Bachelor Degree Project

UNIVERSITY OF SKÖVDE

FUNCTION BLOCK ENVIROMENT IN WISE SHOPFLOOR
Graphical user interface

Bachelor Degree Project in Automation
30 ECTS
Spring Term Year

Mikel Anasagasti Alberdi

Supervisor: Bernard Schmidt
Examiner: Lihui Wang

This project is submitted by Mikel Anasagasti Alberdi to the University of Skövde for the Bachelor Degree in Automation Engineering, in the School of Technology and Society.

Date of Submission 14th of June, 2012

I hereby certify that all material in this dissertation which is not my own work has been identified and that no work is included for which a degree has already been conferred on me.

Signature

Mikel Anasagasti Alberdi

Executive summary

Nowadays, accomplishing real time information applications is one of the goals that customers are looking for in the market. Furthermore, two essential features are required in this society to ensure the success of these applications: reusability and interoperability. The new standard of Function blocks [FB] is one efficient way to reach them. This new standard, whose use has considerably increased, consists of a large number of functionalities divided into distributed shorter codes. In fact, that is why function blocks are different from other programming languages: it works with short distributed functionalities interconnected among them. Therefore, for this reason function blocks are used in distributed automation systems. The function block standard is formally specified in IEC 61499, for distributed Industrial Process Measurement and Control Systems (IPMCS). It is worth of mentioning that function blocks design is usually carried out graphically (interconnecting data and events within a FB network), whose clarity and easiness are always significant requirements, as in any visual system.

On the other hand, web-based application development is getting more and more common during the last years because of its efficiency, portability and flexibility. It also provides many advantages to users, in real-time monitoring, real-time manufacturing, or even remote control. These are what companies are looking for in the 21st century. Nevertheless, combining web-based programming with function blocks can be the key to companies' success.

This project consists of **implementing a web-based function block environment for IEC 61499 compliant control systems development**. It would be a proper way to meet the above mentioned customer requirements (efficiency, portability, flexibility, reusability and interoperability). The main objective of the five-month project is to develop a web-based function block environment with a user-friendly graphical interface. The challenges include:

- Carrying out research on Function Blocks and its run-time environment.

- Implementing an easy-to-use graphical interface for creating, saving and editing different types of function blocks (basic FB, composite FB and applications).

- Introducing iconic design and visual presentation of function blocks and connections via drag-and-drop.

- Developing an application to generate eXtensible Markup Language (XML) files, allowing the exchange of files between different tools and devices of different vendors.

This web-based function block environment will be used at Volvo Cars and Sandvik AB as a part of a big research program. It is anticipated that the function block environment will be extended and applied to a real-time decision support system for walking workers where important events or tasks will be distributed to the right person at the right time.

Table of Contents

Table of figures

Table of abbreviations

4DIAC	Framework for Distributed Industrial Automation and Control
ADM	Adaption Decision Making
API	Application Programming Interface
DOM	Document Object Model
DTD	Document Type Definition
ECC	Execution Control Chart
FB	Function Blocks
FBD	Function Block Diagram
FBDK	Function Block Development Kit
HTML	Hypertext Markup Language
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IL	Instruction List
I/O	Input/Outputs
LD	Ladder Diagram
IPMCS	Industrial Process Measurement and Control Systems
OO	Object Oriented
PLC	Programmable Logic Controller
SAX	Simple API for Xml
SFC	Sequential Function Chart
SGML	Standard Generalized Markup Language
ST	Structured Text
URL	Uniform Resource Locator
XML	Extensible Markup Language
XtAX	Streaming API for XML

1. Introduction

In the following pages, a web based function block environment for development of IEC 61499 compliant control systems (function block programs) implementation will be explained. It is an automation degree final-year project, which will mostly be focused on the graphical interface as well as files saving and editing operations of the above-mentioned function block environment.

First of all, it is worth of mentioning that the implementation of the web based environment was already started (an existing Wise ShopFloor framework) at an earlier stage. This thesis work is an extension of the work implemented in Canada (Wang, et al., 2003). The most important improvements applied to the already implemented web based environment, are changes in: the user graphical interface and the capacity of generating XML files when saving a program and also the possibility of opening and editing them. Besides, the implemented program has been created according to the international standard IEC 61499. So, this means that some more enhancements have been applied in the thesis; because, previously, it was not developed according to the international standard IEC 61499.

In addition, note that this automation final year bachelor project titled “Function Block environment in Wise-ShopFloor: graphical user interface” has a direct connection with another parallel automation project (Arrieta, 2012) and two design projects (Transpaderne & Vidal, 2012) (Cervera, 2012). They are small projects of a major research program (developed by Wise-ShopFloor research group) that will be presented at “Volvo Cars” and “Sandvik AB” in the near future. As explained in the executive summary, the research consists of an interface design and development for an information presentation device in order to establish a direct connection between the company and its workers.

As far as this particular thesis is concerned creating a graphical user interface, it has been implemented simultaneously with the other automation bachelor thesis project just mentioned. Its objective is also to apply some other improvements to the already extended thesis; such as, java code generation, algorithms representation or ECC scheme representation. Most of the time during the project development has been shared with the other automation project member, Aitor Arrieta Marcos. Taking into account all the significant similarities between the projects, many ideas and information have been shared and exchanged. Additionally, it is supposed that another different automation thesis (Díaz, 2012) as well, will be joined to this one.

On the other hand and focusing on the implementation of this project, “Java” has been the selected programming language in order to achieve the objectives. Java is an object oriented language which is becoming more and more popular because of its open usability, as it will be explained later. All the generated code has been implemented in NetBeans.

Furthermore, a great deal of research and investigation about Function Blocks and topics related to them (International Standard, FB diagram language, XML, functionalities…) has been performed during the first months of the project.

1.1 Objectives

A web-based Function Block environment for IEC 61499 compliant control systems development should be implemented in this project. The main objective of the five-month thesis work is **developing an integrated graphical user interface for the function block environment**. The interface must be user friendly and easy to use. Furthermore, XML documents generation, reading and editing must be implemented as well.

Working with function blocks in web based environment can be a proper way to meet nowadays customer´s requirements, which are highlighted with some specific features, such as, efficiency, portability, flexibility, reusability or interoperability. Obviously, **real time application**s are related to web based environments. They provide immediate information and that is the main objective of the information presentation device that will contain the developed program: providing real time information. On the other hand, function blocks provide reusability and interoperability, important characteristics for an easy programming and use of the device. Different challenges are included within this particular project:

- Carrying out research on Function Blocks, which is considered the base of the project, and its run-time environment.

- Implementing an easy-to-use graphical interface for creating, saving and editing different types of function blocks (basic FB, composite FB and applications). Easy menus and toolbars must appear in the user interface.

- Introducing iconic design and visual presentation of function blocks and connections via drag-and-drop. It provides an easier use to the operator when manipulating figures in the program.

- Developing an application to generate and read from Extensible Markup Language (XML) files, allowing the exchange of files between different tools and devices of different vendors. Figure 1 shows graphically what ought to be reached.

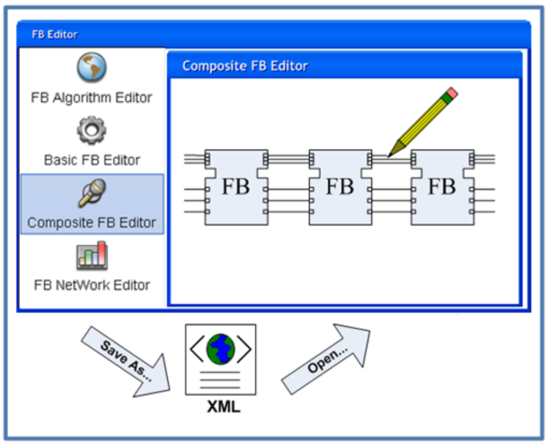


Figure 1: Scheme of the challenges (*project definition, 2012*)

1.2 Motivation

Nowadays speed has become a determining factor in business world. This society asks for fast answers to customer's petitions, such as, fast transports, fast deliveries, fast manufacturing processes, fast run-time programs… The faster these operations are performed, the more successful the company will become. So, obviously, real time working applications, the fastest control system at the moment, are becoming common in today's enterprises.

On the other hand, adaptive decision making, is also introduced into firms with a special aim: decreasing the degree of uncertainty (operator´s sick-leave, emergencies, missing tools, machines´ unavailability…). Adaptive Decision Making (ADM) is based on real time information availability. This means, a closed loop must be formed among sensors, monitors, plans and the control to accomplish an efficient Adaptive Decision Making. Figure 2 shows how an ADM would close the loop. The rest of the thesis is focused on the execution control section (green colored in figure 2) developing a web based Function Block environment for IEC 61499 compliant control systems.

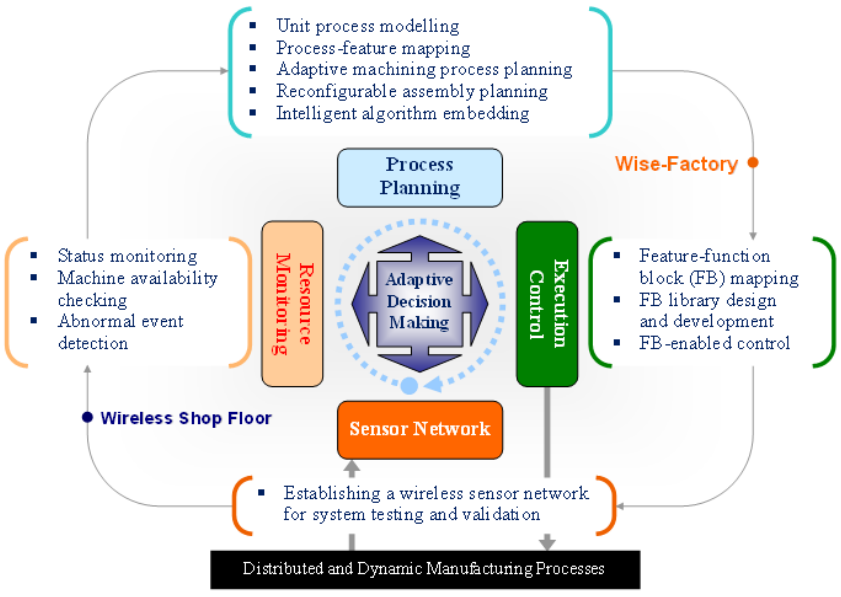
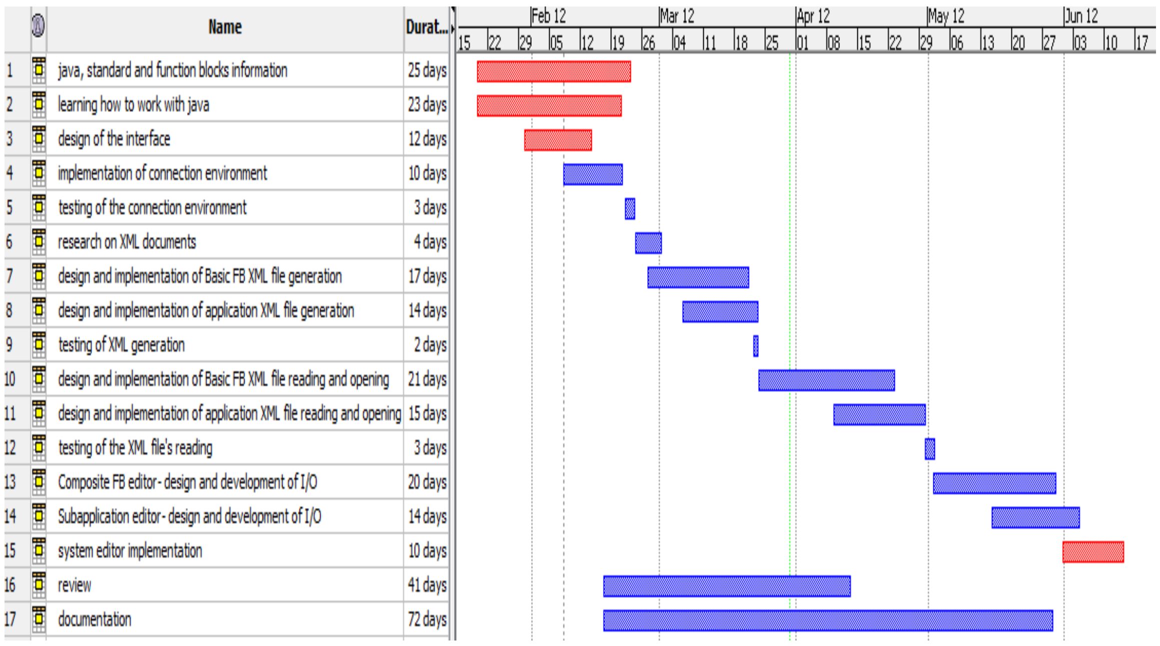

Unit process modelling
Process-feature mapping
Adaptive machining process planning
Reconfigurable assembly planning
Intelligent algorithm embedding
Process Planning
Wise-Factory
Status monitoring
Machine availability checking
Abnormal event detection
Resource Monitoring
Adaptive Decision Making
Execution Control
Feature-function block (FB) mapping
FB library design and development
FB-enabled control
Wireless Shop Floor
Sensor Network
Establishing a wireless sensor network for system testing and validation
Distributed and Dynamic Manufacturing Processes

Figure 2: Closed loop, sensing-monitoring-planning-control (*project definition, 2012*)

1.3 Project planning

In this section how the thesis has been structured and planed will be explained. The project was planned to be able to end in five months: From the middle of January till the middle of June in 2012. As it can be seen in figure 3, which shows the Gant diagram of the project, the thesis was fragmented into shorter different tasks. Moreover, each task has been assigned with its own, more or less, beginning and ending dateline.



	Name	Durat...
1	java, standard and function blocks information	25 days
2	learning how to work with java	23 days
3	design of the interface	12 days
4	implementation of connection environment	10 days
5	testing of the connection environment	3 days
6	research on XML documents	4 days
7	design and implementation of Basic FB XML file generation	17 days
8	design and implementation of application XML file generation	14 days
9	testing of XML generation	2 days
10	design and implementation of Basic FB XML file reading and opening	21 days
11	design and implementation of application XML file reading and opening	15 days
12	testing of the XML file's reading	3 days
13	Composite FB editor- design and development of I/O	20 days
14	Subapplication editor- design and development of I/O	14 days
15	system editor implementation	10 days
16	review	41 days
17	documentation	72 days

Figure 3: Gant diagram of the thesis

On the other hand, most of the working days during the thesis work were structured similarly. Five working days a week divided in two parts: in the morning, all the work was focused on the implementation of the function block environment (development of the functionality); during the afternoon, apart from the implementation, the written documentation of the thesis was carried out.

2. Literature survey

This literature review contains important information about the needed knowledge to manage to understand the thesis work. This section is divided into different parts. On the one hand, standards necessary for utilization in automation will be explained and, it will focus on the latest international standard: IEC 61499. On the other hand, global information about function blocks and their environment will be provided. Moreover, there will be two more sections where generic information about web based environments and XML documents will be provided.

2.1 Why the use of standards in automation

In the early 80's, the revolution of automation changed the way industrial production used to work. Most of the enterprises decided to work with automated machines because of its significant advantages: productivity and efficiency, for instance. Nevertheless, a major problem was generated because each firm owned its automated machines rules, languages, laws, standards... A global standard was necessary for industrial automation in order to solve this really important problem. It would enable connections among different firms making possible the interaction among them. So, that is why the international standard 61131 for Programmable Logic Controllers (PLC) was published in 1993 (called IEC 1131 at the beginning). The international standard was divided into three parts (Siemmens, u.d.):

- 61131-1: General information

- 61131-2: Operation equipment requirements and tests

- 61131-3: Programming languages

One of the most important novelties of this first automation focused standard was the 61131-3 part. Five programming languages were allowed to be used for PLC programming: Ladder Diagram (LD), Sequential Function Charts (SFC), Function Block Diagram (FBD), Structured Text (ST) and Instruction List (IL) (Tisserant, et al., 2007). The first three languages are graphical languages while ST and IL are commands using test languages. LD is a widely used relay based language which has a ladder shape. The figures created are similar to a hard-wired relay sequence. The SFC is a graphical language based on the French language GRAFCET. This language uses a simply execution order: it always executes from the top to bottom; so, usually, the program is quite simple to understand. Function Block Diagram, instead, is a graphic language where functionalities are distributed and encapsulated within blocks. Blocks appear interconnected among each other containing its own outputs and inputs providing reusability and interoperability. A deeper analysis about function blocks will be performer later on. On the other hand, IL is a written programming language based on test using commands; and, finally, ST is a high level language based on block structure, similar to the Pascal code.

According to the mentioned international standard, all this programming languages kept a cyclic scan execution control specified in the international standard 61131. This means that the controller is continuously running the program each cycle; e.g. each millisecond. It facilitates quick data storage apart from internal variables and outputs refresh according to the inputs values.

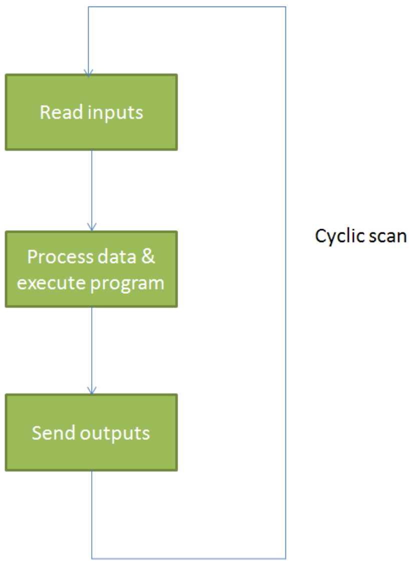


Figure 4: Cyclic scan based execution control

The International Standard IEC 61499, which will be explained in a separated section, was published as an improvement of the IEC 61131 by the Industrial Electrotechnical Commission (IEC). This is a function block oriented standard, where the execution control system is considerably improved: a real-time control application is possible.

2.2 Function Blocks

Within the following section information about function blocks and their environment will be provided. As known, the international standard IEC 61499 is based on function blocks. This means that before entering within the international standard, it would be interesting to understand what function blocks are.

2.2.1 Why are Function Blocks so used in automation?

Nowadays, function blocks utilization is getting more and more usual in real-time control applications. There are many reasons that support this idea. Function Blocks is an object oriented (OO) software model that works as in an electronic circuit when treating the encapsulated behavior (Lewis, 2001). They are used to program applications and its major advantage is that, instead of being only focused on one large and long functionality (as with most of the programming languages happen), it is divided in many small codes. Each function block contains a part of the functionality. This provides to the user adaptability, reusability, portability, independence… Rewriting whole new functionality in order to create some other similar functionality does not occur with this standard.

