Project Acronym: MEDIS

Project Title: A Methodology for the Formation of Highly Qualified Engineers at Masters Level in the Design and Development of Advanced Industrial Informatics Systems

Contract Number: 544490-TEMPUS-1-2013-1-ES-TEMPUS-JPCR

Starting date: 01/12/2013 **Ending date:** 30/11/2016

Deliverable Number: 2.4.7

Title of the Deliverable: AIISM teaching resources - Industrial Networks and Fieldbuses – Hierarchical Control

Task/WP related to the Deliverable: Development of the AIISM teaching resources - Industrial Networks and Fieldbuses

Type (Internal or Restricted or Public): Internal

Author(s): Luis Almeida, Mario de Sousa

Partner(s) Contributing: FEUP

Contractual Date of Delivery to the CEC: 30/09/2014

Actual Date of Delivery to the CEC: 30/09/2014

Project Co-ordinator

Company name :	Universitat Politècnica de València (UPV)
Name of representative :	Houcine Hassan
Address :	Camino de Vera, s/n. 46022 - Valencia (Spain)
Phone number :	+34 96 387 7578
Fax number :	+34 96 387 7579
E-mail :	husein@disca.upv.es
Project WEB site address :	http://medis.upv.es/

Context

WP 2	Design of the AIISM-PBL methodology
WPLeader	Universitat Politècnica deValència (UPV)
Task 2.4	Development of the AIISM teaching resources - – Industrial Networks and Fieldbuses – Hierarchical Control
Task Leader	UP
Dependencies	UPV, MDU, TUSofia, USTUTT, UP

Author(s)	Luis Almeida, Mario de Sousa
Contributor(s)	
Reviewers	

History

Version	**Date**	**Author**	**Comments**
0.01	2014-04-14	Luis Almeida	Lecture 7.1
0.02	2014-04-20	Mario de Sousa	Lecture 7.2
0.03	2014-05-05	Mario de Sousa	Labs and Mini-project
1.00	2014-09-19	Mario de Sousa	Final version

Table of Contents

1 Executive summary

WP 2.4 details the learning materials of the Advanced Industrial Informatics Specialization Modules (AIISM) related to the Industrial Networks and Fieldbuses.

The contents of this package follows the guidelines presented in the Partner's documentation of the WP 1 (Industrial Networks and Fieldbuses)

1. The PBL methodology was presented in WP 1.1
2. The list of the module's chapters and the temporal scheduling in WP 1.2
3. The required human and material resources in WP 1.3
4. The evaluation in WP 1.4

During the development of this WP a separate document has been created for each of the chapters of the Industrial Networks and Fieldbuses Module (list of chapters in WP1.1). This document is for the seventh chapter – Hierarchical Control.

In this document, sections 2 and 3 define the lectures for weeks 14 and 15 respectively, sections 4 and 5 describe the laboratory work for weeks 14 and 15, sections 6 and 7 explain the seminar topics for weeks 14 and 15, and sections 8 and 9 define the requirements to fulfill for the mini-project during weeks 14 and 15. Section 10 lists the bibliography and the references.

2 Lecture – Week 14

Industrial Communication Architectures

2.1 Problem statement

This lecture addresses the problem of defining a suitable architecture for industrial systems that supports the integration of all heterogeneous equipment therein, from the shop floor to the management. It will identify the different flows of information and their characteristics, in particular with respect to time constraints and amount of information per transaction. This will show the need for a layered architecture with different technical solutions for each level. The lecture will end with a brief survey of related standards and solutions offered by the manufacturers of current industrial communication systems.

2.2 *Goal*

Allow the students to understand the information flows present in integrated industrial systems and from there to understand their communications architecture. The students will then learn about the attempts that were made to come up with such architecture and the difficulties that stroke some related standardization efforts. In the sequel, the students will learn about several integrated architectures that have been proposed by different industrial communications equipment manufacturers.

2.3 Contextualization

The global context of the topic addressed in the lecture is that of industrial systems and their internal communication architectures to achieve full integration. This will allow embracing the context of related standardization efforts as well as the context of suppliers of industrial communications equipment and their integrated solutions. This will allow understanding why proprietary solutions have been the most common approach followed in that realm.

2.4 Motivation

The snapshot of an industrial system represented in Fig. 1 that we have already shown in the introduction to this module already gives the correct idea of the level of heterogeneity of the equipment present in these systems. The top level includes systems like management, provisioning, sales, design, etc, which typically carry out access to large databases, transfer of large files, access to external systems and the Internet. Communication transactions involving transfer of a few megabytes are common. The level below handles cell configuration, high level coordination, maintenance, production reporting, etc. At this level we find device configuration files, operation reports, alarms logging, etc, which are still reasonably sized files despite smaller, say several kilobytes. The level that follows is that of programmable controllers that carry out the control of large machines and systems, exchanging feedback control information, typically set points and status. This level is then extended to a myriad of devices including sensors, actuators and other controllers, exchanging sensing and actuation signals.

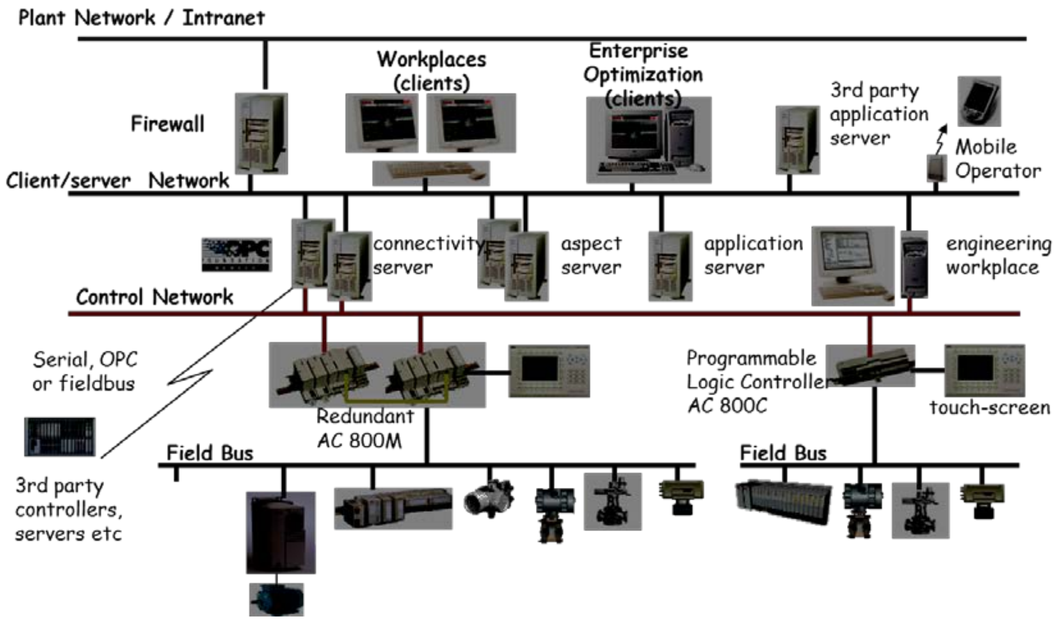

Figure 1. An industrial system as a complex composition of heterogeneous computing elements interconnected with data networks.

