Chalmers University of Technology

Simulation and Control using Supremica

Goran Čengić and Knut Åkesson

Göteborg 2004

Contents

Chapter 1
Introduction

Ever increasing competition forces companies to diversify their product lines. To achieve customizable products the companies require constantly changing and evolving production plants. Such plants are called flexible production plants. The plant is constantly adopted to the products with the goal of production of highest quality products in as little time as possible. To achieve this the software for the automatic control of such a plant has to be developed and upgraded along with the plant. The problem is that when using today established development methods this upgrade is impossible without an interrupt of the production that can last for several weeks in some cases.

To avoid the down time associated with the upgrade of the control software in the production plants a more agile architecture for the control software is needed. To be more specific an object oriented architecture and development method are needed as opposed to the loop based architecture and structured development method implemented with PLCs. The advantages of object oriented software development are reaped across the IT industry for quite some time now. Why it has not been widely adopted in the automatic control software development is for the most part because of the high investment already made in the PLCs making the companies reluctant to invest in yet another expensive technology driven upgrade. In other words, as long as the automation control problems in the existing plants are possible to be solved with PLCs there will be no adoption of object oriented architectures. Another big reason for slow adoption of object oriented development methods is the lack of standard object oriented platform suitable for the automation software applications. The latter reason is being addressed by several vendors but their offerings are proprietary, incompatible and only add-ons to the IEC 61131 standard which defines a non object oriented development platform.

The International Electrotechnical Commission (IEC) is making an effort to standardize a platform for the software of next generation of systems for industrial measurement and control. The standard is named IEC 61499 and it defines the architecture, software tool requirements and application guidelines. The architecture defines object oriented platform for distributable software components and is described in more detail in chapter 2.

The standard defines the common ground for the distributed industrial control applications but it is not enough. To ease the effective and rapid control application development a more user friendly software objects are needed. The kind of objects that will run on any IEC 61499 compatible system but provide a greater support for the task of application developing. That kind of objects are defined in chapter 3.

Chapter 2

IEC 61499 Standard

This chapter,with minor revisions, is copied from chapter 2 in (5).

New technologies and standards are emerging that are set to have a dramatic effect on the design and implementation of industrial control systems. A new standard IEC 61131-3 (3) was introduced in 1995 to standardize the representation of control designs so as to enable cross-vendor integration. The standard offered significant advantages over earlier vendor proprietary control design representations (6).

- It encouraged well-structured procedural programs that allowed the development of different functional elements into separate procedures or program units.
- It provided standardized languages and methods of program execution so that wide range of technological problems can be programmed as vendor-independent software elements.
- It provided support for various languages like structured text, function block diagrams, ladder diagrams, instruction list and sequential function charts.
- The standard enforced strong data typing and supported user defined data structures.

The standard however was based on a scanning mechanism where the program execution was monitored by a continuous loop. Although different parts of the program could be scanned at different frequencies (based on fidelity requirements), it was tedious to describe such behavior using sequential function charts. The standard was based on polling mechanism rather than an event driven system, and hence its execution order is not clearly defined.

Further the standard was designed to support a software model and languages for PLCs whose software was typically running on a single processing resource. The communication between different resources was supported through global variables and communication blocks, both of which did not provide a clear and concise method for defining the connections between distributed systems.

To overcome these problems, a function block model was sought that would support distributed control systems in a scalable, extensible and flexible manner. IEC 61499-1 (4) standard has now been developed for distributed industrial process, measurement and control systems.

In industrial systems, function blocks are now an established concept for defining robust, re-usable software components. A *Function Block* instance is defined to a software function unit comprising an individual, named copy of a data structure and associated operations specified by a corresponding function type.

Function blocks are analogous to objects in object-oriented programming. Control designers can now take standard proven encapsulated functionality in the form of blocks and link it together in the quickest and most intuitive way possible. They reduce the complexity since system integrators can now build large systems using *black-box composition* i.e. without worrying about how the internals of a block work.

