Bachelor Degree Project

FB-ENVIRONMENT IN WISE-SHOP FLOOR

Algorithm parser and code generation

Bachelor Degree Project in Automation Engineering
30 ECTS
Spring term 2012

Aitor Arrieta Marcos

Supervisor: Bernard Schmidt
Examiner: Prof. Lihui Wang

This project is submitted by Aitor Arrieta 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

Aitor Arrieta Marcos

I. Executive Summary

IEC (International Electrotechnical Commission) is the authority that publishes different standards in the fields of electrical and electronics engineering, to be used internationally. In the area of manufacturing, it has demanded a new standard to fulfil better solutions of dynamic requirements. The IEC 61499 redacted by IEC offers interoperability, portability, configurability and distributed control applications for manufacturing processes. However, this standard is not a replacement of IEC 61131-3, one of the most used standards in industry; instead, it is a complement of it. The basic software units of IEC 61499 are named Function Blocks (FBs), which can be described as blocks that encapsulate functionality. By combining FBs together, it is possible to solve complex problems.

The objective of this project (in close cooperation with another project) is to develop a software environment in Java language. It follows the requirements of IEC 61499, and implement a Function Block designer and a runtime execution environment, as a part of an existing Wise-ShopFloor framework. The scope of this project covers:

- FB algorithm editor: Each FB has one or more algorithms, which can be defined in the algorithm editor using IEC 61131-3 or Java.
- FB serialization: Opening and saving the configuration of FBs in Java Class file is one of the tasks of this project. As soon as the configuration is saved, the Java code of FB can be generated. Java code is generated because compiled Java allows execution of FB. Saving in Java Class file permits portability, i.e. the saved configuration can be opened in any JVM system, and vice versa.
- Case study: A simulation of an assembly station using an ABB IRB 140 robot is studied and implemented using the runtime simulator of the Java platform, in which some basic FBs have been also created in a library.

This project also includes: (1) implementation of user interface and (2) FB serialization in XML.

It is anticipated that the developed environment will be able to save and open FBs configurations either in XML or in Java Class, following the specification of IEC 61499. It will allow portability and reusability. Because of the portability, the so-designed FBs can be validated using another FB environment such as FBDK (Function Block Development Kit).

II. Acknowledgements

Firstly, I would like thank my supervisor of the project, Bernard Schmidt for his guidance in the project, helping me orienting the project, giving advises, explaining all my doubts and explaining and giving help with the Java programming code, as well as for his work with the simulator of the program.

Secondly, I would also like to say thanks a lot to Mikel Anasagasti who has been running his project in close collaboration with me, for making a good project team, and for encouraging me in the most difficult moments of the project.

I also want to thank Markel Garzia and Tomas Alberto Rodriguez for helping me with Java questions when I needed help.

Of course, I would like to thank Ainhoa Goienetxea, who has taken all my problems as her personal problems in all my Erasmus stay, and also for making it possible for me to stay the second semester developing this project in Skövde; for this last reason I have to thank also to the Erasmus team of Mondragon Unibertsitatea, Edurne Agirre and Fernando Garramiola.

I also want to thank my colleagues studying in automation engineering, that I have met in this Erasmus, for making such a good studying team: Ceferino Arias, Alvaro Miranda, Victor Carretero, Karam Arriouach, Carlos Gil, Manuel Guzman, Raul Diaz, Juan Cana and the special one, that even if he is not student of automation engineering, he has taken some courses with us: Antonio Cervera.

I also want to thank the master student Kike Palomeque, for giving me advises for the project, subjects, as well as other type advises.

Finally, I would like to thank my parents and my brother, because without their encouragement and help it could not be possible to stay this year in Sweden.

III. Table of Contents

IV. Table of figures

V. List of tables

VI. Nomenclature

4DIAC	Framework for Distributed Industrial Automation and Control
CNC	Computer Numerically Controlled
EC	Execution Control
ECC	Execution Control Chart
FB	Function Block
FBD	Function Block Diagram
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IL	Instruction List
IPMCS	Industrial Process Measurement and Control Systems
JVM	Java Virtual Machine
LD	Ladder Diagram programming language
PID	Proportional-integral-derivative controller
PLC	Programmable Logic Controller
SFC	Sequential Function Chart
ST	Structured Text
TCP	Transmission Control Protocol (TCP)
XML	eXtensible Markup Language

1 Introduction

The International Electrotechnical Commission has developed a new International Standard, the IEC 61499. This bachelor degree project, in cooperation with another parallel bachelor degree project (Anasagasti, 2012), will have as a result a Web-Based Function Block programming environment, which will follow the requirements of the aforementioned IEC 61499 standard, allowing to make FB system configuration. The environment was previously implemented as a part of the existing Wise-ShopFloor framework in a research organisation of Canada, and the main changes that both bachelor degree projects will develop are focused on the user interface, the algorithm and Execution Control Chart editor, the ways of saving files that will allow portability and the implementation of some library exemplary of basic FB.

