.1.2005.]

SIENA 2004. http://www.cs.colorado.edu/users/carzanig/siena.

Sierla, S. 2003. Middleware solutions for automation applications – Case RTPS. Helsinki University of Technology, Information and Computer Systems in Automation, Report 9. http://www.automationit.hut.fi/index.php?group=Labra&target=julkaisut.

Szyperski, C. 1998. Component Software – Beyond Object-Oriented Programming. New York: ACM Press. 411 p.

Thales 2005. Thales Nederland. http://www.thales-nederland.nl. [20.1.2005.]

Tommila T., Ventä, O. & Koskinen, K. 2001. Next Generation Industrial Automation – Needs and Opportunities. VTT Automation, Automation Technology Review 2001, pp. 34–41. http://www.vtt.fi/tuo/projektit/ohjaava/ohjaava1/atr_2001.pdf.

W3C 2004. W3C Web Services Architecture Working Group. Web Services Architecture. Working Group Note, W3C, February 2004. Available at: http://www.w3.org/ TR/ws-arch/. [4.3.2005.]

Webb, J. & Reis, R. 2003. Programmable Logic Controllers – Principles and Applications (5th ed.). Prentice Hall. 460 p.

Ventä, O. 2005. Research View of Finnish Automation Industry. VTT Industrial Systems, Industrial Systems Review 2005, pp. 48–54.

Wilkes, S. 2004. Business logic & data services: Loosen up. BEA Systems, Inc. http://dev2dev.bea.com/technologies/soa/businesslogic/articles/bizlogic_wilkes.jsp. [28.1.2005.]

YANCEES 2004. http://awareness.ics.uci.edu/~rsilvafi/yancees/.

Published by

Series title, number and report code of publication

VTT Research Notes 2303
VTT–TIED–2303

Author(s)
Tommila, Teemu, Hirvonen, Juhani, Jaakkola, Lauri, Peltoniemi, Jyrki, Peltola, Jukka, Sierla, Seppo & Koskinen, Kari

Title

Next generation of industrial automation
Concepts and architecture of a component-based control system

Abstract

Recent trends in automation are characterised by geographical distribution and functional integration. On the technical level, the goal is to easily connect devices and software components from different vendors. Functionally, there is a need for interoperability of control functions on different hierarchical levels ranging from field equipment to process control, operations management and various Internet-based service applications.

This research report discusses current technological trends and outlines a new, component-based control system platform that supports the reuse of both system and application software. The working hypothesis is that next generation control systems will be a combination of new information technology and the domain-specific concepts found in process automation. The application area is industrial automation, including both process industries (continuous and batch) and discrete manufacturing. In addition, similar applications in other areas, such as environmental monitoring or distributed energy production, have been kept in mind. As this report is an outcome of a research project, the focus lies on concepts and system architectures rather than on hardware issues. To make it more useful however, also selected new implementation technologies have been described.

The discussion starts from business and technological trends and their implied requirements for future control systems. Then we introduce the conceptual model of a component-based control platform. Only general ideas of their implementation are given. Based on these concepts, the application needs and middleware architectures related to local-area distribution are analysed. Also a review of a few Ethernet-based communication standards is included. Next, the domain is extended to geographically distributed applications. This emphasises the role of remote monitoring, data-acquisition, and business-level services. In terms of implementation technology, service-oriented architectures and communication over public networks become more important. This report also describes a prototype implementation that was developed to clarify the suggested concepts and to evaluate some technical solutions. The final section sums up the key findings and lists areas that would need further research.

Keywords
industrial automation, control architecture, component-based control systems, local-area distribution services, information technology, domain-specific concepts, process automation, discrete manufacturing, systems architecture, implementation

Activity unit
VTT Industrial Systems, Tekniikantie 12, P.O.Box 1301, FI–02044 VTT, Finland

ISBN
951–38–6726–9 (soft back ed.)
951–38–6727–7 (URL: http://www.vtt.fi/inf/pdf/)

Project number
G3SU00709

Date	Language	Pages	Price
July 2005	English	104 p.	C

Name of project
Monitieteellisesti hajautettu monitorointi- ja ohjausjärjestelmä OHJAAVA3

Commissioned by
Tekes, Metso Automation Oy, Raute Precision Oy, Vaisala Oyj

Series title and ISSN
VTT Tiedotteita – Research Notes
1235–0605 (soft back edition)
1455–0865 (URL: http://www.vtt.fi/inf/pdf/)

Sold by
VTT Information Service
P.O.Box 2000, FI–02044 VTT, Finland
Phone internat. +358 20 722 4404
Fax +358 20 722 4374

VTT TIEDOTTEITA - RESEARCH NOTES

VTT TUOTTEET JA TUOTANTO - VTT INDUSTRIELLA SYSTEM - VTT INDUSTRIAL SYSTEMS

2240 Jarimo, Toni. Innovation Incentives in Enterprise Networks. A Game Theoretic Approach. 2004. 63 p. + app. 3 p.

2243 Ventä, Olli. Älykkäät palvelut -teknologiatiekartta. 2004. 71 s. + liitt. 11 s.

2250 Sippola, Merja, Brander, Timo, Calonius, Kim, Kantola, Lauri, Karjalainen, Jukka-Pekka, Kortelainen, Juha, Lehtonen, Mikko, Söderström, Patrik, Timperi, Antti & Vessonen, Ismo. Funktionaalisten materiaalien mahdollisuudet lujitemuovisessa toimirakenteessa. 2004. 216 s.