The new standard of Function Blocks is based on function block diagrams, which is a programming language (according to the IEC 61131, IEC 61499 does not recognize it as a language) that consists of blocks interconnections. These boxes contain information stored as inputs and outputs, apart from the internal variables. Besides, inside each block there is a state machine (Execution Control Chart) and one or more algorithms that process all data (information). In Function Block standard there is no global variables available: all variables are encapsulated in blocks.

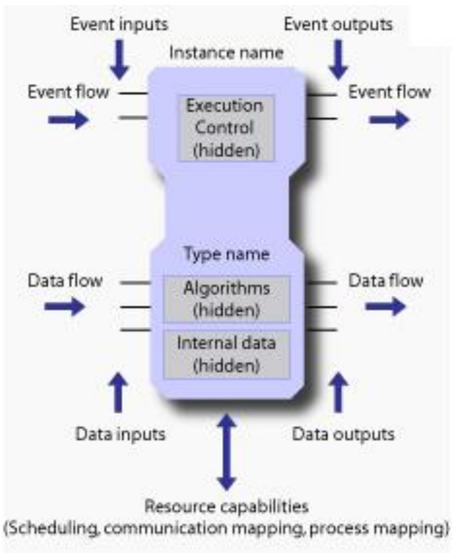


Figure 5: Structure of a basic function block (Isagraf, u.d.)

2.2.2 Types of FBs

There are different types of Function Blocks (International Electrotechnical Commission, 2005): Basic function blocks, composite function blocks and subapplications. The basic function block is the simplest Function Block while the composite function block and the subsystem contain a FB network comprised of more function blocks. It also exists a fourth one, with a different goal: the Service Interface function block.

Basic function block

This is the simplest and easiest structured Function Block type. As it is shown in figure 6, it is composed of different kind of inputs and outputs (data and event), an Execution Control Chart, algorithms and internal variables or data. All the other function block types contain basic function blocks.

Composite function block

This second type of function block has an internal function block network which is composed of basic and also other composite function blocks as image 6 shows. All Function Blocks within the network are interconnected among them. There is no internal variable required in Composite Function Blocks because sampling and storage is defined for all possible sources of data (Lewis, 2001).

Subapplication

The structure and behavior of a subapplication is quite similar to the Composite Function Block´s one. This type of function block has also an internal network composed of basic and composite function blocks. Nevertheless, apart from the basic and composite function blocks other subsystems can also be part of the internal function block network. This is the difference between Composite FB and subsystems.

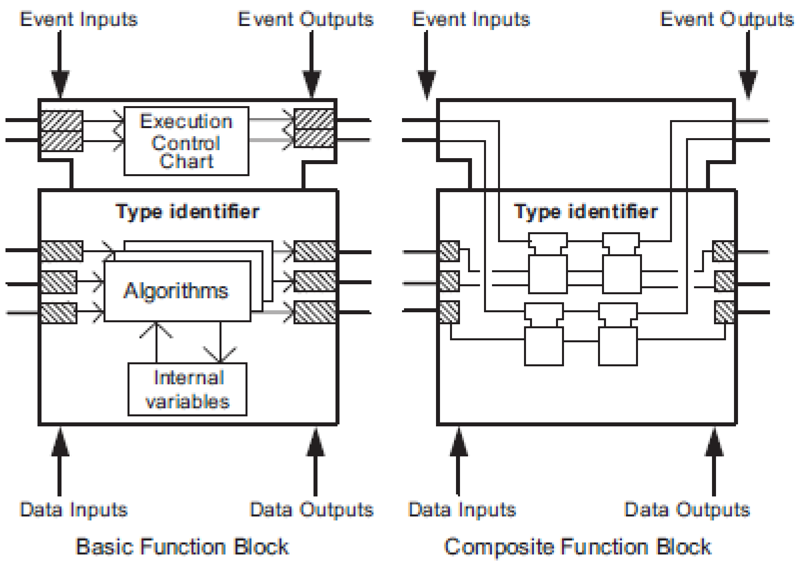


Figure 6: Basic and composite function blocks (Keshavarzmanesh, et al., 2010)

Difference between Composite FB and Subapplication

Composite Function Blocks could have internal variables. It has guarantees for the sampling of I/O data when they are associated with the events. That is why it is not distributable, internal connections cannot be broken running them on different processing resources. However, "*to achieve the same effect this can be done by converting the composite block type into a subapplication type*" (Lewis, 2001).

Service Interface Function Block

Apart from the three already explained types of function blocks, there is another different kind of function block which is not used in the same way as the others; it is called service interface function block.

Its goal is to provide an interface between some other function blocks that are running in the resource and some services outside the resource. It defines the interface into the service and the answer that it provides (Lewis, 2001). Normally, this type of function blocks functionality use to be:

- Read and write inputs and outputs from the device's I/O subsystem.
- Request or respond data to external resources.
- Manage the execution and creation of function blocks in the resource.

In addition to the functionality of this function block, it has to be mentioned that it always keeps a determined standard set of inputs and outputs.

2.2.3 External behavior of basic function blocks

Referring to the external behavior of basic function blocks, input/outputs and connections among I/O are found. There are some different types of I/O and connections that are explained in the following.

Kinds of inputs and outputs

All types of function blocks have two kinds of inputs and outputs: data and event. As can be seen in figure 7, event inputs and outputs are situated on the top side of the function block. On the contrary, data information always appears at the bottom side of the box. Moreover, the figure that represents function blocks have a narrower part between the two types of information flow in order to be able to differentiate between them (Christensen, u.d.).

Data input/output

Data information comes from an external source into the function block and it is also delivered from the function block. These external sources can be some other function block´s inputs or outputs, constants, application´s input or output… Data input is in charge of activating the event inputs. Data information is essential for function blocks to work. If data input value does not raise or fall, event inputs and, hence, events will not be activated; as a result, algorithms will not be executed.

Event input/output

Event inputs are always activated when a data input Boolean variable of the same function block is changed (falling or raising the value) or when an event output of another different function block commands it. The corresponding event is enabled, and when event containing algorithm execution has ended, event outputs are triggered. This execution order will be studied in depth in the execution model section.

Kinds of connections

On the other hand, there are also two kinds of connections: first, connection between event and data in the same function block; and connections in order to join two different function blocks, between data outputs and data inputs or event outputs and events inputs (International Electrotechnical Commission, 2005).

Event-data associations in a Function Block

The connection between event and data information in a function block is illustrated vertically in figure 7. These connections associate data values with event data of the same function block, but always between inputs or between outputs. If an event-data association is performed in a function block entrance, the same association should be performed in the output of the previous function block.

These associations aim the activation of an event when its corresponding data entrance requires it. When a data input changes its value, the connected event input will be activated ordering an event execution. The main objective of these connections is to work with a real time control and, in addition, to only execute the necessary events or tasks with valid data and only when it is needed. This will be better explained in the section that the international standard IEC 61499 is explained.

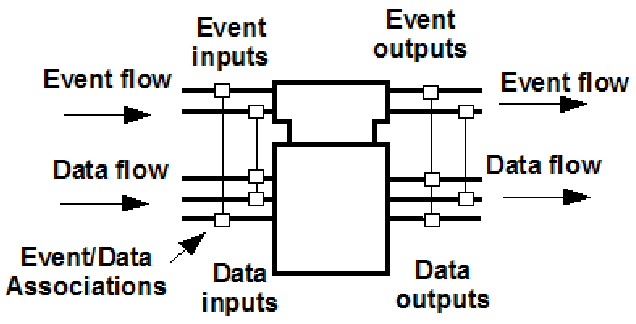


Figure 7: Event-data association (Christensen, u.d.)

Connections between different Function Blocks

As far as connection among function blocks is concerned, its aim is interconnecting function blocks in order to create function block networks. These function block networks are called systems (or applications) and even subapplications (a previously explained smaller system, within another general function block system). A function block system represents the global function block network with all the connections among all function blocks. In short, all the created function blocks in the program must appear within the system. Apart from subapplications and systems, connection between function blocks also appear inside composite function blocks to join its internal function blocks.

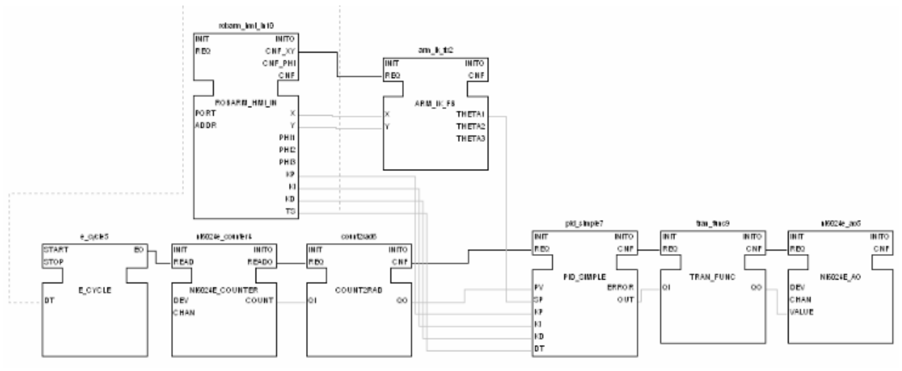


Figure 8: Function Block Network example (Google, u.d.)

Obviously, this type of connection is performed between data connectors or between event connectors, but never mixing them. A good example of a function block system and its communicating connections is illustrated in figure 8. Within the eight function blocks seen in the figure, there might be more function block networks. That depends on the type of function block they are.

2.2.4 Internal behavior of a basic function block

When describing the internal behavior of a basic function block, two important aspects have to be differentiated: the Execution Control Chart and the internal algorithms (Lewis, 2001). Internal variables are also part of the internal behavior of basic function blocks.

Execution Control Chart

Function blocks are controlled internally by a state machine, which is represented by an Execution Control Chart (Wang, et al., 2009). The Execution Control Chart is the function that relates events to the execution of the function block´s internal algorithms; It is dependent on some internal states. This special state transition notation expresses the mapping of the events on to the algorithms within the function block. It is defined in the international standard IEC 61499 as a precise and formal way that shows how events are able to trigger the execution of internal algorithms. This chart is normally represented graphically; however, it can also be shown textually. The SFC programming language representation, defined in the international standard IEC 61131, is very similar to the way of representing an ECC. Figure 9 shows a simple example of the common graphical representation of an ECC.

Three states can be appreciated in the figure: START, the initial state, when one of the event inputs (E_Run or E_Init) is enabled, this state will switch automatically to the next one; INIT, the following state, activates an event (ALG_INIT) triggering an event output (E_Rdy); finally, RAMP, the third state, activates another event (ALG_RAMP) triggering its corresponding event output (E_Ex0).

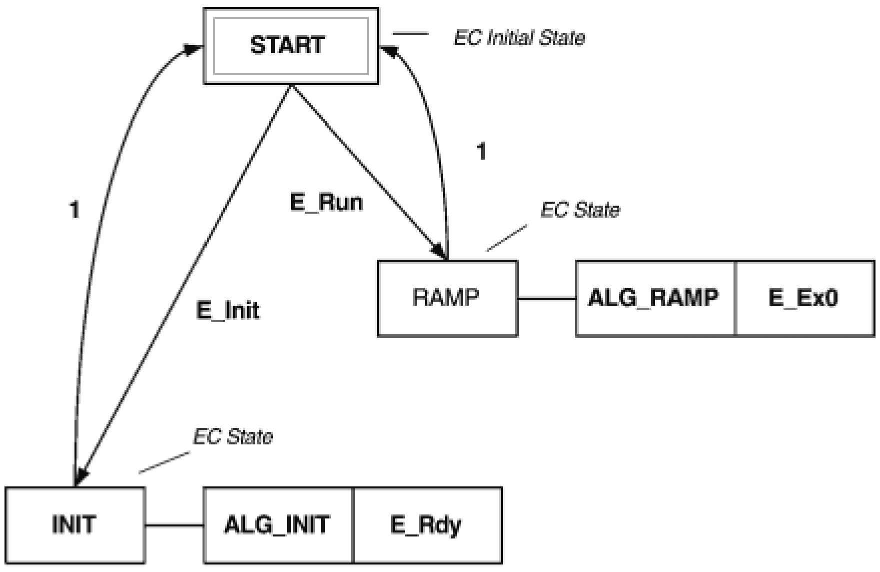


Figure 9: Execution Control Chart´s graphical representation (Lewis, 2001)

The ECC of a basic function block shows: the main internal states; how the function block responds to each type of event input; which algorithm is activated as an answer to event inputs; and, which event output is triggered when algorithm´s execution has finished. Hence, execution control states, transitions and actions are fundamental for the representation of a function block state machine (EEC).

Algorithms

The basic function block almost always contains one or more internal algorithm. These algorithms are invoked by the resource scheduling function answering to an external data input. Algorithm execution means that inputs and internal variables are processed producing new output data and changing internal values. When the execution of algorithms ends, an event output is usually triggered notifying all the other components that the execution has finished. With this notifying signal, some other external function blocks are alerted when the needed output data is ready to be used by them. In the next scheme (“Figure 10”), how an algorithm execution occurs can be visualized with its transitions and states.

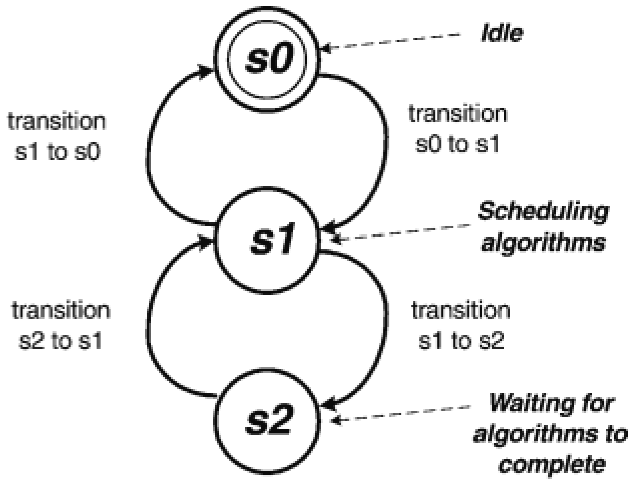


Figure 10: Algorithms´ execution chart (Lewis, 2001)

Figure 10 shows three states (s0, s1, s2). When an event input is activated a transition will switch the control to another state. Then, if an algorithm must be executed, it will be checked; if not, it will return to the previous state.

Regarding the algorithm´s defining language, it has to be added that the IEC 61499 does not specify any determined language. Nevertheless, “JAVA” and “Structured Text” are the most widely used programming languages in order to define algorithms. But some other languages can also be used to define algorithms (Gerber, et al., 2008), such as ladder diagram, instruction list, C language, C++…

2.3 International Standard IEC 61499

The international standard IEC 61499 is a standard related to automation published in 2005. Indeed, another edition of the standard was published in the year 2000, called IEC/PAS 61499-1, but many technical changes have been effected in the IEC 61499; So, it cancels and replaces completely the IEC/PAS 61499 (International Electrotechnical Commission, 2005).

It has to be said that it is not the first standard published that is related to automation as it has been seen before; in fact, it could be said that this standard is the continuation of the IEC 61131. The IEC 61499 standard proposes an open architecture for distributed Industrial Process Measurement and Control Systems (IPMCS). It defines how function blocks must be used in distributed industrial processes, measurement and controlling systems (M., et al., 2008). This international standard creation has been a very significant step for the embedded systems sector due to the necessary real time control.

Three main features stand out in the standard: portability, interoperability, and configurability (Christensen, u.d.). Portability is accomplished because, software components and system configurations produced by any software tool, are read and accepted by any other software tool; interoperability is reached because embedded devices have the possibility of operating together to perform needed functions for distributed applications; and, finally, configurability, means that configuration is free , so all the software devices and components have the ability of being configured dynamically by software tools from any vendor. The international standard is divided into 4 different parts (International Electrotechnical Commission, 2005); however, this literature review will only be focused on the first and second parts.

- IEC 61499-1: Architecture
- IEC 61499-2: Software tool requirements
- IEC 61499-3: Application rules
- IEC 61499-4: Rules for compliance profiles

As explained above, function blocks are the basic functional software unit of this standard. It encapsulates its own data structure and a set of internal algorithms (Martinez Lastra, et al., 2005). The IEC 61499 helps reaching the requirements asked in nowadays society: manufacturing world using decentralization and a hard real time design philosophy.

2.3.1Function Block´s reference models

One section of the international standard is focused on how to structure a function block program. There is a model hierarchy set up in order to accomplish a structured and ordered program (International Electrotechnical Commission, 2005). This hierarchy scheme is found in figure 11. The system model is the major one as can be seen in figure 11, and the rest of the models, explained in the following lines, are smaller models within it.

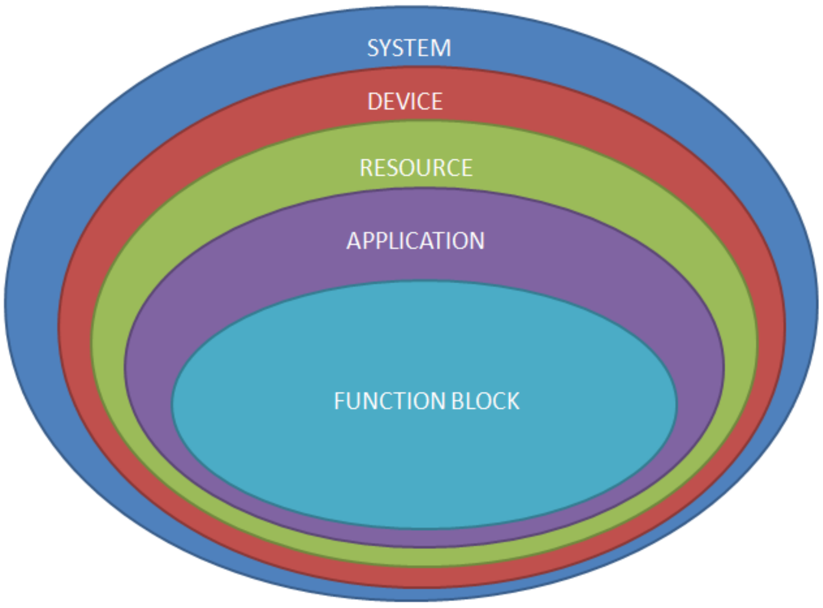


Figure 11: Model's hierarchy

System model

An Industrial Process Measurement and Control System (IPMCS) model is used in order to respond to this specification. This model is in charge of defining the relation between different applications. The system model is comprised of a communication network, a controlled process and machines and devices. These devices are interconnected by means of the communication network which is composed of links and segments (International Electrotechnical Commission, 2005).