Beyond the different kinds of equipment in Fig. 1 there may also exist multimedia equipment, particularly cameras, used both for process and machine feedback control as well as for supervision and surveillance, exchanging images with different resolutions and sizes.

Naturally, Fig. 1 is already arranged so as to highlight the different levels. Moreover, as we referred above, these levels exchange different amounts of information per transaction, from megabytes on the top level to few bytes, or even bits, at the lower level. Moreover, the frequency of such transactions varies in the opposite way, from sparse interactions above, typically at human interaction scale, to very frequent real-time exchanges of process or machine feedback control. Fig. 2 shows the typical types of data exchanges found in industrial systems organized in a bi-dimensional space of data per transaction versus tolerated delay.

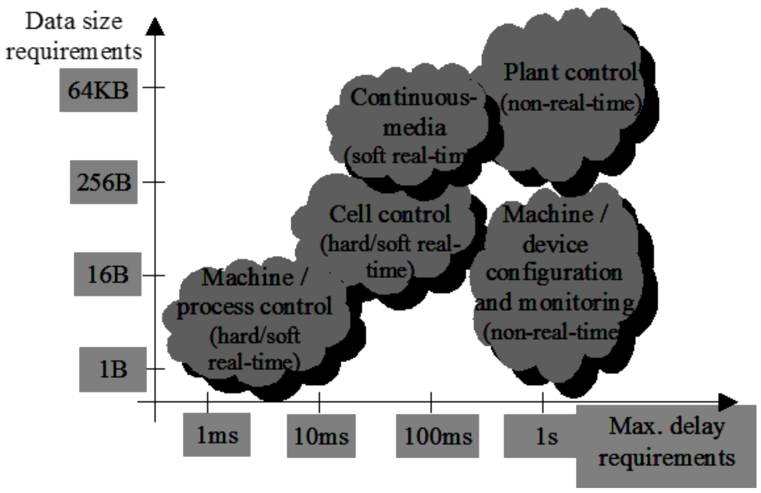

Figure 2. Data per transaction versus tolerated delay of industrial information flows.

These information flows could also be classified in two major types according to their destinations, namely *horizontal* and *vertical*. The former occur within the same level while the latter occur across levels. The *horizontal flows* need to be supported by appropriate communication technologies that allow meeting respective transactions size and time constraints. The *vertical flows* need appropriate gateways to connect across the different levels and allow the respective information to pass through. These different communication technologies are already represented in Fig.1 with different types of networks used at each level, from plant networks on the top level to the fieldbuses on the bottom.

The early works on defining this layered architecture considered different numbers of levels. For example, the authors in (Pleinevaux, P. and J.D. Decotignie, 1988) considered three levels, namely level 2, called factory level, level 1 called cell level and level 0 called sensor/actuator level. However, the most common pattern, presented in (CIM, 1989) is an architecture arranged in five levels (Fig. 3), later also renumbered from 0, lower, to 4, higher. However, in documents dealing with higher abstraction models of management, organization and control of industrial systems it is common to find references to four levels, only, where level 0 referring to the production process is omitted or considered together with level 1. Independently of the number of levels considered, the duality between size of transactions and real-time constraints is similar, as also expressed in Fig. 3 on the right.

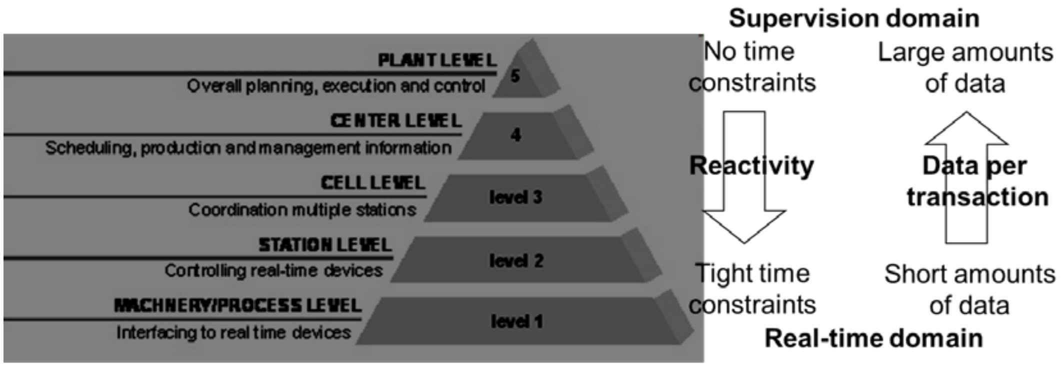

Figure 3. The different levels in industrial systems.

Because of the diversity of requirements found in all the levels in industrial systems, many suppliers of industrial communications equipment started to provide proprietary technical solutions that were not interoperable. This situation was undesirable from the users point of view, reducing equipment options and market competition, and requiring more specialized training, thus incurring extra costs.

Therefore, in 1980 General Motors led a group of north-american companies to create the *Manufacturing Automation Protocol* (MAP) project that aimed at standardizing the architecture of industrial systems. This project was also motivated by the recent OSI standardization efforts and the Internet Protocol development. The project eventually led to the definition of the MAP protocol in 1982. In 1986, it was used by Boeing in its *Technical Office Protocol* that became known as *MAP/TOP*. Despite adoption by several manufacturers, MAP never really gained a market share that could support its sustained development and it was son after abandoned. In fact, MAP specification changed several times in this time lapse, and not always with retro-compatibility. Its link interfaces were considered expensive and the link protocol chosen, the IEEE 802.4 token-bus, was substantially slower and more complex than Ethernet (IEEE 802.3) that was gaining momentum at the time. Adding to these difficulties, it was inadequate for the lower levels, due to large overhead. A reduced version called Mini-MAP was still developed as an attempt to cater for the timing requirements of the lower levels but without commercial success either.