The following sections are adapted largely from the various discussions on the features of the standard in (4; 3; 6; 7; 1; 2)

2.1 Reference Models

IEC 61499-1 has provided a set of models for describing distributed systems that have been programmed using function blocks.

System Model

At the physical level, a distributed system consists of a set of co-operating *applications*. An application can exist on a single system such as application C in figure 2.1 or have functionality distributed over a number of devices such as applications A and B in Figure 2.1.

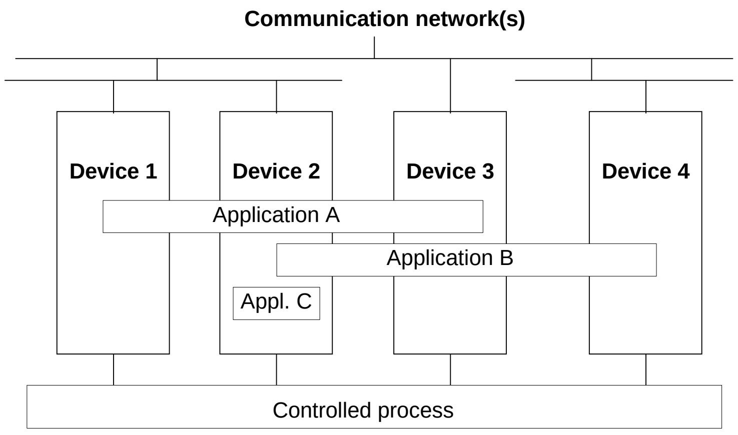

Figure 2.1 *System Model (4)*

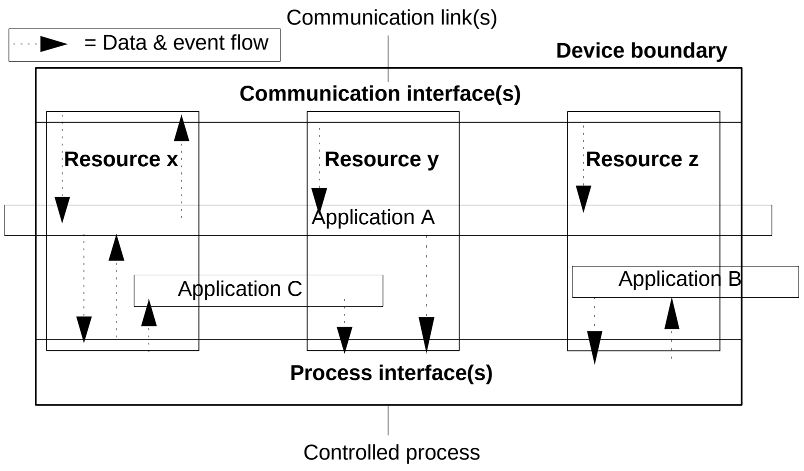

Figure 2.2 *Device Model (4)*

Device Model

A resource provides independent execution and control of networks of function blocks. It supports zero or more *resources* and has at least one interface. The interface could either be a 'process interface' that allows the resources to exchange data with the input and output points on the physical devices, or a communication interface that allows the resources to communicate with external resources in remote devices. The device model is illustrated in Figure 2.2.

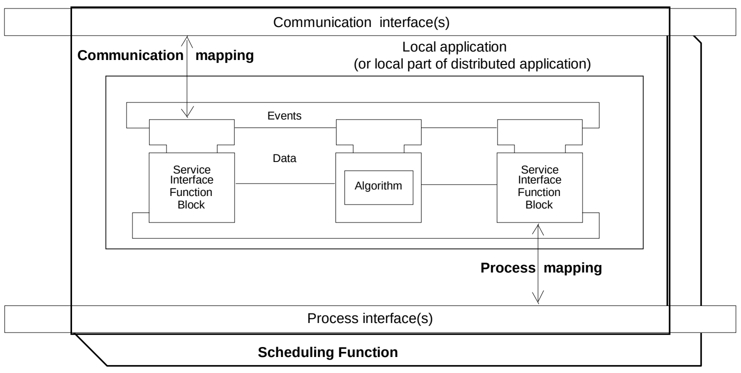
Communication interface(s)
Communication mapping
Local application
(or local part of distributed application)
Events
Data
Service Interface Function Block
Algorithm
Service Interface Function Block
Process mapping
Process interface(s)
Scheduling Function