As can be observed in the figure 12, applications can be distributed in more than one device if the user desires. Operating distributed means that each device loop performs a different task: input sampling, control processing… However, each application can also be performed within only one device.

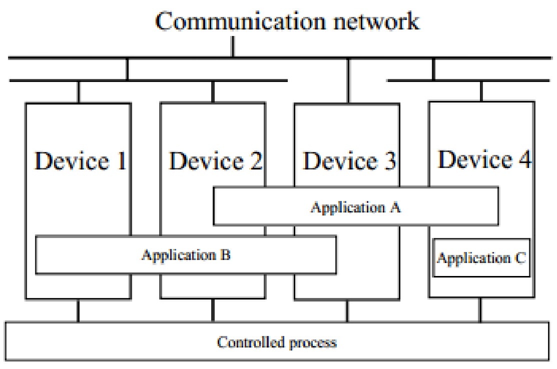


Figure 12: System model (Keshavarzmanesh, et al., 2010)

Device model

According to the device model, it has to be said that it is able to support more than one resource. Nevertheless, it can also contain no resources (Lewis, 2001); it depends on the user. Its composition always supports, at least, one interface. However, one device can also hold up to two interfaces: communication interface and process interface. On the one hand, the process interface is responsible for the performance of the mapping between physical variables, analog input/outputs, discrete inputs/outputs, measurements…, and resources. On the other hand, the communication interface is in charge of the mapping between resources and the information which flows through the communication network (data, events, configuration information…) (International Electrotechnical Commission, 2005). Figure 13 shows an easy and clear device structure.

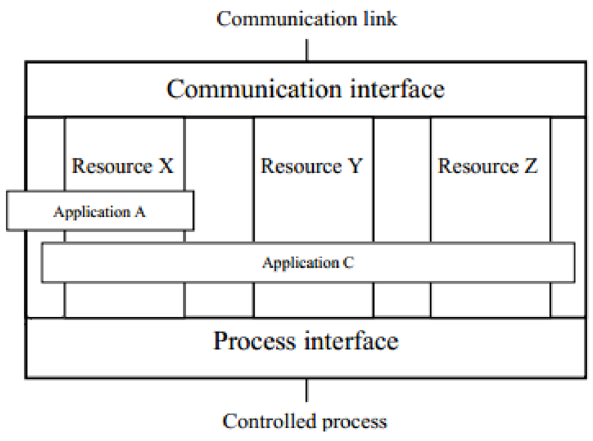


Figure 13: Device model (Keshavarzmanesh, et al., 2010)

Resource model

Resources, functional units that belong to devices, supply an independent execution and control of the applications (Lewis, 2001). A resource is totally independent, as mentioned, and whatever occurs in a resource (configuring, creating, deleting…) does not have an impact on any other resource; i.e. resource execution does not affect other components within the system. The resource model is comprised of process and communication mapping, a scheduling function and one or more applications. Indeed, as shown in Figure 13, it can be observed that resource X contains A and C applications while resource Y and Z only contain one application, C application.

While a resource is working, the mapping function occurs. This means, the resource receives data and event information from the interfaces that the devices contain (communication interface and process interface). After having processed them, resources return the obtained processed information to the same interface. The mentioned scheduling function, apart from transferring data, performs the function blocks execution in the local application.

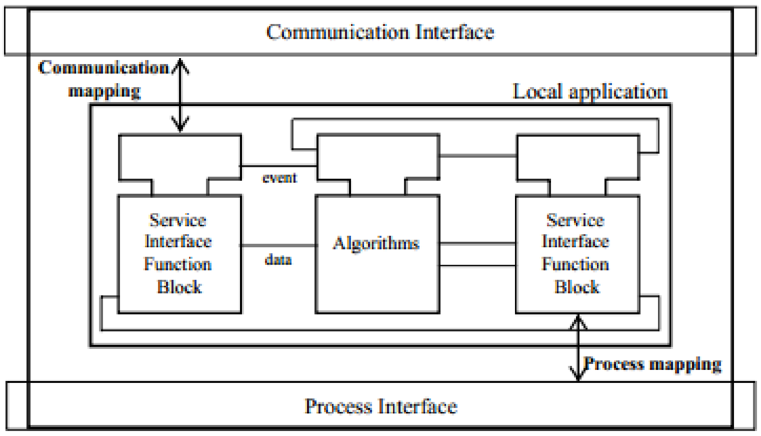


Figure 14: Resource model (Keshavarzmanesh, et al., 2010)

Application model

The application model is where the function block network can be found. In other words, in this section different kinds of function blocks (basic FB, composite FB or applications) appear interconnected by means of event and data connections. The IEC 61499 base is focused on this model: the model referred to Function Blocks Networks. In fact, it can also be called a "function block model". It is not possible for an application to interface with other applications (Lewis, 2001). An application is usually fragmented into many resources, as it has been explained in regards to the device model. In addition to this distribution, a further decomposition is generally performed by means of subapplications. In Figure 15, it is illustrated what simple application model would look like.

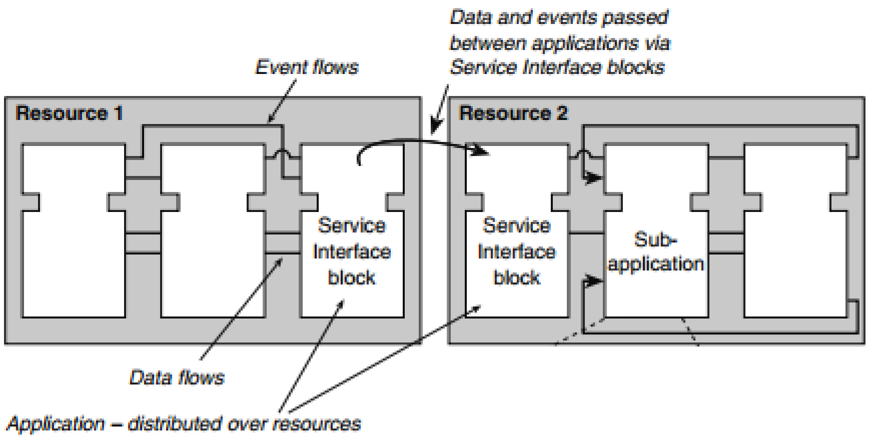

Data and events passed between applications via Service Interface blocks
Event flows
Resource 1
Resource 2
Service Interface block
Service Interface block
Sub-application
Data flows
Application – distributed over resources

Figure 15: Application model (Lewis, 2001)

Function Blocks model

In this model, the normative applied to basic function blocks (point 2.2) is explained: measures, execution orders the size of interface figures, visualization forms, the shape of figures… This is the most important model when referring to function blocks. The base of the programming is located within function blocks, where algorithms, internal data, the execution control and resource capabilities come into action receiving, processing and providing data.

In fact, in the following section how the execution process is performed is explained step by step. In order to reach this sequence, event and data inputs and outputs are necessary as it has been explained in their corresponding sections.

Basic Function Block´s execution model

Basic function blocks, which are the elemental components of all function blocks, always have a determined execution model. The execution always happens in the same way with a specified order as illustrated in Figure 16. Moreover, depending on the priority of the events, the execution order could be different. On the following paragraph, a written explanation of the function block´s execution model is provided.

First of all, data input variable value (step 1) orders the activation of the corresponding event input (step 2), so that the event occurs. The execution control notifies the resource scheduling function to run the event´s corresponding algorithm (step 3). The resource scheduling function is in charge of ordaining commands and priorities to events and tasks. After the algorithm´s execution is completed (step 4), a new data output value is returned (step 5) which means that any other external function block can operate with it. In consequence, the resource scheduling function is notified that the algorithm´s execution has ended (step 6). Finally, as a result of the notification, it sends the information back to the execution control (step 7) so that an event output will be triggered (step 8) (Martinez Lastra, et al., 2005).

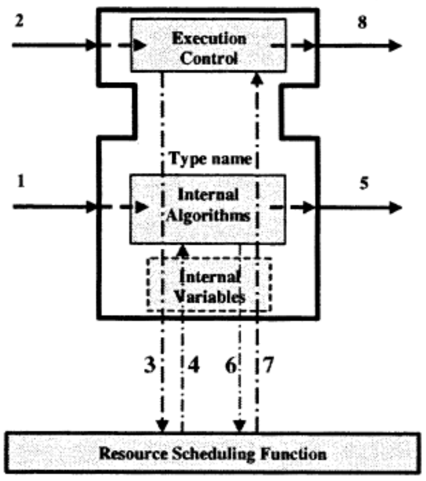


Figure 16: Basic function block´s execution model (Martinez Lastra, et al., 2005)

Distribution model

The distribution model deals with the requirements needed when applications and subapplications are distributed among different devices and resources. Figure 17 shows clearly how applications and subapplications can belong to more than one device and/or resource. A deeper analysis of this model can be found in the international standard IEC 61499.

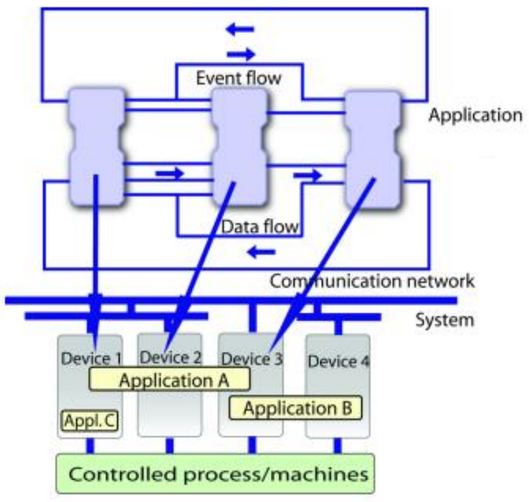


Figure 17: Distribution model (Isagraf, u.d.)

Management model

The management model consists of managing resources and devices, as it can be deduced from its name. It can be divided into two groups of models: the shared management model and the distributed management model. These types of applications can be modeled using service interface function blocks and communication function blocks.

2.3.2 Event driven execution control

This is one of the most significant innovations of the international standard IEC 61499, apart from the reference models. It consists of a real time execution control application which provides an important flexibility and descriptive power (Lewis, 2001). As it has been explained before in 2.1, execution control in the IEC 61131 consisted of a cyclic scan based execution model; this means that the controller was continuously reading the program from the top to the bottom. For instance, every microsecond the whole program should be read looking for input value changes. However, the important default of this system was that it executes the whole program serially. This would mean that if a long task had to be executed, the program would not be able to run the following events until the long task had ended.

In other words, this controlling system was not so effective for real time applications. Besides, it is not a suitable control for event driven distributed systems. Solving this problem has been one of the goals of automation research during the last decade: trying to implement a real time control application. The result has been the event driven execution control. This is the model that provides a real time control, an essential control in many machines and devices nowadays. Changes are achieved instantaneously. Furthermore, the event driven execution control reduces the computing power considerably (Gerber, et al., 2008) and offers to a numerical control (NC) machine intelligence and autonomy (Wang, et al., 2009).

This new execution controlling model published in the international standard IEC 61499 is designed to work by means of an event communication. Tasks are divided into different events within the function block. So, depending on the value of a Boolean data input (could be when rising or falling), one event or another would be executed. That is why each event has its own input and output connected to a data input and output (Keshavarzmanesh, et al., 2010).

Additionally, this new execution model also provides the chance of giving more importance to some events than to others (Martinez Lastra, et al., 2005). This is not an innovative application but, as discussed previously, it maintains efficient functions of the cyclic scan control. Priorities are instructions given in order to execute some events before others, depending on the event´s importance; particularly, when two or more executing commands are ordered at the same time.

2.4 Web based environments

In the next section, web based environments will not be analyzed very thoroughly, but most of the important characteristics will be shown. Regarding the project, creating an IEC 61499 standard compliant function block environment is not so innovative. Indeed, there are some existing programs able to translate function block languages into java, such as, FBDK (Function Block Development Kit) or 4DIAC (Framework for Distributed Industrial Automation and Control). However, the really innovative part of the thesis is creating this program in a web based environment, reaching a real time data process (Wang, et al., 2009). Developing programs in web based applications has a wide range of advantages (Campbell, 2007):

- Cross-platform compatibility: It works with any operating system (UNIX, Linux, Mac and Windows).

- Updates: All users benefit from the updates immediately.

- Immediate availability: Installations and configurations must not be performed.

- Real time data availability: New input information is available immediately for other users.

- Data availability across locations: Data can be used and seen immediately from anywhere all over the world.

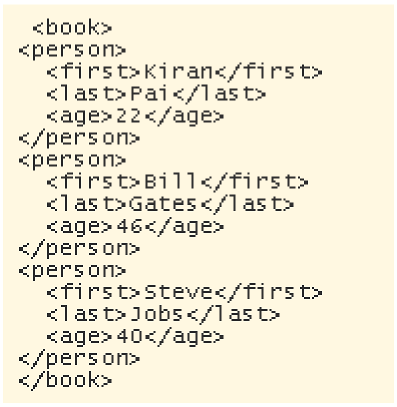
-Data is safer: Hardware sometimes fails. Server uses redundant storage and regularly scheduled backup; so, a hardware error does not mean that data has been lost.

2.5 XML documents

Extensible Markup Language (XML) file is derived as a subset of the generic ISO Standard Generalized Markup Language (SGML). The development of XML has generated a new stage that will replace HTML (Hypertext Markup Language). HTML is the current used format document content on the Internet World Wide Web (Lewis, 2001).

XML is a human and machine readable language. This format is becoming more and more usual because of its simply structure, easy understanding and variety of advantages. XML is a vendor with its own independent format with a very similar structure of HTML. The most important feature of this kind of files is that it allows the exchange of files between different tools and devices of different vendors. Computer programs use to be converted into XML files when they are saved so they are able to port it in a format which any operative system and program accept. One advantage of XML documents comparing with HTML files is the DTD creation method. XML documents are marked up by tags to have an ordered tree structure. The set of tags are defined by a Document Type Definition (DTD). In HTML files a single document type, which defines all the HTML tags, is created. The problem is that HTML cannot provide enough hosts of tags and functionalities needed in web content. XML language, instead, has a standard mechanism to define new tags for any document builder. So, Extensible Markup Language can also be used to define new DTDs (Lewis, 2001). Another advantage of the XML is that it is extremely flexible in diverse information storage. XML is believed to be the new effective way for Internet transferring information. Additionally, it will probably replace the HTML language from Web pages in few years time.

XML documents must have start, end or empty-element tags as we can see in Figure 18. In addition, within tags use to appear child elements (more tags inside global tags), and this child elements use to have attributes in order to define which child or element they belong to.



```
 <book>
<person>
  <first>Kiran</first>
  <last>Pai</last>
  <age>22</age>
</person>
<person>
  <first>Bill</first>
  <last>Gates</last>
  <age>46</age>
</person>
<person>
  <first>Steve</first>
  <last>Jobs</last>
  <age>40</age>
</person>
</book>
```

Figure 18: XML file shape (Pai, 2002)

As can be observed in the figure, within "book" element, three different children ("person") are defined. These children's information is defined by means of the attributes: "first", "last" and "age". The international standard IEC 61499 also defines a determined normative for XML files that are created from function blocks' programs (International Electrotechnical Commission, 2005): what information should appear in the file, elements' names, order, structure…

3. Methodology

During methodology section, diverse ways to reach the objective of the thesis will be analyzed, checked and compared. On the next pages, how these challenges could be accomplished will be explained. In implementation (point 4), instead, one determined way will be chosen and specifically explained; in few words, what has been performed will be explained. The main challenges of this project are implementing a function block serialization for saving and editing XML files; developing an integrated user interface (focused on implementing connections among function blocks according to the international standard IEC 61499 and adding some more little enhances to the graphical user interface: buttons, menus…); and preparing library of exemplary basic function blocks.

3.1 Research on Function Blocks

First of all, performing deep research in function blocks is the base of the project. It is the first step to effect during the project because, if a high quality thesis is desired, having a strong base is essential. Research can be performed looking for information in several safe sources like articles, journals, internet, books, encyclopedias… Afterwards, all that information must be mixed and understood. Carrying out deep research on the topic always helps to solve a variety of problems and errors that appear during the implementation stage. It must be added that research on Function Blocks and its run-time environment has taken a long time of the thesis´ initial months.

3.2 Programming languages

The current project is the extension of an existing thesis, the main part of which had been developed before release of IEC 61499 International Standard. In order to continue with the implementation of the thesis, a programming language is needed. Nowadays, there is a great deal of programming languages in the market. Nevertheless, only some of them have a significant usability all over the world. “C” language, “C++” (an extension of C language), “Java” (this project´s previous stage was implemented in java) and “PHP” are the most used programming languages as shown in Figure 19. So, it would be an intelligent decision selecting one of these four languages. From the beginning, “Java” language was chosen for the programming as it will be shown during the implementation part. There are many compelling reasons to support this decision.