The MAP initiative was, in fact, an initial push for achieving *Computer Integrated Manufacturing* (CIM), addressing some of the involved challenges, particularly the integration of equipment from multiple suppliers, the assurance of data integrity which is crucial in automated processes, and support for process control and supervision. However, a crucial component for efficient process control was the fieldbus. The first fieldbuses appeared in the 1980s in Europe and were standardized at the national level, e.g., FIP in France (C 46-601 to 607), P-Net in Denmark (DS 21906) and PROFIBUS in Germany (DIN 19245-1 to 3). The 1990s saw a strong movement towards the definition of an international standard with a unified single protocol. However, such unification was never achieved due to multiple non-technical reasons (Thomesse,1998).

The referred three fieldbuses were joined in 1996 as different profiles in the EN 50170 European standard. In 1998, another European standard, the EN 50254, was defined aiming at high efficiency communication subsystems for small data packages for the lower automation levels, including Interbus-S, PROFIBUS-PA and Device WorldFIP. Finally, the main international standard was defined in the end of 1999, the IEC 61158, including a proposal for a unified protocol that became known as Foundation Fieldbus H1 plus ControlNet, PROFIBUS, P-Net, Fieldbus Foundation High Speed Ethernet, SwiftNet, WorldFIP and Interbus-S.

In a different thread, the Controller Area Network was developed in the early 1990s and standardized within ISO in the mid-1990s. It was then included in the fieldbus arena by different manufacturers in different profiles. Two notable cases were CANopen by CAN-in-Automation (CiA) and DeviceNet by Open DeviceNet Vendors Association (ODVA).

In the sequel of all this activity, despite the large diversity of options, two main integrated communications architectures emerged, which became the most common support for deploying CIM. One is called *Totally Integrated Automation*, by Siemens, and supported by the PROFIBUS & PROFINET International (PI) association (Fig. 4). This architecture includes PROFIBUS-DP and PROFIBUS-PA for direct digital control in machines and process control respectively, plus Ethernet technology in two main profiles, PROFINET-CBA (Component-Based Automation) based on the distributed Function Blocks of the IEC 61499 standard to design and commission industrial automation systems, and PROFINET-IO that connects to peripheral devices using the producer-consumer model.

While PROFINET-CBA uses standard TCP/IP, supporting cycle times in the range of 100ms, PROFINET-IO can either use the Real-Time (RT) protocol based on standard switched Ethernet technology with priorities, thus supporting cycles time in the range of 10ms, or the Isochronous Real-Time (IRT) protocol that uses enhanced switches supporting very precise cycles (sub-microsecond jitter) with periods in the range of 1ms. IRT, which applies TDMA over switched Ethernet, is particularly tailored for very fast digital control, such as for high speed servos.

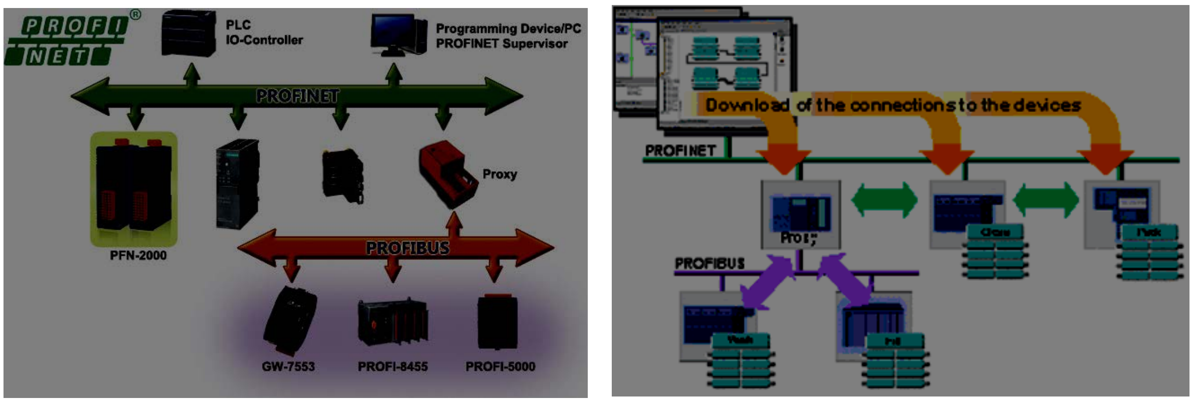

Figure 4. Totally Integrated Automation by Siemens, based on PROFINET and PROFIBUS.

The other contender is the *Common Industrial Protocol* (CIP), supported by ODVA and particularly Rockwell Automation. CIP was initially known as Control and Information Protocol. It defines an Application Layer based on an Object Library with specific application profiles, e.g., motion, valves, safety, synchronization, configuration and information, and which maps on a common producer-consumer messaging system that provides unified access to manufacturing information.

This system maps transparently on any of a set of network technologies that natively support the producer-consumer model, which is central in CIP, and cover all the communication needs of industrial enterprises (Fig. 5). These technologies include Ethernet/IP at the highest level, where IP stands for Industrial Protocol and which is based on standard Ethernet technology and essentially UDP communication, ControlNet for the manufacturing and cell levels, relying on a cyclic master/multi-slave approach also referred to as centralized TDMA, DeviceNet based on CAN as well as CompoNet based on TDMA, for the lower levels of machine control and very fast IOs, respectively.