Figure 2.3 *Resource Model (4)*

Resource Model

A resource is an independent functional unit, contained in a device which has independent control of its operation. The resource can be created, configured, parameterized, started up, deleted etc. without effecting other resources. Each resource provides an interface to the communication systems and tot the device specific process. The resource is therefore, concerned with the mapping of data and event flows which pass between function blocks in the local resource to remove resource function blocks via the device communication interfaces.

Figure 2.3 depicts the main features of an IEC 61499 resource. Within the resource, it shows a network of interconnected function blocks linked by data and event flows. A scheduling function ensures that the events are executed in the correct serial order. *Service Interface* function blocks are a special form of function blocks that provide a link between function blocks and the interface of the resource.

Application Model

An application is defined as a network of interconnected function blocks, linked by data and event flows. An application can be fragmented and distributed over many resources. The resources ensure that the events are executed in the correct serial order based on the causal relationships specified by the applications.

An application could be understood as an entire set of function blocks and interconnections to solve a particular automation control problem. Figure 2.4 illustrates an application, where each individual block is either a function block or a subapplication.

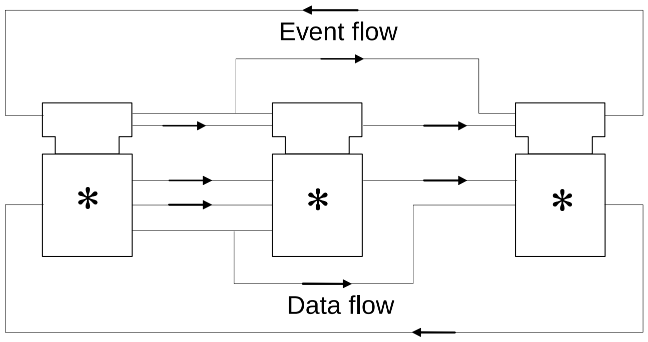

Figure 2.4 *Application Model (4)*

Function Block Model

A function block is a functional unit of software that its own data structure which can be manipulated by one or more algorithms. The formal description of the data structure is provided by the function block type definition.

The main features of function blocks are:

- Each function block has a type name and an instance name.
- Each block ha a set of event inputs, which can receive inputs from other blocks through event connections.
- Each block has one or more outputs, through which it can propagate its results to other blocks.
- There is a set of data inputs that allow values to be passed in from other blocks.
- There is a set of data outputs, through which the block can pass its results to other blocks.
- Each block may have a set of internal variables that hold its state between algorithm invocations.
- The behavior of the block is defined in terms of algorithms and state information. This behavior is defined in the function block type definition.

Figure 2.5 shows the main characteristics of a function block. The top part of the block, called the 'execution control' portion contains in some cases, a state machine that maps events to the algorithms that should be triggered as a consequence of the event. The lower portion of block contains both the algorithms and the internal data, both of which are encapsulated within the block.