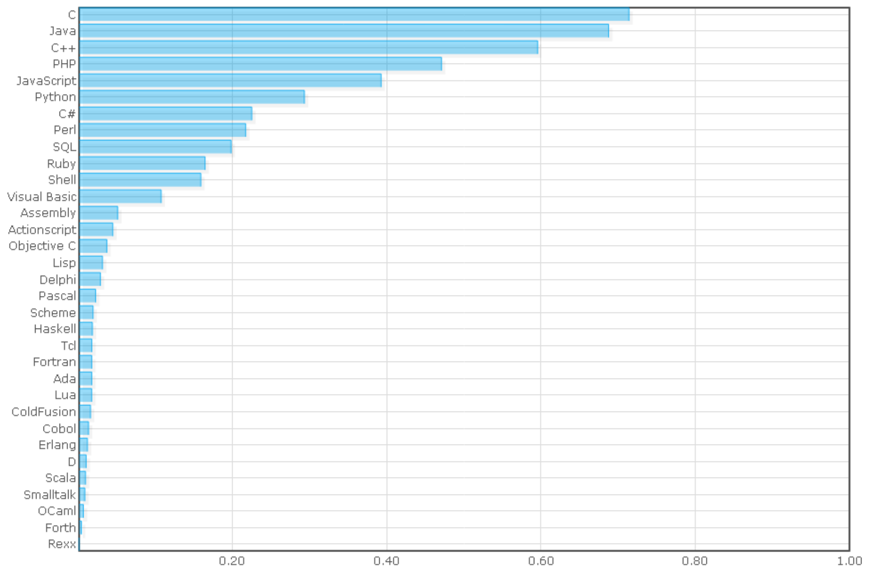


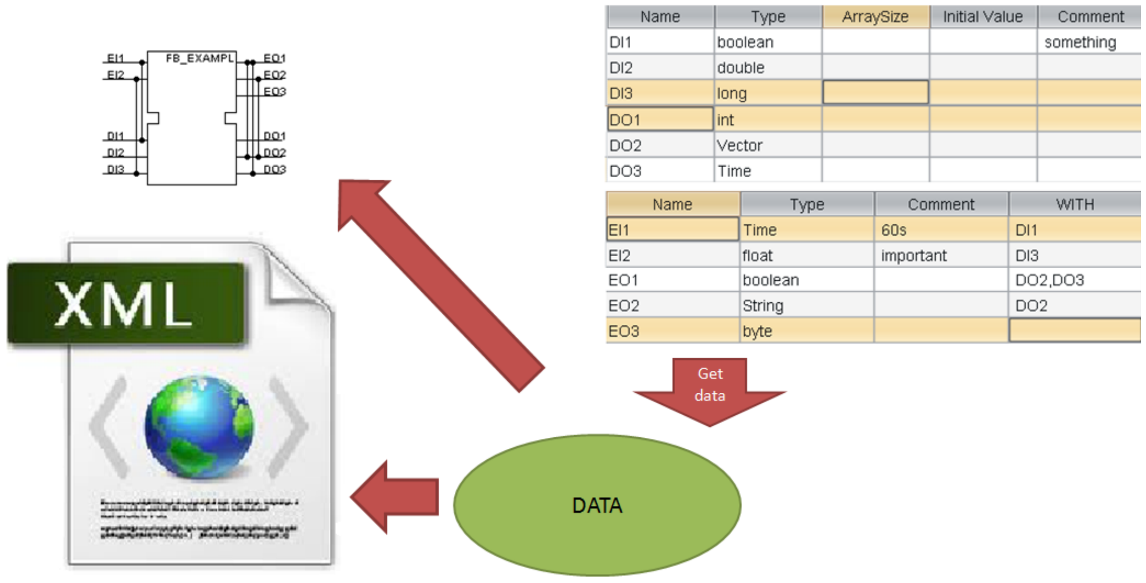
Figure 19: Programming languages popularity (Langpop, 2011)

3.3 Saving and opening XML files

The following subsection will be focused on one of the objectives of this thesis: generating and reading XML documents. These two actions should be achieved with basic function blocks, composite function blocks, subapplications and systems. XML files are the documents that must show data from the created program in a specific textual way. The structure of the XML file must be different when referring to basic function blocks or the rest (applications). Applications, that is, composite FB, subapplications and systems have a similar structure as explained in the literature survey. They are based on a function block network. So, information about connections, connectors, source and destination function blocks and position coordinates of all figures within the FB network must be provided. Moreover, information about each function block within the network must be provided. On the other hand, within XML files of basic function blocks, data input/outputs and event input/outputs information must be provided. Associations between data and event connections should be shown too. Furthermore, all data of the ECC should also be a part of basic FB's XML documents.

3.3.1 Generate XML file when saving

In the next figure (Figure 20), it can be graphically observed what the next lines will consist of: the XML file generation. In other words, data will be retrieved from data storages and shown in another different way: in an XML document.



Name	Type	ArraySize	Initial Value	Comment
DI1	boolean			something
DI2	double			
DI3	long			
DO1	int			
DO2	Vector			
DO3	Time			

Name	Type	Comment	WITH
EI1	Time	60s	DI1
EI2	float	important	DI3
EO1	boolean		DO2,DO3
EO2	String		DO2
EO3	byte		

Figure 20: XML generation scheme

The most common ways to generate a XML document from a function block graphic are DOM parser and SAX parser methods (Yandell, 2002). Both are specified in API (Application Programming Interface), a source-code based specification intended to be used as an interface by software components for communication (Anon., 2012). However, these two are not the only method to decipher a XML file: StAX, StringBuffer class, XMLWritter class, XSLT… could also be used. Parsers are fundamental components because they provide a bridge between XML documents and the application that processes that XML file (Fyicenter, u.d.). DOM and SAX parsers analyze the document´s contain, recognize data and get the required information that has been asked in the functionality.

On the one hand, Dom Parser (Document Object Model) is a simple java programming way focused on working with an object. The DOM method is a tree structure document which fits very well the XML file's structure (Ziesemer, 2007). The document can be created in segments, reordered, edited whenever it pleases the user before being serialized. Some other classes and a transformer must to be added to the document in order to accomplish a valid xml file. The object, which is created by default, is in charge of writing (“”), (< >) or tabulators in its corresponding moment. The user does not have to take care of those symbols. This is one of the reasons that DOM is said to be the easiest and cleanest method to generate an XML file (Yandell, 2002). Nevertheless, its only problem is related with memory space. It is not recommended to use this method when generating large xml files because a storage problem can occur.

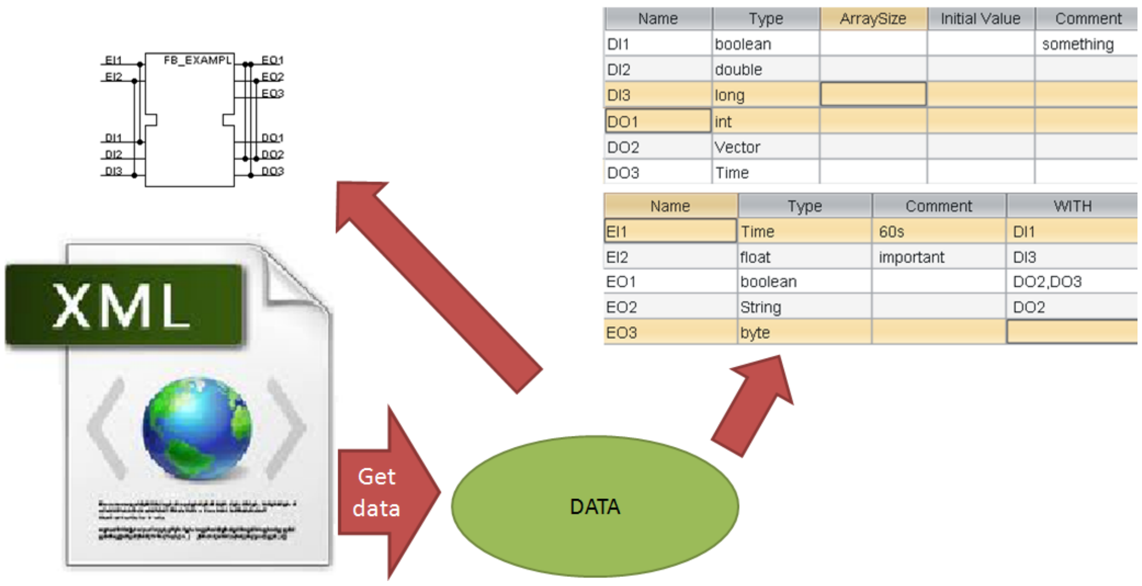
On the other hand, SAX Parser (Simple API for Xml), which is a less known method, is another way to generate xml files. SAX works generating events and it can only read XML documents (Vogel, 2008). It is true, that sometimes this model does not create valid XML files due to some disadvantages, such as forgetting elements or ending tags. However, it has no problem, as DOM method does, with the memory space (Yandell, 2002). Moreover, having a XML output system around SAX events has a wide range of advantages, but is not a fast way to output results. In the following table the most significant features of both parsers are compared.

DOM	SAX
Tree of nodes	Sequence of events
Occupies more memory	Does not use much memory
Slower at runtime	Faster at runtime
Stored as objects	Objects not created
Easy programming	Need to write code to create objects
Ease of navigation	Backward navigation impossible

Figure 21: Comparison between DOM and SAX parsers

3.3.2 Read and open XML files

Another challenge of the project, as mentioned earlier, consists of implementing an application for reading information from XML files. The obtained data should be saved and displayed in two different forms (see Figure 22): represented in a table and graphically showing function blocks and connections by means of figures and icons. In other words, this function should perform the way back to point 3.3.1.



Name	Type	ArraySize	Initial Value	Comment
DI1	boolean			something
DI2	double			
DI3	long			
DO1	int			
DO2	Vector			
DO3	Time			

Name	Type	Comment	WITH
EI1	Time	60s	DI1
EI2	float	important	DI3
EO1	boolean		DO2,DO3
EO2	String		DO2
EO3	byte		

Figure 22: XML file opening scheme

This application will ease the edition of previously created Function Block programs. API´s DOM and SAX parsers are also the most famous and usual ways to reach this application. When reading from XML files each parser has its own execution pattern. In DOM parser case, before reading the java functionality, the entire document is read by the parser. When the document reading is ended, the file is loaded into the memory in the application creating an object model (Rosen, 2002). After the parser has finished reading and loading, begins the real code deciphering. On the other hand, SAX parser model consists of a document instantaneous step by step reading. Instead of reading it twice as DOM parser model does, it performs the reading of the document in detail.

3.4 Connections representation

It is true that connections were already implemented in the previous thesis. However, these connections did not fit requirements within the international standard IEC 61499.This project aims to implement an application for IEC 61499 compliant systems development. So, connections´ shape has been enhanced. In implementation section is explained the followed steps. More details are found in appendices.

3.5 Composite Function Block Editor

Application´s editor implementation has been an important part of the project. Not many functions about it were implemented prior to this thesis and considerable improvements had to be developed. As explained in the literature review, FB applications contain FB Networks. So as seen in Figure 23, apart from having input and output connectors within the FB network, an application should also have external I/O connectors. When application word is used in this section, it includes all types of applications except systems, which do not need external I/O connectors. The next implementation has been focused on external I/Os creation in order to edit, connect and delete them.

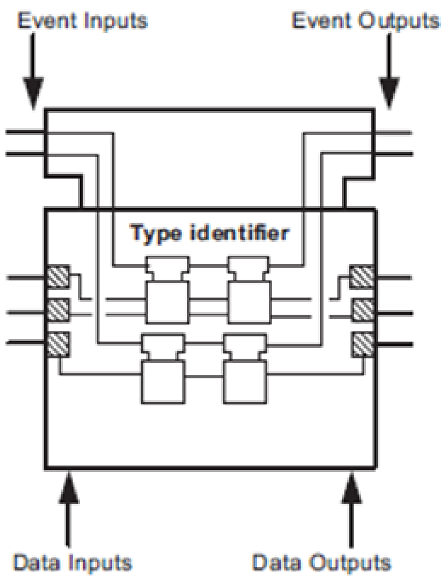


Figure 23: Application figure (Keshavarzmanesh, et al., 2010)

Obviously, there is quite a great deal of ways to reach this aim. A great research of different possible options has been performed; then most possibilities have been discarded in order to reach the best solution. After these steps, only two of the solutions looked efficient, so, both options have been deeply analyzed.

The first option consisted of creating a big box, as shown in Figure 24, which would represent the FB Application. This box should always stay in the background of the screen while the user designs the Composite function block. Of course, the FB Network ought to be designed within the big box. External I/Os would be at the left and right side outside the box. A significant disadvantage of this first option is that the number of I/O connectors of the FB Application determines the length of the Composite FB box. This means that a long number of inputs or outputs would make the box larger and another application should be implemented. Besides, using this method would reduce drawing space in the panel.

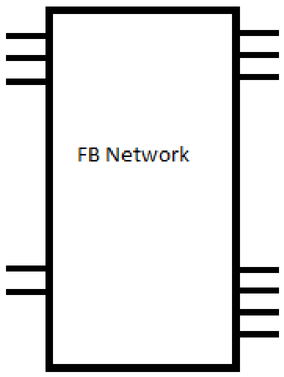


Figure 24: First option of the application design

On the other hand, the second option is focused on using small figures in order to represent external I/O connectors. Each input or output would be represented by means of an arrow shape as shown in Figure 25. These interface figures could be inserted anywhere in the panel. This means that they do not have to be represented as a vertical row. Nevertheless, normally they would be located at the left and right side of the function block network. This solves the problem that occurred within the other main option just discussed. This last reason has been determinant when deciding the final Composite FB editor’s shape. Later on, in implementation part, how the Composite FB editor’s implementation has been developed will be explained in detail.

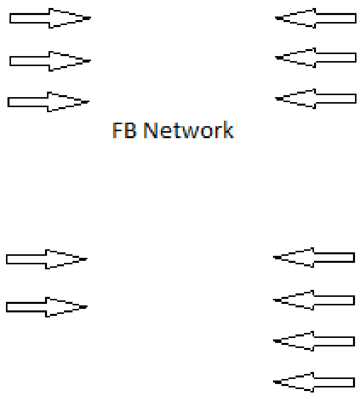


Figure 25: Second option of the application design

In addition to the Composite FB editor design, an application to connect the external I/Os with internal I/Os (I/Os within the FB Network) has been developed. These connections join inputs with inputs and outputs with outputs. The chosen methodology has been inserting a connector point within each I/O interface figure so that they can be threatened as connectors within the FB Network. It will be specified in the implementation section.

3.6 Enhances within the configuration of the Interface

In the following section, minor improvements applied to the web based environment will be presented. Indeed, many used functions were previously implemented for other purposes. So, taking advantage of them, some other tasks have been improved.

At first, function blocks were graphically identified by means of the type. Besides, when representing them, the type of function block was shown within the box. However, the problem appeared when more than one function block from the same type were within a FB network. Function Blocks with the same type contain the same number and name of I/Os and that was quite confusing. Hence, in this project separating “name” and “type” has been decided. Two functions have been differentiated in order to get and set the type and, even the name. A third function called “getID” is responsible of numbering each function block while maintaining the order that blocks are generated. Then, this ID is attached to the type of the generated function block creating the name. The result is a representation of a FB Network where each function block has its own name (type + number of FB (ID)) and type. The name of the function block can be changed when double clicking on the function block. Its name is represented at the top of the box. The type of the function block is found just below the name. The difference between the old and new version can be observed in Figure 26.

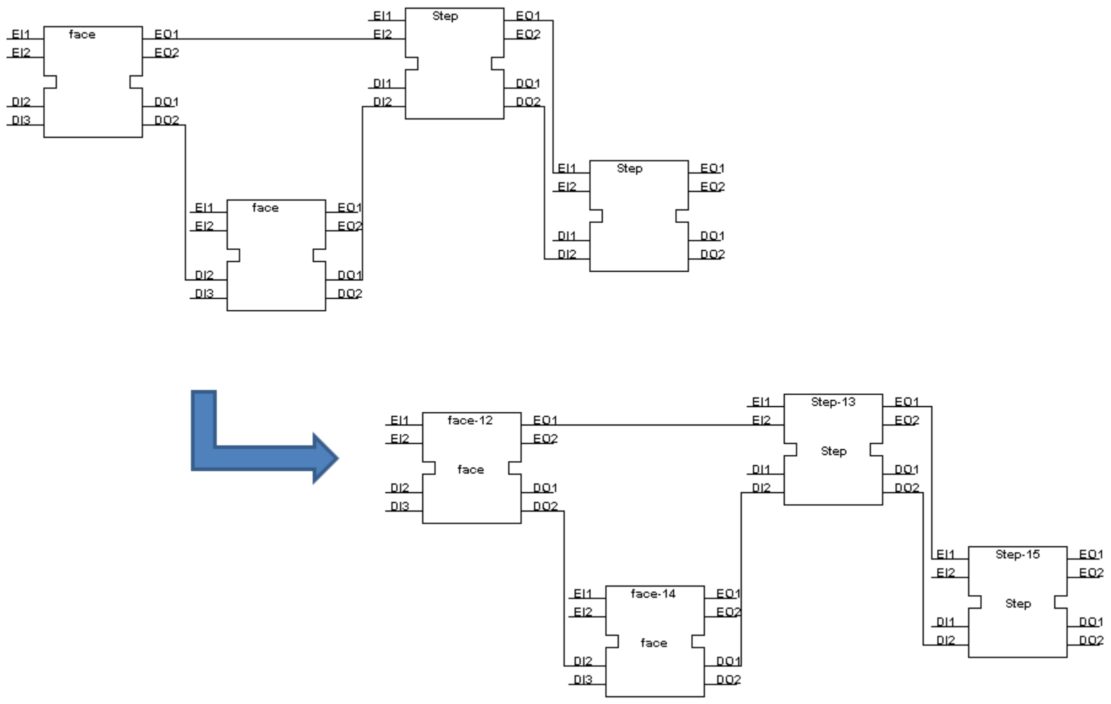


Figure 26: "Name" and "Type" differentiation

On the other hand, previously implemented “drag and drop” application has been exploited. It was just applied in order to move function block´s boxes within the screen. Nevertheless, the possibility of connections to be dragged was not developed. Connection´s drawing uses the same function to interconnect all function blocks and connections usually get overlapped. This “drag and drop” application has been reused for connections dragging. Consequently, connections overlapping problem has disappeared; furthermore, a much better FB Network visualization has been achieved. Figure 27 presents connections at both stages: before and after the problem was solved.