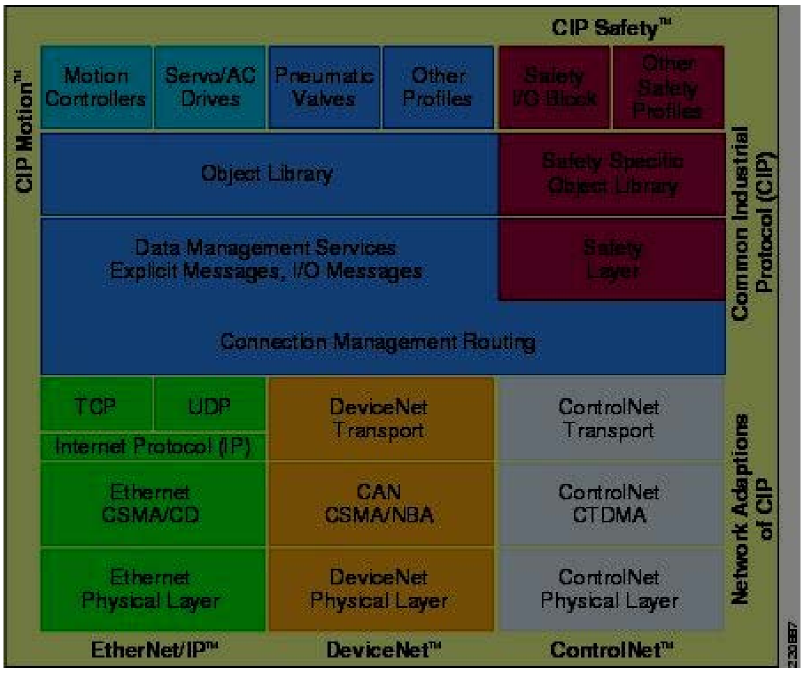

Figure 5. The Common Industrial Protocol promoted by ODVA.

These integrated communication architectures, among other similar ones, became instrumental to realize *Computer Integrated Manufacturing* (CIM). CIM is essentially a manufacturing method proposed by the *Computer and Automated Systems Association* and the *Society of Manufacturing Engineers* (CASA/SME) in the 1980s. The motivation was to connect functional areas in the upper business-related levels, e.g. design, planning, accounting, provisioning and distribution, with the lower production-related levels for direct production control and monitoring. This was essential to support emerging concepts such as *flexible manufacturing*, i.e., prompt factory reconfiguration to either produce different products or in different quantities, and *zero stock*, i.e., tightly controlling the provisioning and production according to the distribution minimal needs, which were key for high efficiency.

In the 1990s in Europe, the AMICE consortium developed the CIMOSA initiative towards an Open Systems Architecture for CIM to facilitate CIM adoption by European companies. The CIMOSA architecture aimed at supporting both enterprise modeling and integration using vendor-independent standardised CIM modules.

In 1995, ISA approves the ISA-88 standard that aimed at creating a universal model for controlling batch processes that was vendor-independent and facilitated both the expression of user requirements as well as the control of batch operations. This standard, updated in 2010, was another contribution to the CIM framework representing a design philosophy for describing equipment, and procedures in the process control layer, thus contributing to interoperability.

Basically, the standard allows defining the *process model*, as well as the *physical model*, *procedures*, and *recipes*, all as hierarchical abstractions. The process model describes the *process* as an ordered set of *process stages*, each consisting of an ordered set of *process operations* which in turn consist of an ordered set of *process actions*. The physical model begins with the *enterprise* that must contain a *site* possibly composed of *areas*, possibly containing *process cells*. Each of these cells must contain a *unit* with *equipment modules* that may contain *control modules*. The procedural control model consists of *recipe procedures* made of an ordered set of *unit procedures* which consist of an ordered set of *operations*, in turn formed by an ordered set of *phases*. Finally, there are four types of recipes: *general, site, master, control,* all including a *header, formula, equipment requirements, procedure,* and possibly other recipe-related information.

In the same trend and approximately the same period, ISA launches the ISA-95 standard for developing an automated interface between enterprise and control systems, in an attempt to reduce the gap that was then growing between the business-oriented systems of CIM level 4 and the execution-oriented systems that controlled the production plant, particularly at CIM level 3. The standard provides a terminology that sets a common ground for both suppliers and manufacturers, providing information and operations models that clarify the application functionality and information use.

The ISA-95 standard has 5 parts. Part one provides standard terminology and object models that help setting the boundary between the enterprise and control systems, for example defining the tasks that should be executed by each function and the information that needs to be exchanged. Part two defines the attributes of the objects defined in part one. Part three focuses on the functions and activities related to level 3, the production plant control level, and gives guidelines for describing in a standard way the production level of an enterprise so that it can be compared with others.

Part four, which defines the object models that determine the information exchanged between the production plant control activities defined in part three, and part five that defines the operation between office and production systems, are still under development.

As shown in Fig. 6, ISA-95 provides means to define in a standard way the functions carried by the production plant control, typically called *Manufacturing Execution System* (MES), as well as the interface between the MES and the upper level business system, typically called *Enterprise Resource Planning* (ERP). Fig. 6 also shows the relative position of the ISA-88 standard in the CIM architecture, providing definitions for lower production control levels.

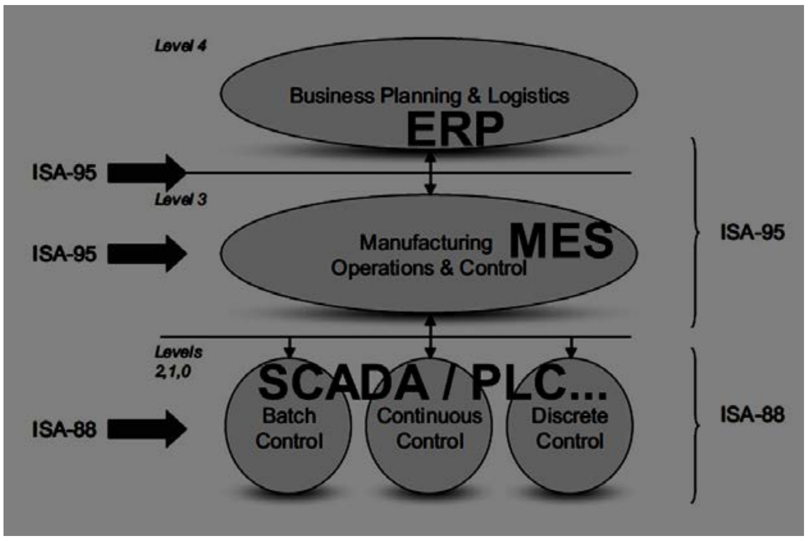

Figure 6. The standard ISA-95 and ISA-88 in the CIM architecture.