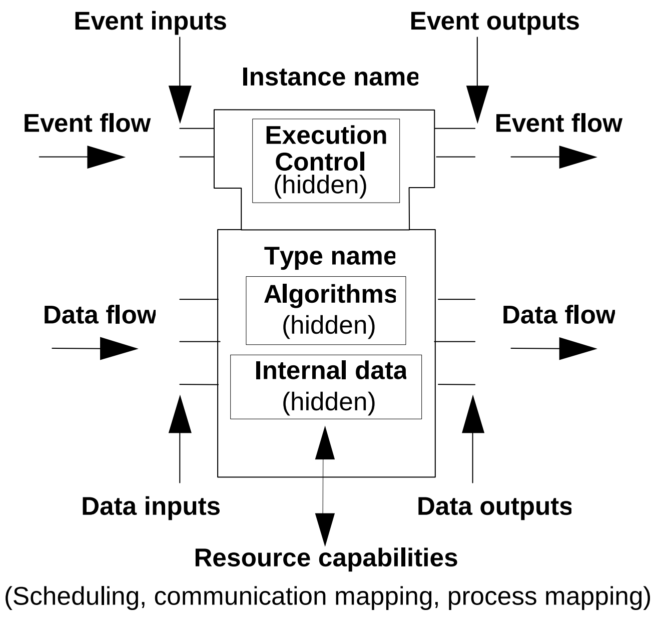

Figure 2.5 *Function Block Characteristics (4)*

2.2 Function Block Types

A function block type is defined by a type name, formal definitions for the input events and data and the output events and data. The type definition also includes the internal behavior of the block. The notion of a function block type is analogous to the class definition in object-oriented programming while function block instances correspond to the object instances of the classes.

Basic Function Blocks

The behavior of basic function blocks is defined in terms of algorithms that are invoked in response to input events. As algorithms execute, they trigger output events to signal their state to other blocks. The basic blocks hold their state in terms of internal variables. Basic function blocks also can communicate their state using input and output data variables.

The mapping of events onto algorithms is expressed using a state transition notation known as execution control chart and is described in the next section. Figure 2.6 shows an example of a basic function block.

Composite Function Blocks

The behavior of a composite function block is described in terms of the event and data connections between its component function blocks. A composite

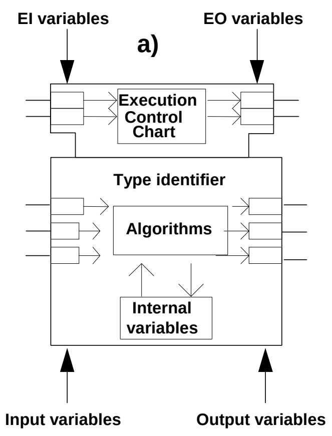

Figure 2.6 *Basic Function Block (4)*

function block can be used to hide the internal interactions between the component blocks and presenting a simplified interface for use by the outside world. A composite function block is illustrated in Figure 2.7.

Service Interface Function Blocks

Service interface function blocks provide an interface between the function block domain and the external services. Since a service interface function block is primarily concerned with data exchange, it is defined using time sequence diagrams. Examples of service interface function blocks are communication blocks for publishing and subscribing over a network and management blocks for dynamically creating component blocks and event connections.

Event Function Blocks

Event function blocks are some special basic function blocks defined in the standard to ease the management of events. These blocks are used for control, generation and detection of events and can be used by other composite blocks. These blocks are defined using an execution control chart or a time sequence diagram. These blocks can be used for the following purposes:

1. To split events to produce new events.
2. To allow events to be merged.
3. To permit the propagation of certain events.

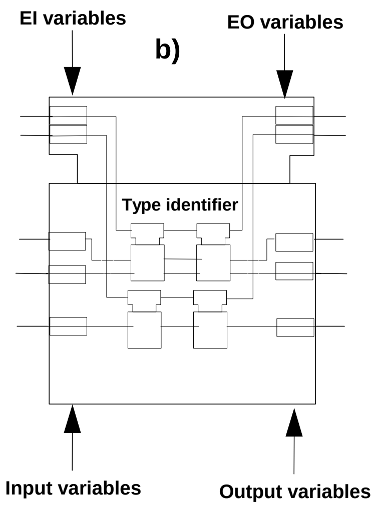

Figure 2.7 *Composite Function Block (4)*

4. To select between two or more events.
5. To delay an event by a given period.
6. To generate a stream of events at a regular interval.