2251 Riikonen, Heli, Valkokari, Katri & Kulmala, Harri I. Palkitseminen kilpailukyvyn parantajana. Tuotantopalkkauksen kehittämismenetelmät vaatetusalalla. 2004. 67 s.

2254 Nuutinen, Maaria. Etäasiantuntijapalvelun haasteet. Työn toiminta- ja osaamisvaatimusten mallintaminen. 2004. 31 s.

2257 Koivisto, Tapio, Lehto, Taru, Poikkimäki, Jyrki, Valkokari, Katri & Hyötyläinen, Raimo. Metallin ja koneenrakennuksen liiketoimintayhteisöt Pirkanmaalla. 2004. 33 s.

2263 Pöyhönen, Ilkka & Hukki, Kristiina. Riskitietoisen ohjelmiston vaatimusmäärittelyprosessin kehittäminen. 2004. 36 s. + liitt. 9 s.

2264 Malm, Timo & Kivipuro, Maarit. Turvallisuuteen liittyvät ohjausjärjestelmät konesovelluksissa. Esimerkkejä. 2004. 90 s. + liitt. 4 s.

2265 Alanen, Jarmo, Hietikko, Marita & Malm, Timo. Safety of Digital Communications in Machines. 2004. 93 p. + app. 1 p.

2269 Mikkola, Markku, Ilomäki, Sanna-Kaisa & Salkari, Iiro. Uutta liiketoimintaa osaamista yhdistämällä. 2004. 65 s.

2271 Häkkinen, Kai. Alihankintayhteistyö konepajateollisuudessa ja sen laadun arviointia. 2004. 64 s. + liitt. 17 s.

2277 Kondelin, Kalle, Karhela, Tommi & Laakso, Pasi. Service framework specification for process plant lifecycle. 2004. 123 p.

2283 Lemström, Bettina, Holttinen, Hannele & Jussila, Matti. Hajautettujen tuotantolaitosten tiedonsiirtotarpeet ja -valmiudet. 2005. 62 s. + liitt. 10 s.

2284 Valkonen, Janne, Tommila, Teemu, Jaakkola, Lauri, Wahlström, Björn, Koponen, Pekka, Kärkkäinen, Seppo, Kumpulainen, Lauri, Saari, Pekka, Keskinen, Simo, Saaristo, Hannu & Lehtonen, Matti. Paikallisten energiaresurssien hallinta hajautetussa energiajärjestelmässä. 2005. 87 s. + liitt. 58 s.

2287 Wahlström, Björn, Kettunen, Jari, Reiman, Teemu, Wilpert, Bernhard, Maimer, Hans, Jung, Juliane, Cox, Sue, Jones, Bethan, Sola, Rosario, Prieto, José M., Martinez Arias, Rosario & Rollenhagen, Carl. LearnSafe. Learning organisations for nuclear safety. 2005. 58 p. + app. 7 p.

2289 Laakso, Pasi, Paljakka, Matti, Kangas, Petteri, Helminen, Atte, Peltoniemi, Jyrki & Ollikainen, Toni. Methods of simulation-assisted automation testing. 2005. 59 p.

2303 Tommila, Teemu, Hirvonen, Juhani, Jaakkola, Lauri, Peltoniemi, Jyrki, Peltola, Jukka, Sierla, Seppo & Koskinen, Kari. Next generation of industrial automation. Concepts and architecture of a component-based control system. 2005. 109 p.

Tätä julkaisua myy	Denna publikation säljs av	This publication is available from
VTT TIETOPALVELU PL 2000 02044 VTT Puh. 020 722 4404 Faksi 020 722 4374	VTT INFORMATIONSTJÄNST PB 2000 02044 VTT Tel. 020 722 4404 Fax 020 722 4374	VTT INFORMATION SERVICE P.O.Box 2000 FI–02044 VTT, Finland Phone internat. + 358 20 722 4404 Fax + 358 20 7226 4374

ISBN 951-38-6726-9 (soft back ed.)
ISSN 1235-0605 (soft back ed.)

ISBN 951-38-6727-7 (URL: http://www.vtt.fi/inf/pdf/)
ISSN 1455-0865 (URL: http://www.vtt.fi/inf/pdf/)