2.5 Bibliography

- Richard Zurawski (ed.), The Industrial Information Technology Handbook. November 29, 2004 by CRC Press. ISBN 978-0-8493-1985-3
- Jean-Baptiste Waldner (1992), Principles of Computer-Integrated Manufacturing, John Wiley & Sons, ISBN 0-471-93450-X
- Dennis Brandl (2006), "Design Patterns for flexible manufacturing" (ISA 88). ISBN-13: 978-1556179983
- Bianca Scholten (2007), "The road to integration - A guide to applying the ISA-95 standard in manufacturing"

2.6 Concepts

- Hierarchical organization of industrial processes
- Vertical integration
- Flexible manufacturing
- Zero-stock / Just-in-time
- Integrated communications architectures
- Computer integrated manufacturing
- Manufacturing execution system
- Enterprise resource planning
- Standard interfaces across CIM levels

2.7 Examples

- Totally Integrated Automation
- PROFIBUS & PROFINET International
- Common Industrial Protocol and ODVA
- ISA-88
- ISA-95

2.8 Control Questions

- How does the size of information per transaction vary when going from the upper business levels of a production plant down to the shop floor?
- How do the time constraints of common transactions vary when going from the upper business levels of a production plant down to the shop floor?
- Why many communication standards appeared in the early applications of ICT to industrial systems?
- What was the motivation to develop a unified communications standard?

- What is an integrated communications architecture? Give examples.
- What is Computer Integrated Manufacturing? And which connection has it with the integrated communications architectures?
- What was the problem addressed by the ISA-88 standard? And by ISA-95?
- What is the relationship between both standards, ISA-88 and ISA-95, and to which parts of the CIM architecture do they refer to?

2.9 Recommended Further Reading

- Kalpakjian, Serope; Schmid, Steven (2006), Manufacturing engineering and technology (5th ed.), Prentice Hall, p. 1192, ISBN 978-7-302-12535-8.
- A. de Toni and S. Tonchia, Manufacturing Flexibility: a literature review International Journal of Production Research, 1998, vol. 36, no. 6, 1587-617.
- Thomesse J.P. (1998). The Fieldbuses. *Annual Reviews in Control*, **22**: 35-45.
- AMICE Consortium (1991), Open System Architecture, CIMOSA, AD 1.0, Architecture Description, ESPRIT Consortium AMICE, Brussels, Belgium.
- CIM, A reference model for computer integrated manufacturing from the viewpoint of industrial automation, International Journal of Computer Integrated Manufacturing, Vol. 2 (2), 1989.
- Pleinevaux, P. and J.D. Decotignie (1988). Time Critical Communication Networks: Field Buses. *IEEE Network,* **2(3)**: 55-63.

3 Lecture – Week 15

OPC – Open Process Control

3.1 Problem statement

This lecture addresses the problem of defining a suitable communication protocol to transfer process data between layers 2 and 3 of the ISA 95 model. Although this data transfer typically does not have tight timing constraints, it may require the transfer of more complex data structures. This is in contrast to the communication protocols used inside layers 1 and 2 of ISA 95 model, with strong real-time constraints and very simple data structures (for example CAN Open and Modbus).

3.2 Goal

Allow the students to understand the information flows present between layers 2 and 3 of the ISA 95 model, and to become acquainted with one specific protocol widely used in this application area. The students will then learn about the evolution of this 'standard' - how it started out using proprietary middleware architectures, and why it has recently evolved to use more open software layers and data transfer standards.

3.3 Contextualization

The global context of the topic addressed in the lecture is that of industrial systems and their internal communication architectures to achieve full integration. More specifically, the exchange of information between the upper layer ERP and MES systems, and the lower control and supervision layers .

3.4 Introduction

As was shown in fig. 1 of the previous week's lecture, typical network architectures in an industrial setting include separate networks for each control layer. Layer1 uses fieldbuses, layer 2 typically uses Ethernet or a variant of Ethernet specific for industrial setting (e.g. ProfiNet), while layer 3 usually uses standard Ethernet.

Data flow within layer 3 is accomplished using very many communication protocols, many of them proprietary to the software tools used at this level (MES, ERP, Database Managers – ex. Oracle, etc.). Sometimes standard file transfer protocols are used (ftp, SMB, http, etc...), but the files themselves use proprietary formates (Autocad, …).

However, these upper layer tools need to be able to send manufacturing orders to the lower layers, and also to have access to the current state of the production process. This data transfer is based on data structures that are much simpler than those commonly used inside the upper layers themselves (Autocad drawings, ...), but still more complex than those used by the lower layers (boolean actuators/sensors, 16 bit analog values, …). Data structures for this data transfer may commonly include simple structured variables (like C structs), and arrays of limited length. Another good example of data exchange at this interface is that of recipes (as defined in ISA 88) to be produced in a batch process plant (ex. beer production).

In this scenario many different software applications from different vendors need to exchange data. In order for them to be inter-operable each software would need to implement several communication interfaces and protocols, and then simply hope that each pair of software packages shared a common interface. This was (and still is) typical of SCADA packages (Supervisory Control And Data Acquisition) that need to exchange data directly from PLCs (Programmable Logic Controllers). Very many competing SCADA packages exist in the market place, and the same may be said of PLC vendors. Since each PLC insists on supporting (or giving preference to) a limited number of lower level communication protocols , each and every SCADA package is forced to include drivers to inter-operate with all the competing PLCs and their communication protocols.

In the late 1990's this was recognised as a problem that needed addressing, and a some vendors (ex. Rockwell, Intellution, Siemens, …) got together to specify a common method of exchanging data between software packages at this intermediate level. At this time in history Microsoft dominated the OS (Operating System) market, so all of the SCADA and related software packages (ex. PLC programming environments) were developed for the Windows OS. It was therefore natural that the these companies came up with a data exchange mechanism based on Microsoft technologies. This became known as OPC.

3.5 OPC and DCOM

Unlike the communication protocols studied up to this point in this module, did not start out as a communication protocol. OPC is actually a set of software interfaces, commonly known as an API (Application Programming Interface). When programming in C this would be something similar as to the header (.h) file for a specific library, that contains all the functions that an application may use to ask for services from that library.

OPC therefore defined a set of functions that an OPC server would have to offer, so that OPC clients could call those services (functions). Typically through these functions an OPC client is able to query the server as to what process variables/data is contains, and to read and write those same variables. This is somewhat similar to what is achieved in a communication protocol, with the distinction that now both the server and the client are actually two independent programs running on a windows OS.