2.3 Execution Model

The execution of algorithms for basic function blocks is invoked by the execution control portion of a function block instance in response to events at event inputs. This invocation takes the form of a request to the underlying scheduling function of the associated resource to schedule the execution of algorithm's operations. The model assumes that the resource in which a function block exists provides a scheduling function that ensures each phase of function block execution occurs in the correct order and with the correct priority.

The execution model specifies eight discrete timing points as shown in Figure 2.8 that must occur sequentially for the basic function block to operate. These are:

t_1 : Values coming from external function blocks as input data are stable at the current block.

t_2 : The event associated with the input values occurs.

t_3 : The execution control function notifies the resource scheduling function that it has the input values and is ready to execute the algorithm.

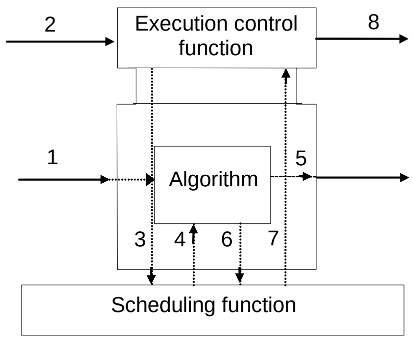

Figure 2.8 *Execution Model (4)*

t_4 : The scheduling function begins the algorithm execution.

t_5 : The algorithm processes input values, (and in some cases, internal variables) to create new output values that are written to the block's outputs.

t_6 : The algorithm notifies the scheduling function that it has completed its execution.

t_7 : The scheduling function invokes the execution control to generate the output events.

t_8 : The execution control generates the appropriate output events at its event output ports. The event may be used by downstream blocks to signal that they may use the output values generated by this block.

To ensure consistency, the algorithms must use the snapshot of input variables that exists when the input event arrived. Further the scheduling function must be able to detect the arrival of input events at a faster rate than the rate at which they can be handled by the block.

Chapter 3
Automation Objects

With the function blocks the main technological problem of distributed automation has been solved. A general framework for development and deployment of distributed automation systems has been established. What is still needed is a way of encapsulating function blocks together with the data about their properties and, more importantly, about their functionality in a way that will enable diminishing of the time-to-market and facilitate effective rapid object oriented development of automation systems. In this chapter the concept of automation objects is presented as far as the research has come. There are still plenty of details to be worked out but this chapter should clarify the role and the benefits of using the automation objects in the development of the industrial, as well as other, automated control systems.

3.1 Background

Considering the following arguments it is clear why such encapsulation is needed and how it can help in achieving the afore mentioned goal.

Object oriented system development as presented in the IEC 61499 standard is ideal for the purpose that it is designed for but not enough for achieving effective software reuse in automation systems development.

Effective software reuse facilitates effective decentralization of software development. If anyone could use the software that somebody else developed without wasting time on resolving the difficulties that arise from poorly documented interfaces, or resorting to reimplementation because of the same problem, than everyone could concentrate on providing the best implementation for their part of the automation system software.

Leveraging effective software reuse and decentralized software development makes integration and deployment much easier. System integrator's task is hereby reduced to picking the right software objects from the library and connecting them for the specific application.

Using the encapsulation of software objects with well specified information about their usage makes it possible to build libraries of elements that can easily be shared over the Internet since everything that is needed for the usage of the elements comes with the elements themselves. This is unlike the Java system where the objects are distributed in binary only format and the interface documentation has to be downloaded or reviewed on-line prior to usage.

The arguments presented are enough to support the idea that the software objects encapsulated with the information about their usage will increase the speed of the development of the automation systems.

3.2 Specification

To realize all of the above mentioned arguments a new kind of software objects are needed, preliminary called *automation objects*. Even though this kind of objects are still very much a research subject the following is the specification of the objects so far.