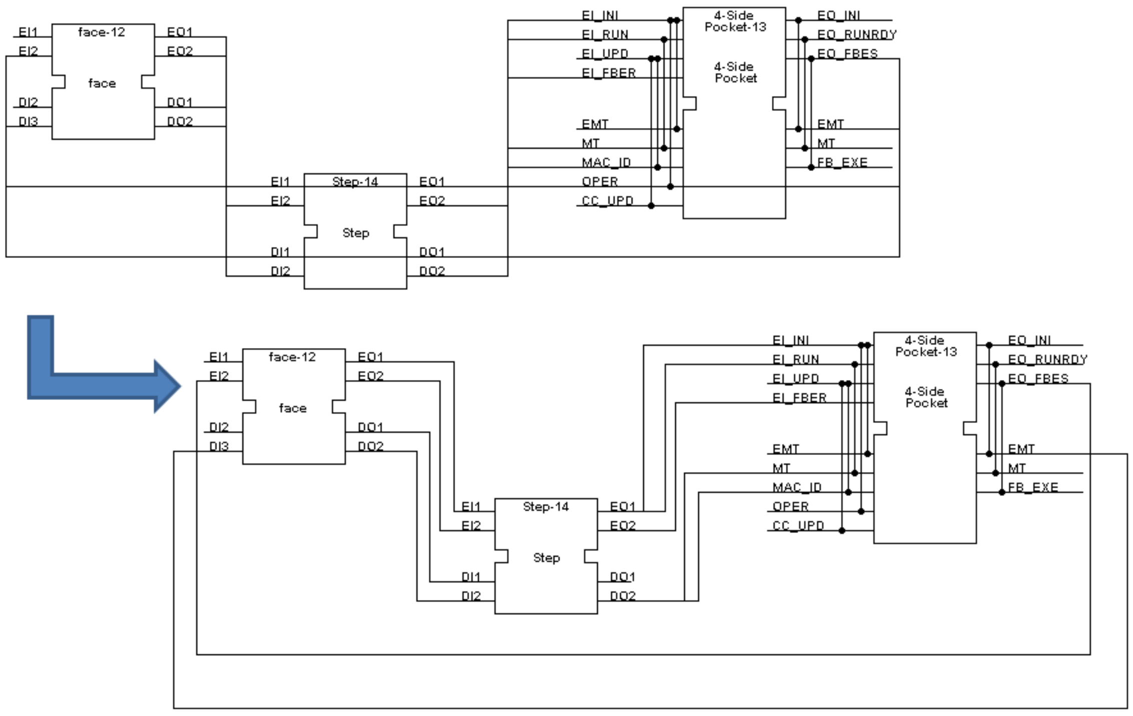


Figure 27: Connection´s drag and drop application

4. Implementation

In the following section, how the project implementation has been carried out will be explained. The whole implemented functionality has been programmed by means of Java. Therefore, at least, a basic Java programming knowledge is essential for understanding the implementation. In the previous chapter, how the development could be performed and different methods to reach it have been discussed. Now, how the development of the application has been really performed and the chosen ways to performing this implementation will be analyzed.

4.1 Research

Most of the main research on Function Blocks has been performed through internet, books and articles. No information has been found in journals or encyclopedias for instance. The result of such great research is found in the literature survey. There can be found any kind of information about Function Blocks and its environment, the International Standard IEC 61499, web-based environments or, even, XML files.

4.2 Chosen programming language: "Java"

For the implementation of the added improvements to the thesis, as mentioned, java programming language has been selected. It has been the key during the implementation of the functionality. Java, originally called OAK, is an object-oriented programming language and computing platform developed by Sun Microsystems in the early 90´s (Java, u.d.). It is a programming language which can be used in any operating system. Besides, java technology´s versatility, platform portability, security and efficiency make it an excellent technology in network computing world (Java, u.d.). These are one of the reasons of its big utilization all over the word and that is why it has been the chosen programming language for the interface development. In spite of the fact that its syntax is very similar to the C or C++ (well-known languages), its object model and fewer low-level facilities make Java being a simpler programming language (Schneider, u.d.). The first version of java was published in 1995, but many newer versions have been published till then.

For the development of a web-based function block environment, java is an adequate programming language because of all its advantages just mentioned. Besides, the implemented functionality was already begun in Java, so there was no reason to select another different programming language. Learning how to use Java language can be found in tutorials, through internet, examples, forums… So, the second step in the project after performing the research on Function Blocks and its run-time environment was learning how to program in Java. Nevertheless, in order to develop the functionality, compile and debug it, a compiler program is necessary: "NetBeans" has been selected.

4.2.1 NetBeans

"The NetBeans IDE is a modular, standard-based, Integrated Development Environment (IDE) written in Java programming language. The NetBeans project consists of an open source IDE and an application platform, which can be used as a generic framework to build any kind of application (Netbeans, u.d.)*"*.

The applications programmed in NetBeans, available for Windows, Mac, Linux and Solaris, can be programmed in different languages: Ruby, Groovy, C/C++, PHP, java platform… This thesis has been developed in NetBeans 7.1, last version, by means of Java platform. The developed functionality can be found in appendices section.

4.3 Implementation of XML document

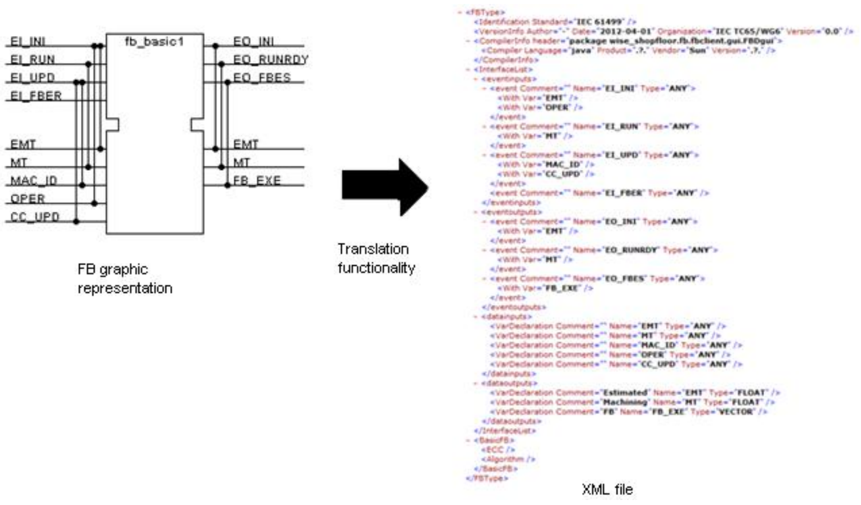
In methodology section two different ways of translating information into a XML file have been discussed. After a deep analysis, using the DOM parser method has been selected in order to transfer data between the program and the XML file. It will be used, well for the XML document generation, and well for the XML document reading. The fact that with DOM parser objects are created by default makes programming easier comparing to the SAX parser. Furthermore, it is a simpler (Harold, 2012) and more used method; so, much more information can be found comparing to the SAX parser. Regarding to memory space, XML files that will be generated from the program are not so large. Thus, there should not be any problem to use DOM parser method.

On the other hand, a security application has been inserted when creating a new project or opening a saved one. After pressing “new” or “open” buttons and before erasing the drawn panel, a menu appears asking if the user wants to save the last project.

4.3.1 Implementation of the basic FB XML file generation

First of all, basic function block´s XML file generation has been implemented. A new button has been added to the interface (“generate XML”) within basic function block tab. A new function (“saveAsXML_basic”) has been set up in “BasicFBEditor.java” class. Then this class calls to another function in a new class, FunctionBlock.java (extended from FeatureData.java). The name of this function is “SaveinXML_BasicFB” and it generates data into XML format. Within this function, DOM parser functionality and a transformer can be found.

DOM parser is in charge of the establishment of elements and their attributes in the XML document that is going to be generated. The structure of the document is generated according to the international standard IEC 61499. This means that normative information (required elements, structure, attributes…) has been collected from the IEC 61499 standard (International Electrotechnical Commission, 2005). Event-data I/O information and ECC information are recollected. On the other hand, the second part of “saveAsXML” function consists of a transformer which provides the tree shape to the created XML document. An instance of this abstract class can transform a source tree into a result tree. Figure 28 shows how the result of this implementation looks like. For more information about the functionality of XML generation, the implemented java code in NetBeans is attached in appendices.



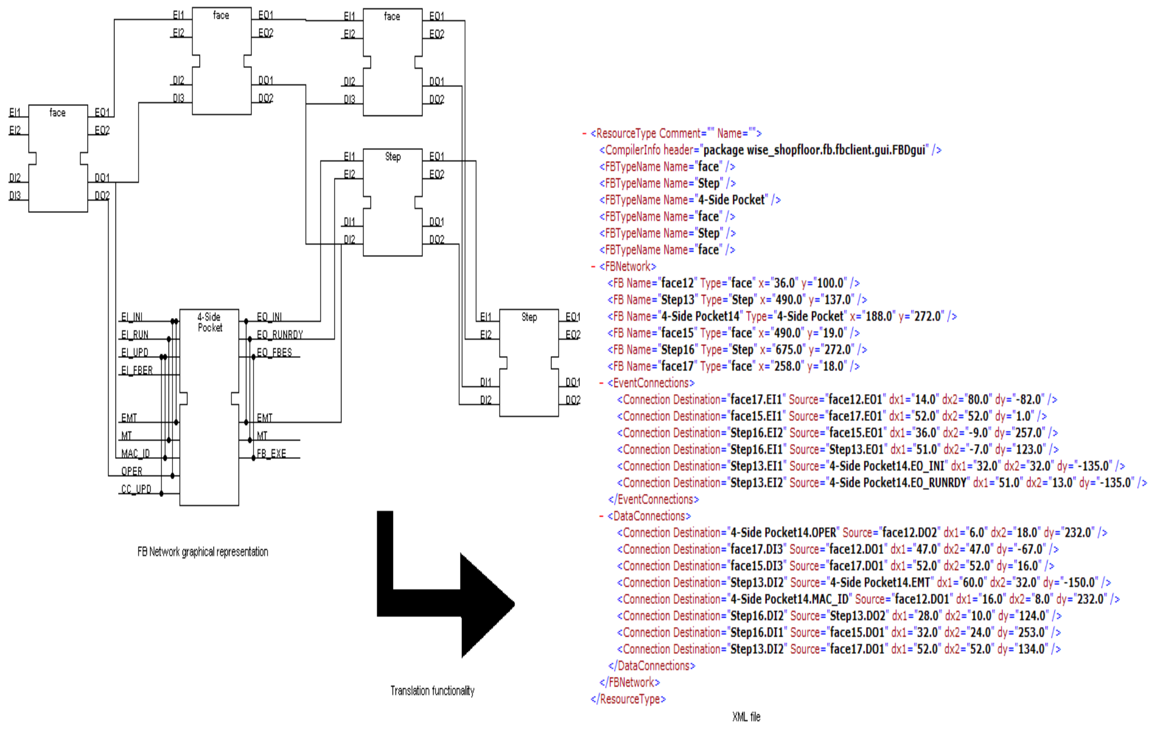
```xml
- <FBType>
    <Identification Standard="IEC 61499" />
    <VersionInfo Author="-" Date="2012-04-01" Organization="IEC TC65/WG6" Version="0.0" />
  - <CompilerInfo header="package wise_shopfloor.fb.fbclient.gui.FBDgui">
      <Compiler Language="java" Product=".?." Vendor="Sun" Version=".?." />
    </CompilerInfo>
  - <InterfaceList>
    - <eventinputs>
      - <event Comment="" Name="EI_INI" Type="ANY">
          <With Var="EMT" />
          <With Var="OPER" />
        </event>
      - <event Comment="" Name="EI_RUN" Type="ANY">
          <With Var="MT" />
        </event>
      - <event Comment="" Name="EI_UPD" Type="ANY">
          <With Var="MAC_ID" />
          <With Var="CC_UPD" />
        </event>
        <event Comment="" Name="EI_FBER" Type="ANY" />
      </eventinputs>
    - <eventoutputs>
      - <event Comment="" Name="EO_INI" Type="ANY">
          <With Var="EMT" />
        </event>
      - <event Comment="" Name="EO_RUNRDY" Type="ANY">
          <With Var="MT" />
        </event>
      - <event Comment="" Name="EO_FBES" Type="ANY">
          <With Var="FB_EXE" />
        </event>
      </eventoutputs>
    - <datainputs>
        <VarDeclaration Comment="" Name="EMT" Type="ANY" />
        <VarDeclaration Comment="" Name="MT" Type="ANY" />
        <VarDeclaration Comment="" Name="MAC_ID" Type="ANY" />
        <VarDeclaration Comment="" Name="OPER" Type="ANY" />
        <VarDeclaration Comment="" Name="CC_UPD" Type="ANY" />
      </datainputs>
    - <dataoutputs>
        <VarDeclaration Comment="Estimated" Name="EMT" Type="FLOAT" />
        <VarDeclaration Comment="Machining" Name="MT" Type="FLOAT" />
        <VarDeclaration Comment="FB" Name="FB_EXE" Type="VECTOR" />
      </dataoutputs>
    </InterfaceList>
  - <BasicFB>
      <ECC />
      <Algorithm />
    </BasicFB>
  </FBType>
```

Figure 28: XML generation when saving a Basic FB

4.3.2 Implementation of an application XML file generation

This section is related to the previous section. This application is divided in two steps. First of all, all created data is saved into vectors. Then, an XML document is generated by means of another function. A similar method as for basic Function Blocks has been utilized for application´s XML generation. In fact, in order to provide Basic FB data within the network, the same function as in point 4.3.2 is called: "SaveinXML_BasicFB". This function is called from "generate_connection_XML" method, where a DOM parser and a transformer are found. This last method has been set up for the XML writing. However, this XML functionality is filled in another way because elements and XML structure for FB Networks is different comparing with basic function blocks. Different information is required according to the IEC 61499. As mentioned, composite FB XML also includes data containing basic FBs.

Before generating the document, some parameters must be identified. On the one hand, information related to function blocks should be identified, such as, function blocks name, type, event-data input/output information… On the other hand, representing data about connections within the FB Network is essential, such us, connection begin output and end input, data or event I/O, which function block do I/Os belong to… Therefore, three more classes have been created: "Connections.java", "FunctionBlock.java" and "Network.java". Within "Connections.java", connections´ coordinates and their begin and ending connectors and Function Blocks are captured; in "FunctionBlock.java", instead, all functions from "FeautureData.java" (a class where methods related to I/O are established) are acquired, such as, getting event and data I/O, updating them, adding them to the FB…; the third new class, "Network.java" contains two important functions. One of the functions is the just mentioned translating functionality "generate_connection_XML". The objective of the other function ("createNetwork") is inserting and saving all the required data (all data-event I/Os, coordinates, number of Function blocks…) into two vectors. In it, instances of the other two classes are generated. This saved information will be collected from the "generate_connection_XML" function in order to write it in the XML document. The three java classes are called from "CompositeFBEditor.java" when a saving button is clicked. At the same time, data is saved and the Extensible Markup Language file is generated. An example of how the created file would look like can be observed in Figure 29. The whole implemented functionality can be found in appendices.


face
Step
4-Side Pocket
FB Network graphical representation
Translation functionality
XML file

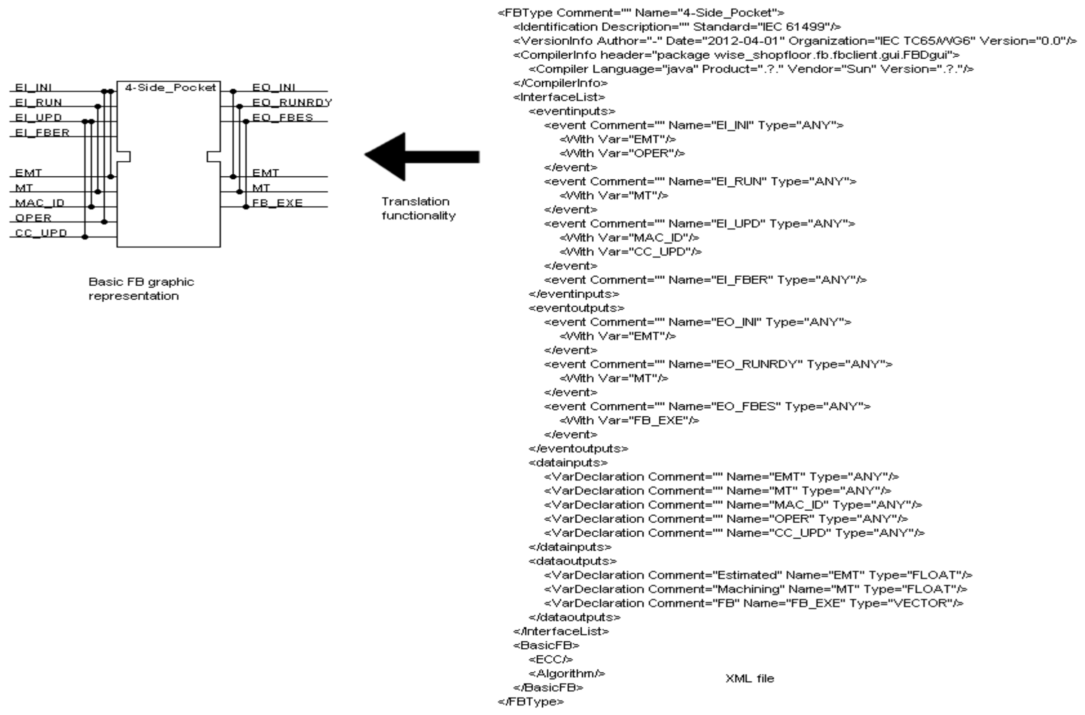
```xml
- <ResourceType Comment="" Name="">
    <CompilerInfo header="package wise_shopfloor.fb.fbclient.gui.FBDgui" />
    <FBTypeName Name="face" />
    <FBTypeName Name="Step" />
    <FBTypeName Name="4-Side Pocket" />
    <FBTypeName Name="face" />
    <FBTypeName Name="Step" />
    <FBTypeName Name="face" />
  - <FBNetwork>
      <FB Name="face12" Type="face" x="36.0" y="100.0" />
      <FB Name="Step13" Type="Step" x="490.0" y="137.0" />
      <FB Name="4-Side Pocket14" Type="4-Side Pocket" x="188.0" y="272.0" />
      <FB Name="face15" Type="face" x="490.0" y="19.0" />
      <FB Name="Step16" Type="Step" x="675.0" y="272.0" />
      <FB Name="face17" Type="face" x="258.0" y="18.0" />
    - <EventConnections>
        <Connection Destination="face17.EI1" Source="face12.EO1" dx1="14.0" dx2="80.0" dy="-82.0" />
        <Connection Destination="face15.EI1" Source="face17.EO1" dx1="52.0" dx2="52.0" dy="1.0" />
        <Connection Destination="Step16.EI2" Source="face15.EO1" dx1="36.0" dx2="-9.0" dy="257.0" />
        <Connection Destination="Step16.EI1" Source="Step13.EO1" dx1="51.0" dx2="-7.0" dy="123.0" />
        <Connection Destination="Step13.EI1" Source="4-Side Pocket14.EO_INI" dx1="32.0" dx2="32.0" dy="-135.0" />
        <Connection Destination="Step13.EI2" Source="4-Side Pocket14.EO_RUNRDY" dx1="51.0" dx2="13.0" dy="-135.0" />
      </EventConnections>
    - <DataConnections>
        <Connection Destination="4-Side Pocket14.OPER" Source="face12.DO2" dx1="6.0" dx2="18.0" dy="232.0" />
        <Connection Destination="face17.DI3" Source="face12.DO1" dx1="47.0" dx2="47.0" dy="-67.0" />
        <Connection Destination="face15.DI3" Source="face17.DO1" dx1="52.0" dx2="52.0" dy="16.0" />
        <Connection Destination="Step13.DI2" Source="4-Side Pocket14.EMT" dx1="60.0" dx2="32.0" dy="-150.0" />
        <Connection Destination="4-Side Pocket14.MAC_ID" Source="face12.DO1" dx1="16.0" dx2="8.0" dy="232.0" />
        <Connection Destination="Step16.DI2" Source="Step13.DO2" dx1="28.0" dx2="10.0" dy="124.0" />
        <Connection Destination="Step16.DI1" Source="face15.DO1" dx1="32.0" dx2="24.0" dy="253.0" />
        <Connection Destination="Step13.DI2" Source="face17.DO1" dx1="52.0" dx2="52.0" dy="134.0" />
      </DataConnections>
    </FBNetwork>
  </ResourceType>
```

Figure 29: XML generation when saving a FB application

4.3.3 Implementation of the basic FB XML file reading

As explained in the chapter of Methodology, this application aims to open XML documents so that the user can edit them. It is just the reverse way of Section 4.3.1. For this, a function called "basicfunction_xml_read_and_draw" has been set up in BasicFBEditor.java class. Within this function, DOM parser functionality has been redacted. A function within FunctionBlock.java class, "ReadfromXML_BasicFB" is called to read the XML document. Furthermore, data-event I/O tables are also created while capturing data from the XML file. For this, new rows filled with information are added to the corresponding table. The transferred information about I/Os is the following: name, type, comments and data-event associations, if exists any. Data of algorithms and ECC (states, actions, transitions, events…) is also captured. With this information the Execution Control Chart of the corresponding basic function block is created and represented.