The mechanism used to transfer the function invocations between one program and the other uses a proprietary Microsoft technology. This started out using the OLE (Object Linking and Embedding) mechanism, and very quickly evolved evolved through a series of technologies, all promoted by Microsoft: DDE (Dynamic Data Exchange), followed by COM (Component Object Model), and soon after by DCOM (Distributed COM). And here it stayed for several years.

The COM technology introduced an object oriented approach, where one program (the client) could call a method in an object residing inside another program (the server). Whith COM both programs had to be running on the same computer, but DCOM introduced the capability of doing remote invocations over a computer network. It therefore became possible to have one client program running on one computer, invoking functions to read and write data values residing on a program running on another computer (the server). It is for this reason that OPC is often confused with a communication protocol as it can be used to exchange data over a network. However, the OPC standard itself only specifies the invocation interface between programs. The communication protocol itself is the one used by DCOM, and is proprietary. Implementation of OPC over any other OS is therefore rather difficult, and solutions are difficult to come by.

OPC version version 2, based on DOCM, remained unchanged for several years, and therefore its used became widespread throughout the industry. Nowadays a new version of OPC, known as OPC-UA (OPC Unified Architecture) is slowly starting to take hold. However since OPC v2 is still very widely used, we will come back to OPC-UA later on, and focus on OPC v2 for the moment.

OPC is a relatively slow communication medium, especially when transferring data between different computers. Even for data within the same computer, the overhead introduced by the DCOM middleware is significant, as is the overhead required by the complexity of the OPC interfaces themselves. Do not expect to be able to transfer data at the rate of tens of milliseconds. Even hundreds of milliseconds may be problematic when large volumes of data need to be exchanged.

3.6 OPC v2

OPC is actually a set of interfaces, each set designed for a specific purpose:

- OPC-DA: Data Access
- OPC-HDA: Historical Data Access
- OPC-DX: Data eXchange
- OPC-AE: Alarms and Events
- OPC-Batch

OPC-DA is by far the most often used interface. This interface allows a client application to read and write process variables residing in an OPC server. If desired, the client may also use this interface to query the server about the available process variables.

A variant of OPC-DA has also been introduced, whose communication is based on the transfer of XML files over the Http communication protocol.

OPC-HDA defines an OPC server that is capable of storing historical values of some process data. In other words, instead of merely reading the current state of a variable, the server will also store all the previous states this same variable had in the past. This information may be passed to the client, who may then display graphs showing how the trend of the process variable. Actually, what the client does with the data is irrelevant to the OPC-HDS specification.

OPC-DX is very similar to two OPC-DA clients connected back-to-back. Notice that OPC relies on a client/server interaction model. When using this model, two servers cannot exchange data directly with each other. OPC-DX is actually a program with two OPC-DA client interfaces. This program is configured to connect to two distinct OPC-DA servers, and periodically transfer data between the two.

OPC-AE provides an interface to a server program that handles alarms or events. An event is typically defined by the fact that a boolean condition becomes true (for example, the water level in a tank reaches the 'full' value). An alarm is a special event that usually needs to be acknowledged by an operator, as it may be dangerous to the plant or the workers (for example, pressure in a boiler surpasses the maximum value). An OPC-AE server allows a client to define the conditions that specify the occurrence of an event or an alarm. The server will monitor which events have been disabled (pressure no longer above maximum), the events that have been acknowledged by an OPC-AE client, and will keep a log of the time of each of these occurrences (when alarm becomes active, disabled, acknowledged, etc.).

OPC-Batch is focused towards providing services for batch processes using the recipe/machine design pattern defined in ISA 88. A batch process is a process that cannot be produced continuously, but due to limitations of the physical process must be done in batches (example, baking a cake must be done one or several at a time). ISA 88 introduced the concept of recipes that each batch process needs to follow, and the machines that execute the recipe. OPC-Batch provides a method of reading and writing the parameters for each specific batch that needs to be executed, as well as the results of the execution of each batch.

3.7 OPC-DA

OPC-DA relies on a data model which oriented towards the individual process variable. Each variable stored in an OPC-DA server has a unique name, and will have one of several datatypes (boolean, byte, word, dword, short, long, float, double, char, array of {byte, word, dword, short, long, float, double}).

When OPC started to be adopted, an OPC-DA server was a much too heavy process to run directly on a PLC. Therefore traditional OPC-DA servers typically run on a PC running Microsoft Windows, and these obtain the values of the process variables using whatever fieldbus network that the PLCs happen to support. In these cases, whenever an OPC-DA client reads a variable from the OPC-DA server, what it actually gets is the current cached value of that variable. It is up to the OPC-DA server to periodically poll the PLC to keep the cache as fresh as required. Nowadays some PLCs may be considered full fledged PCs, and some even run Micrsoft Windows. In these cases then we can expect the data stored in the OPC-DA server to be kecp more current to the actual process value.

All process variables in a OPC-DA server are organized into an hierarchical tree structure (similar to files inside directories of a file-system). How these variables are organized in the tree is completely up to the person configuring the OPC-DA server. Variables may be grouped by their physical location in the plant (example all variables for a boiler inside the same sub-directory), or they may be grouped by their type (for example, all variables representing a temperature in the same sub-directory).

An OPC-DA client can contact the OPC-DA server and browse this tree of process variables. Each variable can also be tagged with information regarding the engineering datatype it contains (pressure in mbars, atmosphere, psi, etc.), although this is not often used.

An OPC-DA client cannot request to read a single process variable directly. It must first ask the OPC-DA server to create a group in which to store variables (known as an OPCGroup object). Although the OPCGroup object resides in the OPC-DA server, it is still reserved for the OPC-DA client that requested its creation. It cannot be shared with other OPC-DA clients connecting to the same OPC-DA server.

Once the OPCGroup is created, the OPC-DA client will request that references to some variables be added to the group (*references*, and not copies). Any variable in any location of the variable tree hierarchy can be placed in any group. OPC-DA clients will then be able to simultaneously read or write all the variables in the group.

An OPC-DA client can even request the creation of several groups. For example, a group of variables that need to be updated at a 5 second interval, and another group that is updated only every minute. The same variable can even be referenced by multiple groups of the same OPC_DA server.

The OPC-DA specification defines two methods of writing to the variables in a process group:

– synchronous
– asynchronous

The synchronous method behaves like a standard function call. The function call only returns once the data from the OPC-DA server is available. In the asynchronous method the OPC-DA client first sends the server a request to write the data, and the function call to send the request returns immediately. Once the OPC-DA server confirms the request was processed correctly, the client program will be interrupted and its execution will jump to a callback function where any further processing related to the write may be done.