Automation objects consists of models, controllers, views and configuration registry implemented using IEC 61499 function blocks and their interfaces, see figure 3.1. Models are interchangeable simulation models with varying level of abstraction. One model might be used for the simulation on a high level of abstraction, typically behavioral model, and another for the simulation on a lower level. This is then used to implement dynamic zooming during the simulation. One of the models will implement the communication with the controlled equipment thus allowing effortless switching between simulation and actual control. This will enable gradual verification and on-line tuning of the big automation systems by allowing the simulation of the system components that are being configured or tuned while running the components that are already configured.

Controllers are function blocks that control the model. The best examples of controllers are regulators as PI- or PID-regulator. On the higher level of the automation system these can be configurable implementations of different automation algorithms. This allows interchange of different controlling strategies for easy verification with simulation and the equipment. The automation object can have as many controllers as it is needed but only one controller may be active at any time.

Views are the function blocks allowing user interaction with the automa-

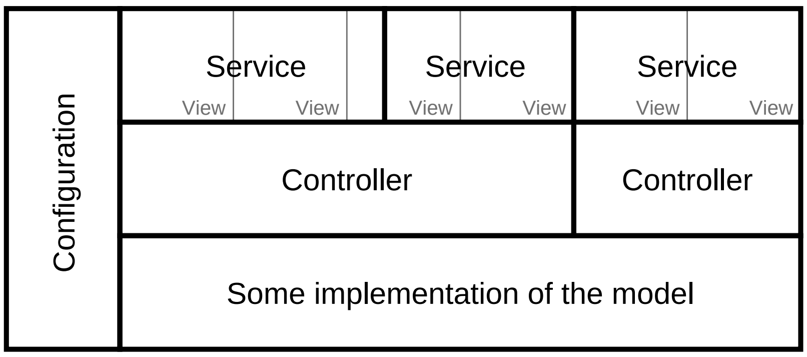

Figure 3.1 *The architecture of the automation object.*

tion object. These implement all the allowable ways of interaction with the automation object. At the lower levels of the automation system they will typically provide various interfaces to the object depending on the user's needs while at the higher levels they will implement the full graphical user interfaces for the operators. As with the controllers there may be as many views as needed for the automation object.

Configuration registry is a function block implementing the configuration database and interface to it. This function block holds all relevant attributes and configuration variables for the automation object and provides an interface for accessing and changing them. Configuration registry is used by models, controllers and the views for storing and retrieving the configuration data. For example, controllers will use this function block to access their regulator parameters and other configurable parameters.

For every automation object a number of services may be defined. A number of views that are needed for a certain task during the development of the control system are specified in the definition of the service thus the service operates on a certain controller. The service can than be invoked during the development and ease the task that it is designed for. One service may only include views that operate on the same controller since only one controller is active at any time. Several services that operate on the same controller may be activated at the same time. On the other hand, services that operate on different controllers are mutually exclusive. During the development of the control system at the highest level of abstraction user of the automation objects is interacting only with services. For example, think of a company that is producing conveyor belts. The company buys motors and produces the belts. With the motor comes the automation object for it from the motor manufacturer. When the motor and the belt are assembled the tuning service of the motor's automation object can be invoked and the motor parameters can be tuned for the conveyor belt. The company also develops an accompanying automation object for the conveyor belt that encapsulates the automation object for the motor and provides a service for the tuning of the conveyor belt when it is installed in some manufacturing process.

All services defined in an automation object are replaceable with user defined services. In case a user is not satisfied with how a service works she can develop new views and define a new service that will replace the service defined in the automation object. The user can also customize existing services by adding views to or replacing views in the defined services. It is also possible to define new services for the automation object by combining existing views in different groups, adding new views or both.

Since the automation objects are built using IEC 61499 function blocks the model, controller and services are distributable among different computing resources. Using this property tuning of the conveyor belt in the earlier example may be done from any computer on the company's network.

Last but not the least, automation objects have a protection mechanism that will hide the implementation of the models, views and controllers from the users of the automation objects.