When the user desires to open a saved XML file, it must only click on "open XML file" button situated on the top of the screen menu. That button calls its own method ("jButton18ActionPerformed") located in "BasicFBEditor" class, which has two purposes. On the one hand, function "ReadfromXML_BasicFB" is called, which reads the file. Then, I/Os tables get new rows and values by means of "basicfunction_xml_read_and_draw". On the other hand, "jButton18ActionPerformed", also updates all I/Os tables showing the new data visualization. Apart from providing this information in the table, a graphical representation of the function block is performed as well. This is obtained automatically by means of another drawing function which gets the information from I/Os tables. It must be noted that this drawing function was already developed during the previous stage of the research. Figure 30 shows how the FB drawing of its corresponding XML document would look like.



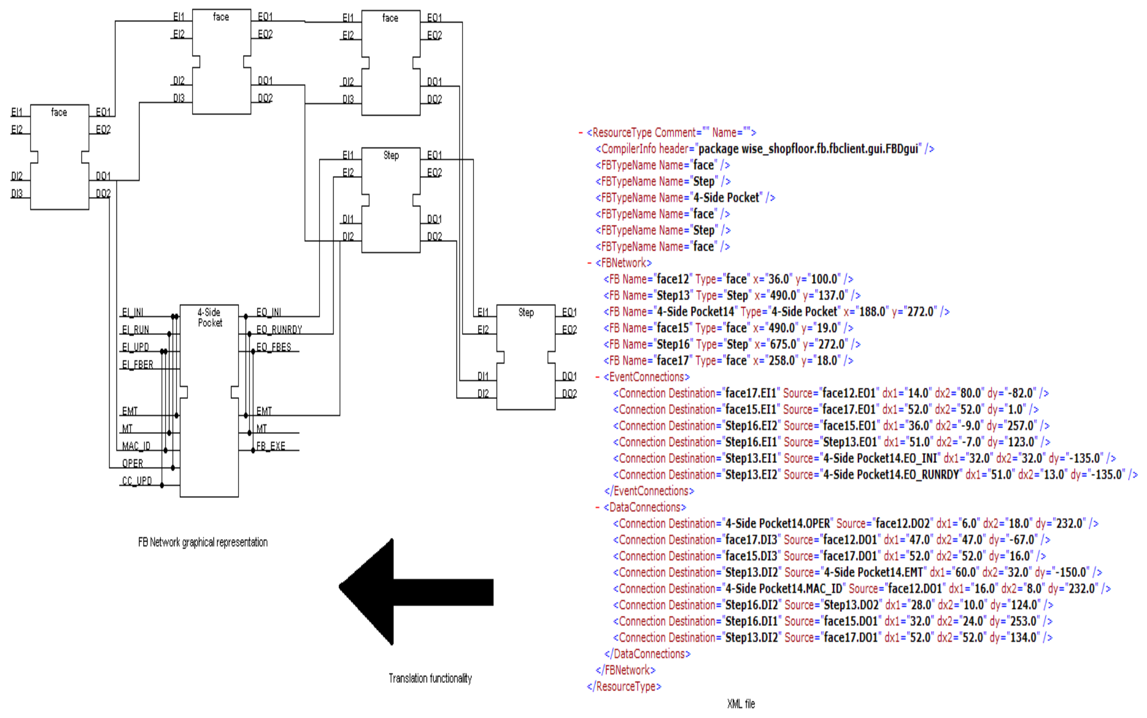
```
<FBType Comment="" Name="4-Side_Pocket">
  <Identification Description="" Standard="IEC 61499"/>
  <VersionInfo Author="-" Date="2012-04-01" Organization="IEC TC65/WG6" Version="0.0"/>
  <CompilerInfo header="package wise_shopfloor.fb.fbclient.gui.FBDgui">
    <Compiler Language="java" Product=".?." Vendor="Sun" Version=".?."/>
  </CompilerInfo>
  <InterfaceList>
    <eventinputs>
      <event Comment="" Name="EI_INI" Type="ANY">
        <With Var="EMT"/>
        <With Var="OPER"/>
      </event>
      <event Comment="" Name="EI_RUN" Type="ANY">
        <With Var="MT"/>
      </event>
      <event Comment="" Name="EI_UPD" Type="ANY">
        <With Var="MAC_ID"/>
        <With Var="CC_UPD"/>
      </event>
      <event Comment="" Name="EI_FBER" Type="ANY"/>
    </eventinputs>
    <eventoutputs>
      <event Comment="" Name="EO_INI" Type="ANY">
        <With Var="EMT"/>
      </event>
      <event Comment="" Name="EO_RUNRDY" Type="ANY">
        <With Var="MT"/>
      </event>
      <event Comment="" Name="EO_FBES" Type="ANY">
        <With Var="FB_EXE"/>
      </event>
    </eventoutputs>
    <datainputs>
      <VarDeclaration Comment="" Name="EMT" Type="ANY"/>
      <VarDeclaration Comment="" Name="MT" Type="ANY"/>
      <VarDeclaration Comment="" Name="MAC_ID" Type="ANY"/>
      <VarDeclaration Comment="" Name="OPER" Type="ANY"/>
      <VarDeclaration Comment="" Name="CC_UPD" Type="ANY"/>
    </datainputs>
    <dataoutputs>
      <VarDeclaration Comment="Estimated" Name="EMT" Type="FLOAT"/>
      <VarDeclaration Comment="Machining" Name="MT" Type="FLOAT"/>
      <VarDeclaration Comment="FB" Name="FB_EXE" Type="VECTOR"/>
    </dataoutputs>
  </InterfaceList>
  <BasicFB>
    <ECC/>
    <Algorithm/>
  </BasicFB>
</FBType>
```

Figure 30: XML reading and drawing when opening a Basic FB

4.3.4 Implementation of the application XML file reading

In 4.3, it has shown the implementation of the application that relates to the function block program and XML documents. In what follows, the reading of application´s (composite FB, subsystems and systems) XML files in order to open and edit them will be discussed. Obviously, applications contain diverse information as previously explained in Section 4.3.2. Information about function blocks within the network as well as data about connections in the FB network. An application has been implemented for reading XML documents that contain mentioned data according to the standard. The result is visualized in Figure 31; nevertheless, as explained, data about each function block within the network should also appear.

Used functions, "ReadfromXML_compositeFB" and "updateDrawing", are developed in "Network.java" class. On the one hand, DOM parser, which provides the reading of the parameters from these kinds of documents, is located in "ReadfromXML_compositeFB". An instance of "FunctionBloc.java" class (called "fblock") has been created to set all FB values in the document to a function block. Other two instances ("event_conn" and "data_conn") of "Connections.java" class have been created to set connections data from the XML file on them. Then, "fblock", "event_conn" and "data_conn" have been added to the two vectors created in "Network": "FBs_vector" and "Connections_vector". It has to be mentioned that "ReadfromXML_compositeFB" calls to "ReadfromXML_BasicFB" in FunctionBlock.java in order to get Basic FB data. When all data and values of the FB network are set, drawing is the only action left to be performed. That is just "updateDrawing" method´s purpose. It accomplishes the drawing of all connections and function blocks within the network in the web based environment. This function calls a "Drawing" class which performs the real drawing action.



```
- <ResourceType Comment="" Name="">
    <CompilerInfo header="package wise_shopfloor.fb.fbclient.gui.FBDgui" />
    <FBTypeName Name="face" />
    <FBTypeName Name="Step" />
    <FBTypeName Name="4-Side Pocket" />
    <FBTypeName Name="face" />
    <FBTypeName Name="Step" />
    <FBTypeName Name="face" />
  - <FBNetwork>
      <FB Name="face12" Type="face" x="36.0" y="100.0" />
      <FB Name="Step13" Type="Step" x="490.0" y="137.0" />
      <FB Name="4-Side Pocket14" Type="4-Side Pocket" x="188.0" y="272.0" />
      <FB Name="face15" Type="face" x="490.0" y="19.0" />
      <FB Name="Step16" Type="Step" x="675.0" y="272.0" />
      <FB Name="face17" Type="face" x="258.0" y="18.0" />
    - <EventConnections>
        <Connection Destination="face17.EI1" Source="face12.EO1" dx1="14.0" dx2="80.0" dy="-82.0" />
        <Connection Destination="face15.EI1" Source="face17.EO1" dx1="52.0" dx2="52.0" dy="1.0" />
        <Connection Destination="Step16.EI2" Source="face15.EO1" dx1="36.0" dx2="-9.0" dy="257.0" />
        <Connection Destination="Step16.EI1" Source="Step13.EO1" dx1="51.0" dx2="-7.0" dy="123.0" />
        <Connection Destination="Step13.EI1" Source="4-Side Pocket14.EO_INI" dx1="32.0" dx2="32.0" dy="-135.0" />
        <Connection Destination="Step13.EI2" Source="4-Side Pocket14.EO_RUNRDY" dx1="51.0" dx2="13.0" dy="-135.0" />
      </EventConnections>
    - <DataConnections>
        <Connection Destination="4-Side Pocket14.OPER" Source="face12.DO2" dx1="6.0" dx2="18.0" dy="232.0" />
        <Connection Destination="face17.DI3" Source="face12.DO1" dx1="47.0" dx2="47.0" dy="-67.0" />
        <Connection Destination="face15.DI3" Source="face17.DO1" dx1="52.0" dx2="52.0" dy="16.0" />
        <Connection Destination="Step13.DI2" Source="4-Side Pocket14.EMT" dx1="60.0" dx2="32.0" dy="-150.0" />
        <Connection Destination="4-Side Pocket14.MAC_ID" Source="face12.DO1" dx1="16.0" dx2="8.0" dy="232.0" />
        <Connection Destination="Step16.DI2" Source="Step13.DO2" dx1="28.0" dx2="10.0" dy="124.0" />
        <Connection Destination="Step16.DI1" Source="face15.DO1" dx1="32.0" dx2="24.0" dy="253.0" />
        <Connection Destination="Step13.DI2" Source="face17.DO1" dx1="52.0" dx2="52.0" dy="134.0" />
      </DataConnections>
    </FBNetwork>
  </ResourceType>
```

Figure 31: XML reading and drawing when opening a FB application

4.4 Connections representation

This section presents how the implementation is compliant with the international standard IEC 61499. Connections representation within FB Networks has been implemented again, but this time, according to the international standard IEC 61499. For this, a new java class has been created: “StandarizedConnection”. Within this java class a simple functionality is implemented. There are two types of connections: three lines containing connections and five lines containing connections, as shown in Figure 32. Detecting if the beginning connection is at the left or right side of the ending connector classifies connections into one or another connection type. Some coordinates, points in the drawing area, must be set up in the interface: dx1, dx2 and dy. They provide information about beginning and ending connector’s location. By means of this coordinate system, some lines are created in order to establish the connection between two function blocks. Indeed, these points are the same coordinates represented in application´s XML documents. In the figure below, it can be observed which point represents each parameter (dx1, dx2, dy).

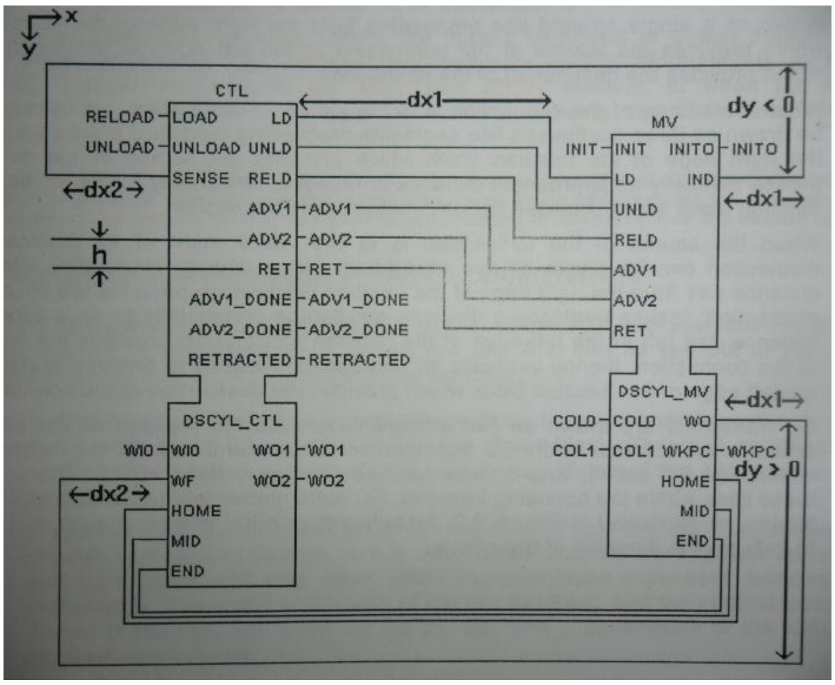


Figure 32: Coordinate system of a FB Network (International Electrotechnical Commission, 2005)

As it can be observed in Figure 32, a connection will be three lines connection type if the beginning output is situated at the left side of the ending output. Nevertheless, if the beginning output is located at the right side of the ending input, five lines will be necessary to be compliant with the standard's requirements.

4.5 File selection menu

The next part is focused on opening previously created function block programs within the web based application. In this particular case, XML documents are the only documents which can be opened, either basic FB or applications. When clicking opening button, in any kind of program, usually a menu appears. This menu is use to display the entire desired library (all available XML documents in this case). So, the user has the chance to choose which file to open. This is the real aim of the following implemented application: to show a file selection menu, before opening any document. Figure 33 shows how the result looks like.

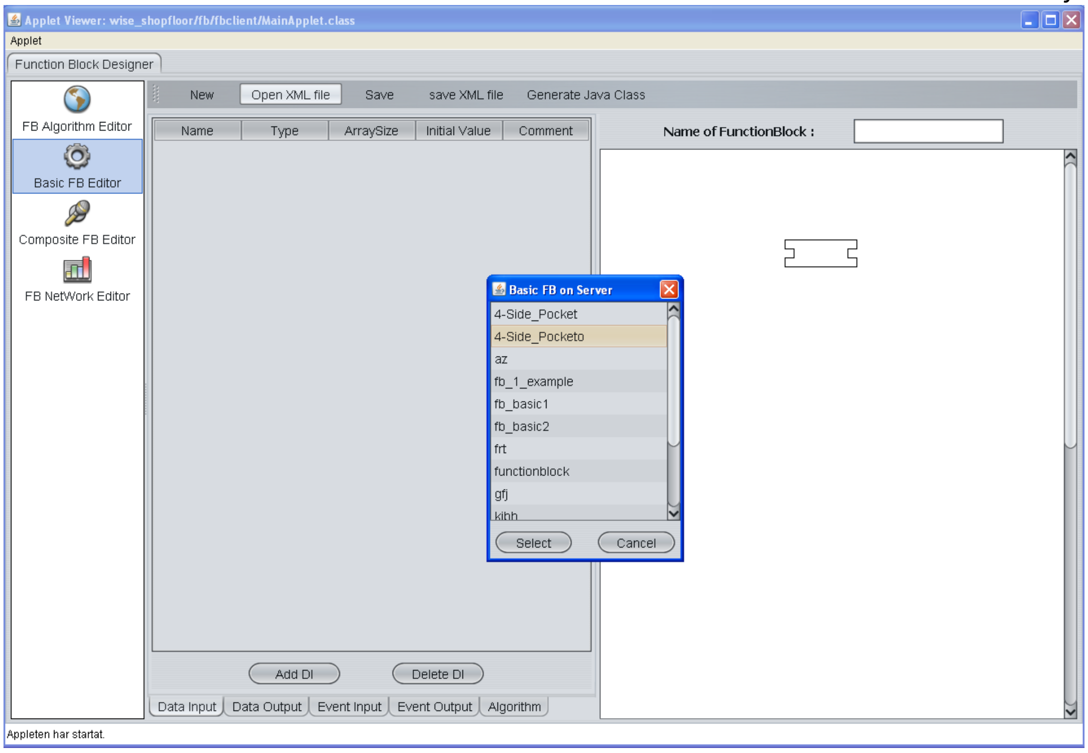


Figure 33: File selection menu

In order to implement this particular application another java project has been necessary. Its name is "FBweb" and it establishes the connection between the program and the server. It provides performing work through the net and even programming into the web based environment. So, this second project´s use is essential in order to save a program within the web. Obviously, exactly the same happens when a program saved in the server ought to be opened. This last action requires the representation of all XML documents within the server. That is why these two interconnected projects have been used. An opening button calls to a function within "BasicFBEditor.java" class named "action_Open". Apart from creating the new menu window, this function also enables "RerieveDirectory" method. This function is responsible for sending request to the server and retrieves answer as a list of files in the specified directory. On the server side, class FBServlet_FB02 from FBweb application is in charge of receiving this request, getting a list of files and sending it back.

4.6 Implementation of Composite FB Editor

In the methodology part, it has been discussed about the two possible options to be carried out in the Composite FB editor. Using arrow shaped figures (second option) has been the final solution in order to obtain an easier representation. User´s I/Os desired location is available by this way. For this, a java class called "InterfaceFigure.java" has been used. This java class contains all the features the interface figure needs: functions defining the drawing, type (data or event I/Os), name, measures…. The appearance of interface I/Os is shown in Figure 34. It can be visualized how I/Os arrows always point to the FB Network.