When reading data from an OPCGroup, a new method becomes available:

– synchronous
– asynchronous
– subscription-based

The synchronous and asynchronous methods have similar semantics as when writing data. The subscription-based method consists of the OPC-DA client asking the OPC-DA server to notify the client of the new variable values when some specific condition is met. The notification process follows a similar mechanism to that of asynchronous calls – the normal execution of the client program will get interrupted when a notification arrives, and a specific callback function is executed.

Several types of conditions may be specified for defining when to send the notifications. For boolean or integer variable, it is possible to request the notification to be sent whenever any change in these variables occur. For integer and float variables it is possible to define a maximum dead-band inside which the variable may change without a notification being triggered. Notifications are only sent once the variable changes to value outside of the dead-band. The dead-band is usually defined as a percentage of the full range of the variable, and is centered around the last value sent to the client.

All of the previous conditions may be considered event triggered. A time-triggered approach is also supported, where the client can define a maximum subscription refresh rate. In this case the server will attempt to keep the client updated with fresh values, at this specified maximum frequency. It also means that the OPC-DA server should try to get the data for its internal cache at a rate equal to or faster than the specified rate.

3.8 OPC-UA

When Microsoft introduced Windows 7, it also introduced the .Net technology which was intended to supersede DCOM. OPC, with a very large installed base, suddenly became outdated and with future support compromised. Unlike the previous times in which this occurred (DDE → COM → DCOM) in which backward compatibility was guaranteed, this time the supporters of OPC decided to take a new road and to adopt open standards, therefore achieving full platform independence.

The new OPC-UA (Unified Architecture) was therefore created as a replacement for the DCOM based OPC, and is now considered an official international standard - IEC 62541. It contains several parts, and covers all of the functionality provided by all the variations of traditional OPC (DA, HDA, AE, …).

3.9 OPC-UA Data Model

In OPC-UA a server will typically contain many variables (that contain data) and many objects (as in object-oriented programming). For OPC-UA, variables and objects are actually just different versions of the same thing: a node.

All nodes (variables or objects) will contain:

– attributes
  (all available properties are defined in the OPC-UA standard)

– properties
  (that can be added by the developer of the OPC-UA server that contains the node)

– references
  (basically point to any other node)

Attributes define the characteristics of the node, i.e. they specify information about the node (what does the node do, what is it about, ...), but all possible attributes are listed in the OPC-UA standard. Properties serve a similar purpose to the attributes - they provide semantic information about the node (for example, a long name for the node, a description, engineering units, conditions when the information contained in the node is no longer valid, etc.), but are different to attributes in the fact that the OPC-UA developer is free to add whatever properties required to fully describe the node.

References point to other nodes, including nodes residing in other OPC-UA servers. Nodes referencing each other can therefore form a mesh of nodes (be they objects or variables). An OPC-UA client can 'browse' and follow these references, just the same way as a classical OPC-DA client can 'browse' the hierarchical tree of variables in an OPC-DA server. Although in OPC-DA these references can also form hierarchical trees, complete mesh networks are also possible. Since references can point to nodes in other OPC-UA servers, the mesh networks can span multiple servers.

Objects are typically created to represent a physical device. It can represent very small devices (a single sensor, a pump), to very large (a processing cell, a complete manufacturing plant). There is actually no limit to the size of an object.

Objects will reference other objects (for example, a cell object will point to the objects representing the machines inside that cell). Eventually objects will reference variables that will contain the data.

Objects can also contain methods that can be invoked, and triggered events that will be transmitted to a client. An OPC-UA client can call (invoke) a method of an object residing in an OPC-UA server. These methods can be used to read the attributes of the object, and also to read the object references. Reading the object references is actually the same as 'browsing' through the mesh of objects.

Each object instance residing in an OPC-UA server will be of a specific class, or object type. The collection of methods of an object type or class defines an interface through which objects of that type/class may be accessed. This constitutes a SOA architecture (Service Oriented Architecture). All remote method invocations are asynchronous, and clients are therefore viewed as asking the invoked object to provide a service. However, all invocations still follow the client/server paradigm, where eventually a called method will reply to each invocation.

Besides the object and the variable node types, more specialized node types exist. For example, there is a node type that describes a datatype. In other words, a node can describe the datatype (or class) of a specific object type. Actually, all objects (nodes) of this specific datatype will always contain a reference to that node that describes the object's (node's) datatype. This means that the OPC-UA data model is fully reflexive – any client can fully determine how every object is built, and the methods it supports, but browsing through the node that describes its datatype. This is why it is said that OPC-UA clients written today, will be able to work with OPC-UA servers written in the future, and that contain objects whose datatype does not yet exist.

The services of traditional OPC (DA, HDA, AE, etc...) are supported in OPC-UA as a specific set of objects of a specific class (or type). A profile defines a collection of objects types, as well as the object instances that an OPC-UA server must instantiate in order to be able to say that it supports that profile. The DA, Hda, AE, and other services are therefore simply defined as profiles in OPC-UA. An OPC-UA client can browse the OPC-UA server to determine which profiles (and object types) it supports.

3.10 OPC-UA Communication Protocols

Notice that the object interfaces (collection of methods it includes) are defined independently of the mechanism used to transmit the data associated with a method invocation. Two mappings for this transmission have currently been defined. One mapping uses XML, SOAP and HTTP to encode and transmit the method invocations. This is relatively slow, as the parsing of XML and the transmission of data over the stateless HTTP protocol involves significant overhead. As a faster alternative, a direct binary mapping over TCP has also been defined.

The binary protocol works over a single TCP connection (single ip:port address), and is therefore tunneling through firewalls is easy to configure . The web service protocol uses standard HTTP and HTTPS ports.

If a client OPC-UA program supports both protocols, the person configuring this client will only recognize which protocol is being used by the URL to the server. The binary protocol uses the URL 'opc.tcp:servername', whereas the web service protocol uses 'http:servername'. All other services work identically whether they work over one protocol or the other.