3.3 Implementation and Support in Supremica

First implementation of the automation object concepts will be done in the Supremica. The architecture for the automation objects will be concurrently implemented with the development tools and functions. In this section implementation considerations are discussed and the development work flow is presented.

An automation object is distributed as compressed archive of XML-files. The archive contains XML definitions of models, views, controllers, registry and a file containing the meta information for the automation object. In the meta information file all the automation object attributes (ie name, creator, version), all of the possible configuration variables, service definitions and interface explanations are stored. In other words everything needed for successful reusage in the development, configuration and execution of the automation object.

It is envisioned that after the initial acceptance period significant amount of automation objects are developed. These are stored in the repositories for easy access. The repositories are non-hierarchical pools of automation objects, probably implemented using database technology. These repositories are preferably shared over the internet. The most important feature of the repositories is the ability for extensive search and sorting by multiple attributes defined in the registry as well as the defined services.

The Supremica tool provides the user interface for the development and the deployment of the automation object applications. The implementation that supports the user interface consists of two cooperating parts. The development environment for the automation objects and the run-time environment for the function blocks.

The DE provides the tools and functions for the repository handling and automation object development. The user is able to choose among the available automation objects from the repository and include them in the application. While working with the objects the user can activate different services or bring up specific views and exchange the models and controllers. User can also connect the views of the different objects in the application. Working in this way the user can build and tune the application to it's liking.

All the time during the development process the run-time environment is active and taking commands from the development environment to add function block types, add instances and connections to composite function block types, to add instances and connections to the application, to configure the resources, to start and stop the execution of the application and so, thus creating the application in the run-time environment as it is developed. These commands are all at the function block level. It is the development environment that transforms the automation object operations into the function blocks running in the run-time environment and thus making all of the automation objects alive throughout the development.

While the user have one of the simulation models activated the run-time is executing the code that simulates the behavior of the hardware that the object is supposed to control. Switching to the model that talks to the process interface issues a command to exchange the function blocks in the run-time and results in the control of hardware instead of the simulation of the control. This gives user the freedom to incrementally develop large systems.

For example, consider a production plant that is connected to the network of control computers. Engineers responsible for each part can develop, tune and test their part on any computer on the network. As each one switches to the model that communicates with the hardware the plant becomes more and more functional until it is fully operational.

To accomplish this the development environment uses the database technology to hold the application data that is shared. While working on the application the engineers can access this data from any computer on the network. The application data includes the distribution configuration governing which automation objects are run by which computers on the network. By issuing the commands to the run-time environment of the computer running the automation object that is currently worked on, the development environment allows the user to work on the whole application from any computer on the network. This means that if something goes wrong or software/equipment needs upgrading anyone that can access the application data can, from any computer on the network, switch the model of the automation object in question to simulation model so that the parts of the plant that don't need the resource being upgraded in the production can continue to function, resolve the problem or perform the upgrade and finally switch back. This feature can also be used for active monitoring, tuning and control of the system by a human operator given that the views implementing the human machine interface are developed.

Bibliography

[1] J.H. Christensen. Design patterns for systems engineering with iec 61499. `http://www.holobloc.com`.

[2] J.H. Christensen. Open distributed automation and control with iec 61499. `http://www.holobloc.com`.

[3] IEC. IEC 61131 programmable conrollers – part 3: Programming languages. Technical report, International Electrotechnical Commission, 1993.

[4] IEC. IEC 61499-1: Function blocks for industrial-process measurement and control systems - part 1: Architecture. Technical report, International Electrotechnical Commission, 2000.

[5] S. Jain. Design and implementation of an electro-mechanical workbench. Master's thesis, University of Illinois, Department of Industrial Engineering, 2002.

[6] R.W. Lewis. *Programming Industrial Control Systems using IEC 1131-3*. The Institution of Electrical Engineers, 1995.

[7] R.W. Lewis. *Modelling Distributed Control Systems Using IEC 61499*. The Institution of Electrical Engineers, 2001.