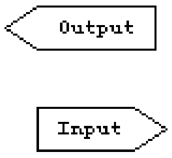


Figure 34: Result of InterfaceFigure.class

Interface figures are divided into four categories: data inputs or outputs and event inputs or outputs. As discussed, figures should be located anywhere, so a drag and drop application has been added to the figure so that the user can choose where to position it. After having created the figure, a connector had to be set up within it in order to establish connections with FB Network connectors. Connector.java is the class that creates a connector inside the figure. In this class begin and end connectors are set up.

An instance of the final shape of a FB application can be observed in Figure 35. It can be seen external I/Os at both sides of the FB Network and, even connections between FB network connectors and external connectors of the application.

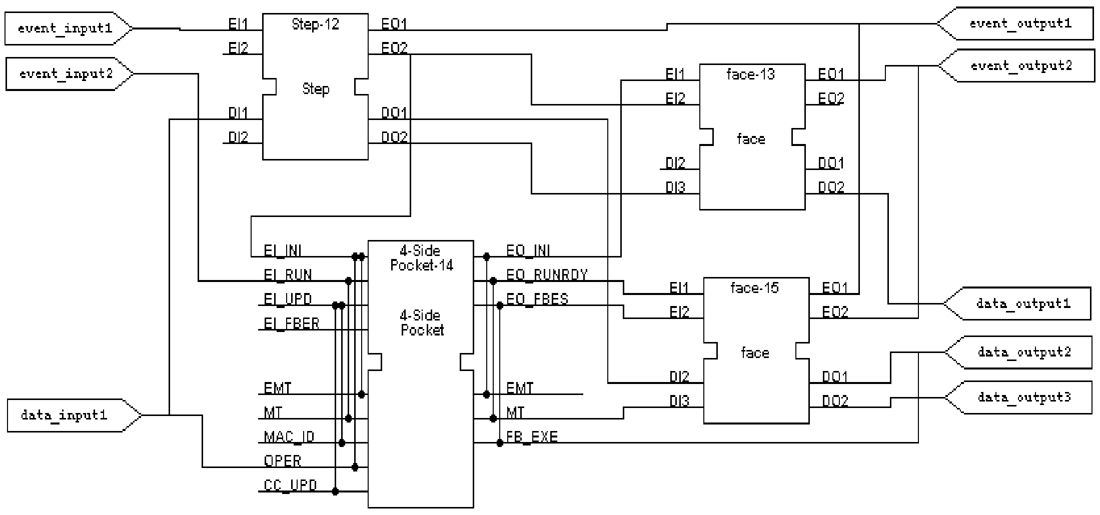


Figure 35: Example of a FB application within the web based environment

4.7 Saved Function Blocks cloning

All saved Basic and Composite function blocks are cloned in order to be reused in the FB Network. For Basic FBs cloning, a function named “createToolIcons” in CompositeFBEditor.java clones all saved FBs in the server and then calls to another function in the same class, “addToolIcon”, to add an icon at the left panel of FB Network Editor. For Composite FBs, instead, the cloning function is called “createToolIcons_CompositeFB” and the name of the function that creates icons is “addToolIcon_Composite”.

Automatically, when initializing the program all saved FBs icons appear at left panel of FB Network Editor as it can be seen in Figure 36. This panel is divided into two tabs in order to differentiate between Composite (CF Blocks) and Basic (BF Blocks) saved Function Blocks. Moving the mouse arrow over the FB icons, their names appear at the bottom of the panel.

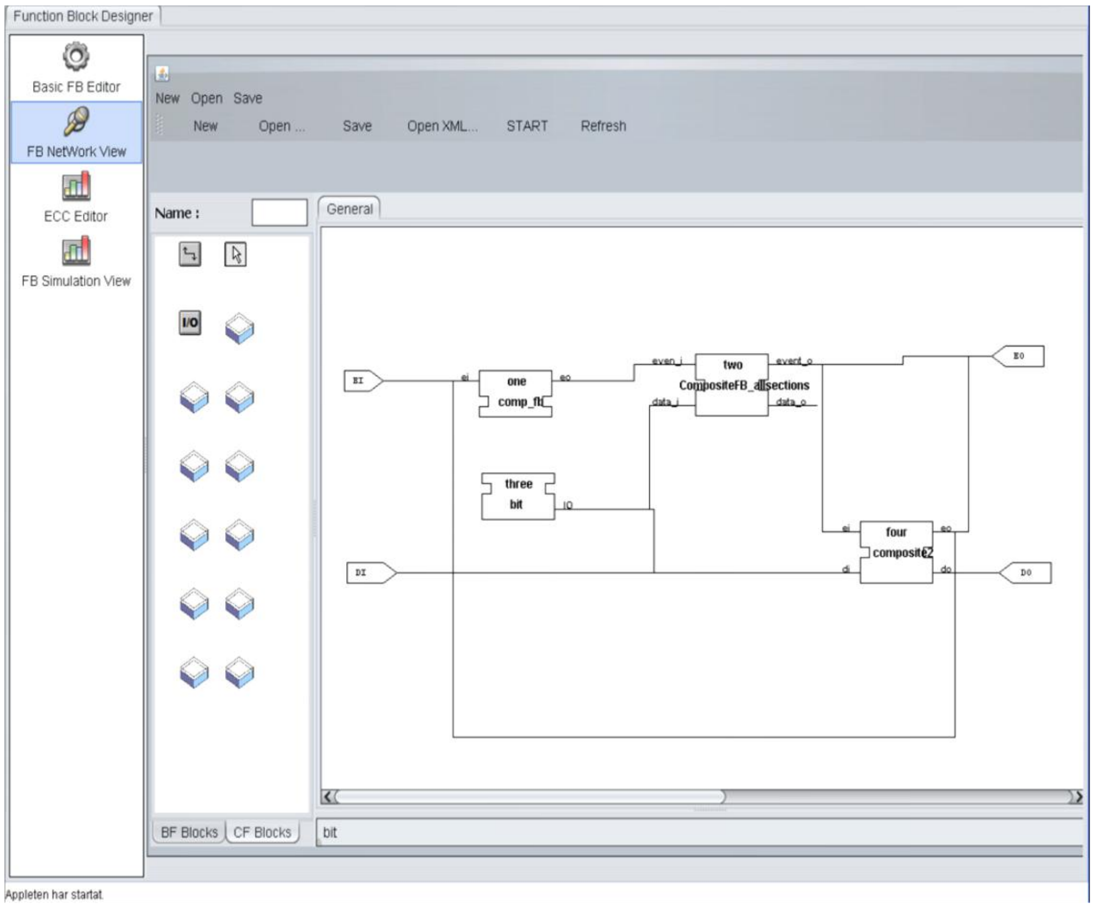


Figure 36: FBs cloning and icons creation

Apart from being initialized all icons when opening the program, a new button called "Refresh" has been integrated which aim is to initialize and recreate all the saved FBs cloning. This button is useful when a new FB has been created and the user wants to use it within the FB Network.

5. Case Study

This part was implemented together with Aitor Arrieta Marcos (Arrieta, 2012).

After finishing any kind of work, a thesis in this case, it is always essential to perform some tests in order to know if everything works properly. In this specific case, an additional part has been joined to the project, apart from the completed challenges. This part is the run-time of the program implemented by Bernard Schmidt, supervisor of this thesis work. The simulation, which is found in simulation tab, will be used as a proofing tool in order to verify the positive results of the thesis work. Using the simulation tool, created and saved FB applications or systems can be run. The entire system is represented, as in Composite FB Editor, indicating with a red light which function block is running at each moment.

So, once the FB platform was implemented and ready to be used, an application has been thought to test using the run-time tool and present how the implemented environment should work. An assembly process has been decided to be implemented by means of Function Blocks, using an ABB IRB 140 robot (Figure 37) already integrated in Wise-ShopFloor. It is being researched so that the FB environment could enable *"approach to achieving adaptability and flexibility in assembly planning and control"* (Wang, et al., 2010). In this article just mentioned, some new developments are presented using the ABB IRB 140 "*robot mini-cell for testing and validation of a FB enabled assembly planning"*.

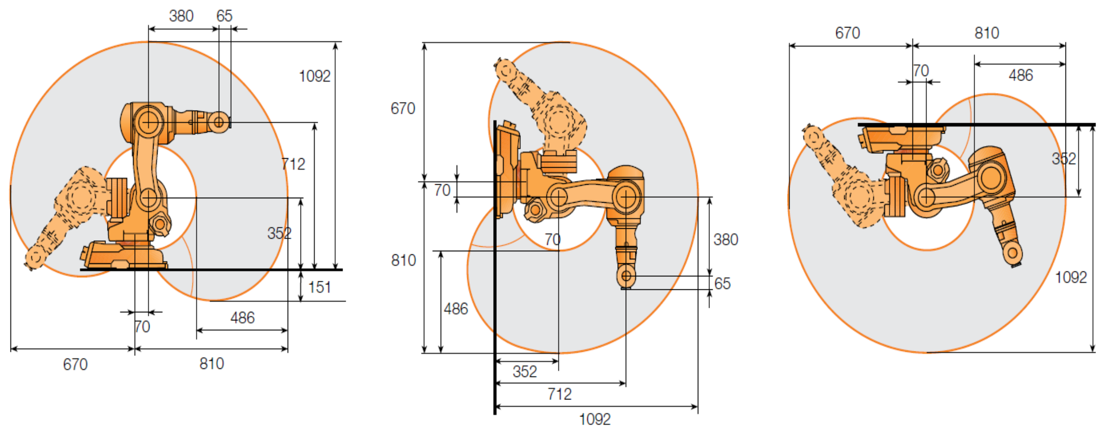


Figure 37: Sizes, configurations and the region of space the robot can reach for each configuration (ABB, 2012)

In (Wang, et al., 2010), a case study is implemented in which the simulink blocks of a simulation in Matlab Simulink replace the FBs to control an ABB robot. The outputs of this simulink blocks generate commands of Rapid programming language, language that is understood by the controller of the ABB robot. The goal of the case study of this project has been to simulate the pick and place movements of an assembly process using the ABB IRB 140. Different options have been studied taking into account the time to finish the project.

The assembly operation process simulation will be to pick shafts from their magazines and place them into the measure station; secondly, the robot will pick the washers and place it into the shafts. In this station, shaft´s measurement will be known and depending on the detected measurement, the robot will perform 3 different paths. So, the measurement inspection station will detect if the piece is alright, wrong (in this case the piece is removed from the assembling station) or it has to be fixed (it will be transported to a particular magazine). If the shaft has the correct measures, before leaving it in the corresponding magazine, the robot will transport the shaft and washer (at the same time) to a press station in order to be pressed. Figure 38 shows the robot working during the assembling process.

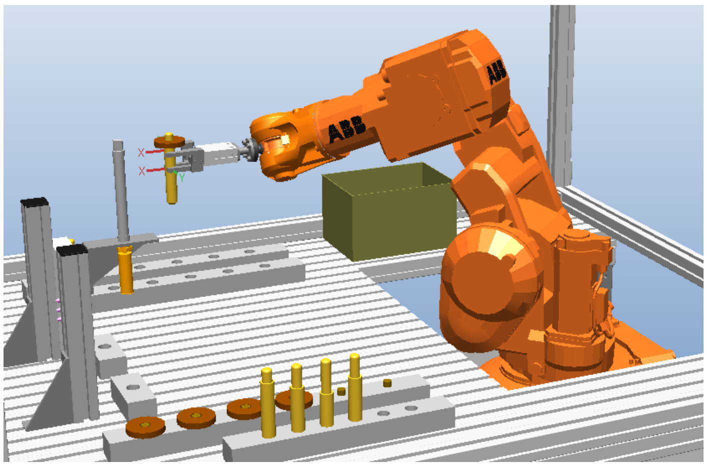


Figure 38: Robot working during the assembling process

This process will be performed one by one in 6 occasions (total number of shafts and washers). On the other hand, Figure 39 represents the flow chart of the assembly process whereas Figure 40 shows the entire mini-cell 3D model.

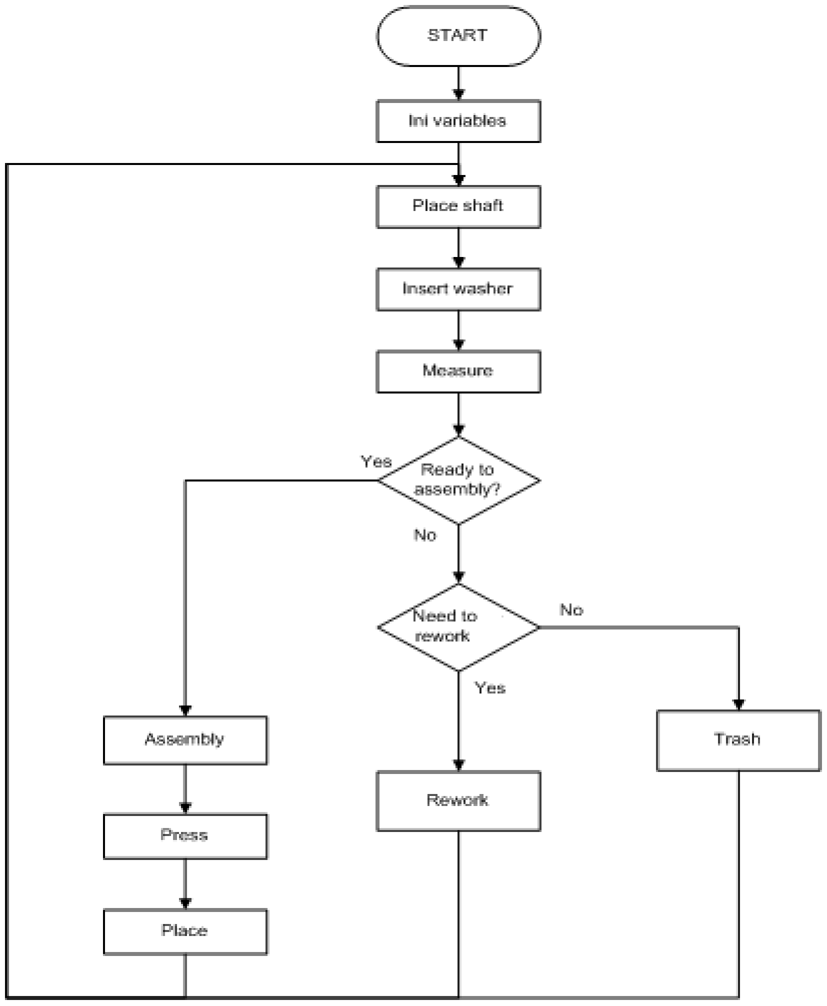


Figure 39: Flow Chart of the simulated assembly line

As mentioned before, different options have been studied in order to carry out the implementation of the case study. One of the options was to implement some function blocks that generate Rapid programming language commands, something similar as explained in (Wang, et al., 2010), in which some simulink blocks actuate as FBs and generate Rapid commands. After that, the next step would be to send these commands to the robotic mini-cell in Wise-Shop floor, to ensure that it is working in a correct way. However, this could not be possible to be performed due to two reasons: the lack of time for this project and the availability of the robot being used in (Cana & Gil, 2012).

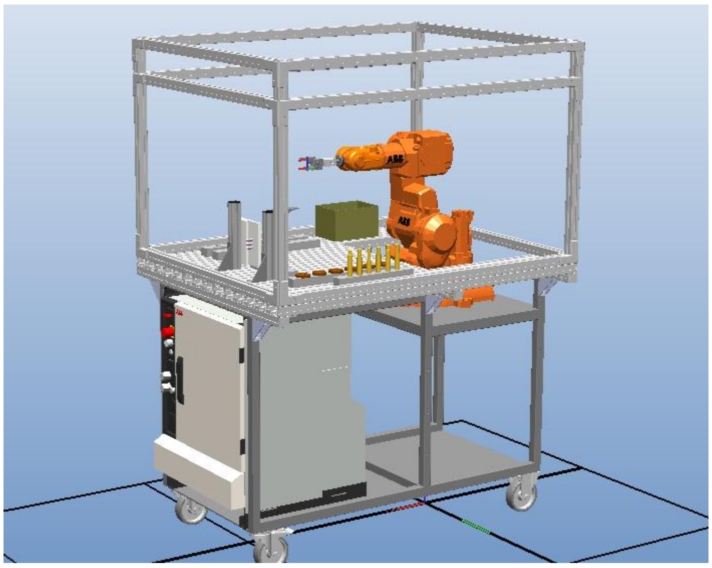


Figure 40: 3D model mini-cell robot assembly station

Another option was to use some specific FBs explained in (Wang, et al., 2010) and (Wang, et al., 2012), where the MH-FB will open and close the fixture, the SI-FB will check the position of the TCP reading the I/Os of the robot´s PLC and the M-FB will *“send the confirmation event to the right composite FB to start its operation”.*

Focusing on the generated project, it consists of ten basic function blocks as it can be seen in Figure 41. Each function block within the created FB Network has a different objective. It cannot be appreciated in the figure but this FB Network simulation is reached visualizing every time which FB is running. As it can be seen, a serial application is effected until the measuring station where three parallel ways appear, depending on the sensors´ signal. Each function blocks´ goal is briefly explained just after the Figure 42.