Both these communication protocols support:

– heartbeat for connections in both directions

(both client and server can recognize connection interruptions)

– buffering of data

– acknowledgments of transmitted data

3.11 Bibliography

- Richard Zurawski (ed.), The Industrial Information Technology Handbook. November 29, 2004 by CRC Press. ISBN 978-0-8493-1985-3
- OPC Foundation, Data Access Custom Interface Standard - Version 3.00, Released March 4, 2003
- OPC Foundation, OPC Unified Architecture, Prototype Development Specification, Part 1: Concepts,
- OPC Foundation, OPC Unified Architecture, Prototype Development Specification, Part 3: Address Space Model

4 Lab – Week 14

The objective of this lab session is to allow the student to become acquainted with the very basics of using a SCADA package (Supervisory Control And Data Acquisition). These applications are commonly used to develop an upper layer graphical application through which it is possible to supervise the current status of the plant. Many commercial SCADA packages exist in the market, however in this lab the open source ScadaBR software will be used instead.

This package should be pre-installed in the lab computers before the lab session.

Procedures:

Open the ScadaBR software.

In the main menu, choose the option “DataSources” .

Choose the Modbus/TCP option, and configure a connection to the plant floor simulator (IP 127.0.0.1, port 5502). Give your DataSource a name. When you save this DataSource, the options to add Data Points become enabled.

Click “Adicionar” and add data points to access the sensors in the plant floor simulator. Give each DataPoint (commonly called a 'tag', or 'variable') a name.

Now activate all your DataPoints by clicking the icons , and activate the DataSource by clicking on .

Once you have launched the plant floor simulator (with its embedded Modbus/TCP server), you should now be able to see the status of the sensors by opening the “Watch list”.

To create a graphical view of these sensors, start by creating a graphical page view. On the main menu, choose “Representação Gráfica”. Click on “Nova represent ção” .

To choose an image for the background, click on “Escolher arquivo”, select an image, and confirm by clicking “Fazer upload de imagem”.

Now add an image whose representation will change, depending on the state of a DataPoint. On the component list choose “GIF binário”, and choose an image as well as the DataPoint.

You should now be able to visually see the status of the sensor on the plant floor simulator, by simply looking at the graphical representation.

5 Lab – Week 15

During this week's seminar the student should have come across servel OPC servers that are freely available (for example, http://www.opcconnect.com/freesrv.php). Some of these are servers that act as a gateway to devices using the Modbus communication protocol (for example, https://www.matrikonopc.com/downloads/54/drivers/index.aspx).

For this lab session the student is requested to download and install an OPC-DA server that can obtain information through a Modbus connection. Use this OPC server to make the data on your Arduino (through the Modbus/RTU protocol) or the plant floor simulator (through the Modbus/TCP protocol) available over OPC.

Use a freely available OPC client to browse and read the data. For example,

https://www.matrikonopc.com/products/opc-desktop-tools/opc-explorer.aspx

Have the underlying data (buttons on the Arduino, sensors on the plant floor simulator) change with varying frequency (speed), and try to determine whether the OPC protocol is sufficiently fast to detect all these changes. At what speeds does it start failing to detect the changes in the input data?

6 Seminar – Week 14

Students must perform a survey of commercial ERP (Enterprise Resource Planning), MES (Manufacturing Execution System), or any other software package that support any of the data and integration models referenced in the lecture (ISA 95, ISA 88, …).

7 Seminar – Week 15

Some groups of students should search on the Internet for commercial products supporting OPC. For each product, determine:

- whether it is an OPC server, or OPC client
- if it is OPC-UA or classic OPC
- what services (classic OPC - DA, HDA, AE) or profiles (OPC-UA – DA, HDA, …) it supports
- what the client does with the information obtained from the server

(is it for a SCAD system, for a controller, …)

- where a server obtains the data stored inside

(does it get it from a PLC, a database, ...)

- The license of the product
- If possible the price of the product

Other groups of students should search on the Internet for code or libraries that either implement or may be used to implement OPC servers and/or clients. For each software package, try to determine:

- whether it is an OPC server, or OPC client
- if it is classic OPC or OPC-UA, and in this last case what communication protocols are supported

- what services, if any, are included in the library

(classic OPC - DA, HDA, AE) or (OPC-UA – DA, HDA, …)

- what programming language it is written in
- the license in which it is distributed
- the price of the software package

8 Mini-project – Week 14

Using the ScadaBR software package, make a graphical interface that represents the current state of the plant floor. Use as a background image a screen capture of the plant floor simulator window itself.

Test your application, and determine if the response time is sufficiently fast to represent the changes that occur on the plant floor. Try running the graphical interface on a different computer to the one where the plant floor simulator is running.

9 Mini-project – Week 15

This week is reserved for the students to demonstrate their project. If time allows, a full integration of all student projects should also be attempted, by connecting all Arduinos into a single CAN-Open network.

10 References

- Richard Zurawski (ed.), The Industrial Information Technology Handbook. November 29, 2004 by CRC Press. ISBN 978-0-8493-1985-3

- Jean-Baptiste Waldner (1992), Principles of Computer-Integrated Manufacturing, John Wiley & Sons, ISBN 0-471-93450-X

- Dennis Brandl (2006), "Design Patterns for flexible manufacturing" (ISA 88). ISBN-13: 978-1556179983

- Bianca Scholten (2007), "The road to integration - A guide to applying the ISA-95 standard in manufacturing"

- Kalpakjian, Serope; Schmid, Steven (2006), Manufacturing engineering and technology (5th ed.), Prentice Hall, p. 1192, ISBN 978-7-302-12535-8.

- A. de Toni and S. Tonchia, Manufacturing Flexibility: a literature review International Journal of Production Research, 1998, vol. 36, no. 6, 1587-617.

- Thomesse J.P. (1998). The Fieldbuses. *Annual Reviews in Control*, **22**: 35-45.

- AMICE Consortium (1991), Open System Architecture, CIMOSA, AD 1.0, Architecture Description, ESPRIT Consortium AMICE, Brussels, Belgium.

- CIM, A reference model for computer integrated manufacturing from the viewpoint of industrial automation, International Journal of Computer Integrated Manufacturing, Vol. 2 (2), 1989.

- Pleinevaux, P. and J.D. Decotignie (1988). Time Critical Communication Networks: Field Buses. *IEEE Network,* **2(3)**: 55-63.

- OPC Foundation, Data Access Custom Interface Standard - Version 3.00, Released March 4, 2003

- OPC Foundation, OPC Unified Architecture, Prototype Development Specification, Part 1: Concepts
- OPC Foundation, OPC Unified Architecture, Prototype Development Specification, Part 3: Address Space Model