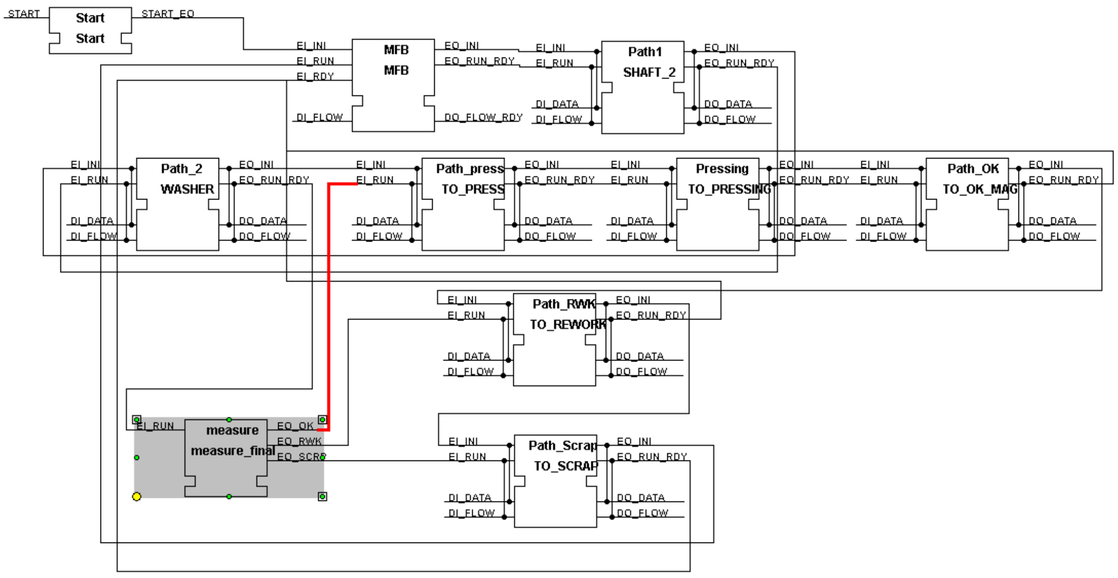


Figure 41: Assembly process running

On the one hand, Figure 41 shows the created FB Network while it is being executed. The red line is the indicator of what part of the network is simulating in each moment. Algorithms in measure FB decides which of the three paths to follow. On the other hand, Figure 42 represents how the real FB Network should look like. On it, all needed data information is updated within function blocks; however, it has been impossible to run it on our program´s run time environment.

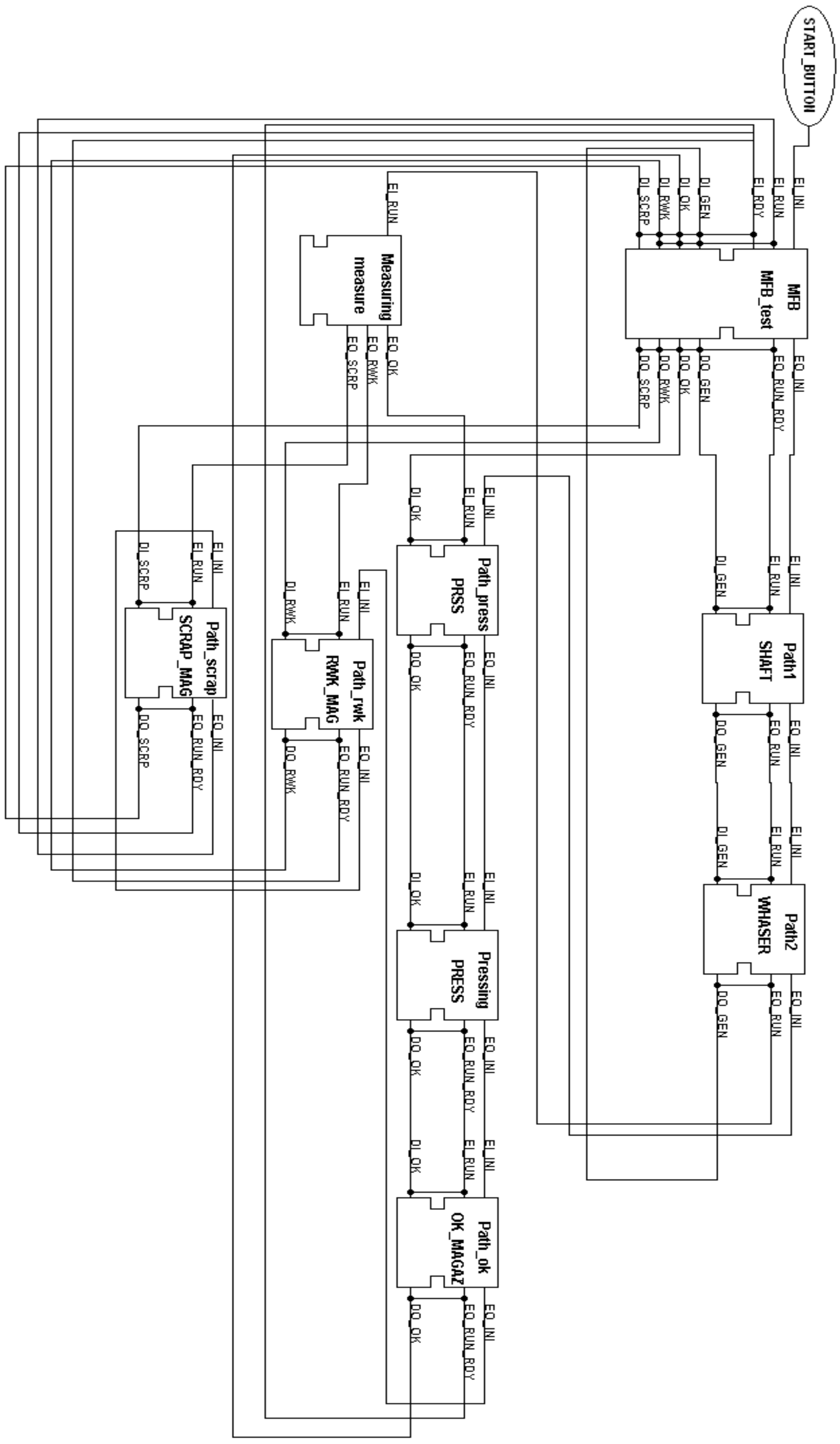


Figure 42: Final shape of assembly process FB Network

This is the list of the created basic function blocks followed by their duties:

- Start: This FB receives the “START” button at the top of the panel as an input providing the beginning of the simulation.

- MFB: As told, this type of FB is in charge of sending the confirmation event to the correct FB in order to enable its operations execution.

- Shaft: This FB is responsible of picking the corresponding shaft from the shaft magazine and placing it at the inspection station. The movement of the robot is created by Rapid language output orders. After this algorithm is executed, an offset is added. This offset reaches the robot to pick the corresponding shaft.

- Washer: This FB works as “SHAFT” function block, however, in this case, the picked and placed object is a washer. This is placed into the shaft at the measure station.

- Measuring: It is the FB that decides which way should get the process. It measures the shaft by means of three sensors. Depending on which of the internal signal is at Boolean “true”, the process will continue a different way.

- To press station: Shaft and washers are picked and placed at the press station. This is the first step of the mentioned first way.

- Pressing: The shaft and washer are pressed by means of two internal variables in the press station: “press down” and “press up”.

- To ok magazine: The robot is ordered to transport the shaft with the washer to the okay magazine.

- To rework magazine: This is the only FB in the second way; it is in charge of picking the shaft with its washer and placing it at the rework magazine.

- Scrap: This FB is also the only component of the third way to chose. It consists of picking bad shafts (with its washer) and leaving then into the scrap.

One of the advantages of using FB technology is that it is object oriented, and this brings lots of advantages to devices to the easy programmed: Objects reflect the real world, that is why the operations of the robot (measure, press, gripper oppen-close, transport...) are represented as objects. This objects are stable, and this allows to program other robots reusing the same objects without changing anything. This is very comfortable if the used robot is broken or has to be used in another part of the factory, because the operator does not have to program again the new robot. Objects reduce complexities, so once the object is created, the user does not have to know how it works internarlly. Furthermore, the application is created joining different objects. Apart of the system reusability, another interesting part of using FBs for this process, is that if the user needs to program any other machine, he/she can get simple function blocks from the system or application. This means that it is possible to get only small parts of the system instead of getting all the program, as explained at the paragraph above. For instance, if a machine needs to be programmed for measuring, only the measuring FB would be got and placed in the new program.

Programming by means of FBs also ensures the possibility of interoperating between embedded devices to perform needed methods and functions for distributed applications.

Besides, if the user desires to run the created program by means of another different simulation program, configurations are possible to be saved in XML file (according to the standard IEC 61499). This saving application reaches that the user can afford a large range of portability, simulating the generated project in any other run time environment.

As it has been shown, the use of FBs in order to program this assembly process provides to the user adaptability, reusability, portability, interoperability and a great degree of independence. On the other hand, talking about the execution model, Function Blocks use the efficient method explained during the thesis: the event-driven model. This would provide a real time control because this model does not need to wait for other tasks to be ended. When the input enables an event, its execution inmediately begins. Finally, as a future improvement of this case study, it would be interesting to put the inplemented FB Network (Figure 42, filled by all needed data) into practice within the run time environment and the real robotic mini-cell. This is possible thanks to the advantage provided by the IEC 61499: configurability. It allows two different programming environments to configure the same device. In this case, the robot could be configured from the developed Function Block platform and Robotstudio programming environment.

6. Discussions

It must be said that good visualization is essential in computer programs. An easy and clear program is always much more understandable for a novice user. Even if it is any kind of user, working on simple programs always is more tolerable than working on difficult and complicated ones. For this reason, the visualization of the user interface is critical when referring to program´s difficulty level. This goal has been unquestionably achieved in this project: A simple program has been implemented.

Nevertheless, from my point of view, the project still could be improved in some aspects. Due to the lack of time in the project, all these applications that will be explained now could not be implemented. Its extension should continue adding some new actions, to enhance an easier visualization and usability. For instance, a recommendable application for the user interface would be the implementation of a function that zooms in and out the FB Network´s size. The wheel in the middle of a computer mouse is usually used for this purpose. Another improving suggestion would be to apply the drag and drop function to Basic Function Block´s data-event connections; since at the moment, it is only possible to join those connectors by means of the text I/Os table. Besides, within Basic Function Block Editor, when opening a saved program, data-event associations appear only at I/Os table; not in the drawing. "Enter" button must be pressed at the table in order to create these connections in the FB image. This should be changed: connections should appear immediately when opening the project. On the other hand, generated XML files have all its elements ordered alphabetically. This should be removed and generated in the same order as in the International Standard IEC 61499.

In the same way, it would also be interesting carrying out an application to be able to change Function Blocks information (name, type, I/O names…) within the graphics panel apart from in data tables. This would help, for example, when a wide range of function blocks are interconnected in a FB Network. Moreover, it would also be nice if within Network Editor the user had the chance of double clicking on a Composite FB and opening its inside network.

Apart from these enhancements, developing automatic routines would be a significant step forward for the program, since all implemented functions are manually performed. These improvements would ease the usability of the program to the human user.

Referring to problems that have appeared during the project, it must be recognized that most of the problems have appeared while programming in Java. First of all, trying to decipher the functionality before implemented, some inconsistencies and many misunderstandings were found. Later on, during the thesis development, problems and errors became the daily bread. Particularly, while compiling the developed program, errors appeared continuously. For instance, a considerable amount of time has been spent during the application´s XML reading and drawing because of the wide range of errors that appeared. But all these challenges, that sometimes looked impossible to resolve, have been overcome with patience and large research.

As regards to the case study, it has provided a clear visualization of this projects result, enabling configuration and simulation. Due to the lack of time, the generated program during the case study has not been downloaded to the robot. But this would also show the connection between the FBs programming web based environment and real machine and devices (robots, PLCs…). So, we have to comply with the simulation.

Finally, it has to be also mentioned, that an informatics engineer would have worked easier and would have had more fun during the development of this project. The fact that I have never worked in java before has complicated the implementation of the thesis. From my honest point of view, the time provided for the development of the final year project (exactly five months) has been quite short. Taking into account the dimension of the thesis and the lack of knowledge in java, a great deal of work has been accumulated in the last months of the project. Perhaps, in order to avoid this issue, some Java courses should be provided to the future final year thesis students.

7. Conclusions

The final year project has been a long and hard programming work that has taught me a great deal of java language apart from learning how to create written documents in English. Honestly, I have reached to develop some programming applications that I could never have thought before I would be able to implement. One of the challenges of this project was learning about Function Blocks, and I have reached this objective. At the beginning of this thesis, I did not really know much about FBs. On the contrary, at the end of the thesis, apart from learning about FBs theory, I have even managed to create a FB application. It explains the progress in FBs. Moreover, I have also learned how to work as a team (do not forget that this project has been developed simultaneously with another automation project) sharing information and knowledge.

Finally, I sincerely hope that the performed work will profitably help in the following years to Wise-ShopFloor research and, hence, to the aimed real-time decision support system.

It must be specified that the created web-based environment can be found in the next URL: wise-shopfloor.his.se/WebFB/

8. Acknowledgments

On the one hand, a special mention has to be awarded to the supervisor of this project, **Bernard Schmidt**. Without his help it would have been impossible to carry out this project. On the other hand, Lihui Wang, an expert on the topic, has to be also mentioned. He has provided some courses during the thesis to teach students how a bachelor final year project should be realized.

Furthermore, I am grateful to have worked with Aitor Arrieta, who has always collaborated and worked as a team. Finally, this document could not be ended without rendering thanks to Markel Garzia Gomez (informatics engineer), for helping me with java programming; and Ainhoa Goienetxea, who has made possible that I could stay during the final year thesis in Skövde. Besides, she has always helped out to me during this period advising and collaborating in relations with the home university.

9. Bibliography

ABB, 2012. *Datasheet robot ABB IRB 140,* s.l.: ABB.

Anon., 2012. *Wikipedia.* [Online]
Available at: http://en.wikipedia.org/wiki/Application_programming_interface
[Accessed 03 04 2012].

Arrieta, A., 2012. *Function block environment in Wise Shopfloor: algorithm parser and code generation,* Skövde: s.n.

Campbell, A. D., 2007. Benefits of web based. *Blue Dog.*

Cervera, A., 2012. *Design of an information carrier device for Decision Support System,* s.l.: s.n.

Christensen, J. H., n.d. *holobloc.com.* [Online]
Available at: http://www.holobloc.com/papers/iec61499/overview.htm
[Accessed 26 02 2012].

Diaz, R. & Palomeque, J. E., 2012. *Robot & CNC Machine Function Blocks,* Skövde: s.n.

Fyicenter, n.d. *fyicenter.* [Online]
Available at: http://dev.fyicenter.com/Interview-Questions/Java-1/Parsers_DOM_vs_SAX_parser.html
[Accessed 27 03 2012].

Gerber, C., Hanisch, H.-M. & Ebbinghaus, S., 2008. From IEC 61131 to IEC 61499 for Distributed Systems: A Case Study. *EURASIP Journal on Embedded Systems,* p. 8.

Google, n.d. *google.com/images.* [Online]
Available at:
http://www.google.se/imgres?um=1&hl=en&biw=1344&bih=767&tbm=isch&tbnid=C-LKPdPvDv4dzM:&imgrefurl=http://seg.ee.upatras.gr/seg/dev/RoboticArm.htm&docid=-jbl7EEep9_4GM&imgurl=http://seg.ee.upatras.gr/seg/dev/media/corfu_app_basic.png&w=1039&h=490&ei=k0yET7V
[Accessed 29 03 2012].

Harold, E. R., 2012. *Document Object Model, Chapter 9: Choosing between SAX and DOM.* s.l.:s.n.

International Electrotechnical Commission, 2005. *International Standard IEC 61499.* First ed. Geneva, Switzerland: IEC.

Isagraf, n.d. *isagraf.com.* [Online]
Available at: http://www.isagraf.com/pages/newsletter/apr2007.htm
[Accessed 11 02 2012].

Isagraf, n.d. *isagraf.com.* [Online]
Available at: http://www.isagraf.com/pages/newsletter/apr2008.htm
[Accessed 05 04 2012].

Java, n.d. *java.com.* [Online]
Available at: http://www.java.com/en/download/faq/whatis_java.xml
[Accessed 11 02 2012].

Java, n.d. *java.com.* [Online]
Available at: http://www.java.com/en/about/
[Accessed 11 02 2012].

Keshavarzmanesh, S., Wang, L. & Feng, H.-Y., 2010. *Adaptive Assembly Process Planning and Control Using Function Blocks,* Ontario, Canada: s.n.

Langpop, 2011. *langpop.com.* [Online]
Available at: www.langpop.com
[Accessed 07 04 2012].

Lewis, R. W., 2001. *Modelling Control Systems Using IEC 61499.* First ed. London, United Kingdom: Institution of Engeineering and Technology(IET).

M., M., V., V., Xua X., W. S. & Z., A.-B., 2008. A novel open CNC architecture based on STEP-NC data model and IEC 61499. *Robotics and Computer-Integrated Manufacturing,* pp. 560-569.

Martinez Lastra, J. L., Godinho, L., Lobov, A. & Tuokko, R., 2005. An IEC 61499 Application Generator for Scan-Based Industrial Controllers. *IEEE International Conference on Industrial Informatics,* p. 6.

Netbeans, n.d. *netbeans.org.* [Online]
Available at: http://netbeans.org/community/releases/60/
[Accessed 14 03 2012].

Pai, K., 2002. *A simple way to read an XML file in Java.* [Online]
Available at: http://www.developerfusion.com/code/2064/a-simple-way-to-read-an-xml-file-in-java/
[Accessed 25 02 2012].

Rosen, 2002. *XML Reading Using DOM and SAX.* [Online]
Available at: http://www.codeproject.com/Articles/2741/XML-Reading-Using-DOM-and-SAX
[Accessed 15 03 2012].

Schneider, L., n.d. Java Programming Language Information.

Siemmens, n.d. *automation-course.com.* [Online]
Available at: www.automation-course.com
[Accessed 14 03 2012].

Tisserant, E., Bessard, L. & Sousa, M. J. R. d., 2007. An Open Source IEC 61131-3 Integrated Development Environment. *International Conference Proceedings,* pp. 183-187.

Transpaderne, T. & Vidal, E., 2012. *Wise ShopFloor decision support system,* Skövde: s.n.

Wang, L., Feng, H.-Y. & Cai, N., 2003. Architecture Design for Distributed Process Planning. *Journal of Manufacturing Systems,* XXII(2), pp. 99-115..

Wang, L., Givehchi, M., Schmidt, B. & Adamson, G., 2012. A Function Block Enabled Robotic Assembly Planning and Control System with Enhanced Adaptability.

Wang, L., Keshavarzmanesh, S. & Feng, H.-Y., 2010. *A function block based approach for increasing adaptability of assembly planning and control,* Skövde: s.n.

Wang, L., Song, Y. & Gao, Q., 2009. Designing function blocks for distributed process planning and adaptive control. *Engineering Applications of Artificial Intelligence,* p. 1127–1138.

Vogel, L., 2008. Java and XML – Tutorial. Volume 1.3.

Yandell, H., 2002. *Generating XML via java.* [Online]
Available at: http://www.techrepublic.com/article/generating-xml-via-java/1044810
[Accessed 21 02 2012].

Ziesemer, M. A., 2007. *XML generation in java.* [Online]
Available at: http://blogger.ziesemer.com/2007_06_01_archive.html
[Accessed 21 02 2012].

10. Appendix

All codes (Java functionality) about the bachelor final year project are available in the research computers of the Automation Group at University of Skövde.