It is important to say that this project is a small part of a bigger project called Adaptive Decision Making for Wise-ShopFloor research group, and is joined together with the other automation bachelor project (Anasagasti, 2012) and other two design bachelor projects (Transpaderne & Vidal, 2012) and (Cervera, 2012), that could be implanted in Volvo Cars and Sandvic AB as an information presentation device that allows communication between the workers and the working environment. This project will cooperate also with another two bachelor degree automation students´ project (Díaz, 2012) (Arriouach, 2012) as well as with a master in industrial informatics student (Palomeque, 2012).

The remainder of the thesis will explain the main scopes of this project, which can be summarized as (1) the development of the FB algorithm and ECC editor, (2) the implementation of FB serialization for saving and opening Java Class Files, and (3) preparing library of exemplary basic FB of the FB programming environment mentioned before; These tasks have been performed in collaboration with the previously mentioned bachelor degree student´s project (Anasagasti, 2012) in order to share and compare the similarities of both projects that later the implementation of these will be joined being a single Web-Based environment. In addition, the first months have been used to make a research of Function Blocks and their functionality.

Java programming language has been preselected for the development of the environment, which is an object oriented language that is being very used by software developers, and the used program for the implementation has been Netbeans.

2 Project motivation

This project is a part for a bigger project developed in the University of Skövde named Wise-Shop Floor, a *"Web-based Integrated Sensor-driven e-Shop Floor"* (Wang, et al., 2003) (Wang, 2009). Adaptive decision making is a new concept that allows solving problems in a company that usually has a degree of uncertainty, and it is solved thanks to real-time information of resources, establishing a closed-loop real-time information flow between sensing, monitoring, process planning, and execution control, as it could be seen in the figure below:

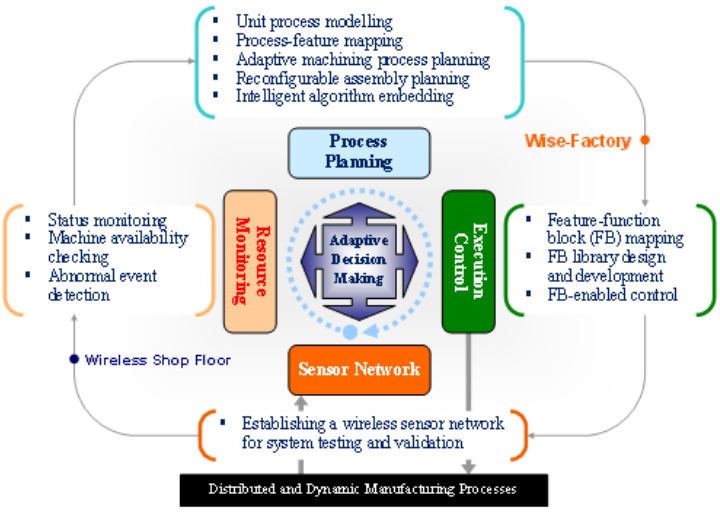


Figure 1: Diagram of the term Adaptive Decision Making (Provided by the University of Skövde in the project definition)

As explained before and as it can be seen in the previous figure, Adaptive Decision Making can be divided into four units: (1) Sensor Network, (2) Resource Monitoring, (3) Process Planning and (4) Execution Control. This project is going to be focused on a part of the Execution Control unit. For the Execution Control, a programming environment is needed, and in Adaptive Decision Making, the Function Blocks (FB), the basic functional software unit of the IEC 61499 is selected because it is able to give solution to complex problems offering the next advantages:

- Interoperability: Allows different units in a factory to work in collaboration.

- Portability: Allows using the same software in different programming environments. In the IEC 61499, this is possible thanks to saving and opening in XML or in Java class format.

•Configurability: it is defined as the ability of the devices and software to be configured from different software environments.

Apart from that, FBs are object oriented, so it takes its benefits which will be explained later.

3 Literature survey

3.1 International Electrotechnical Commission

The International Electrotechnical Commission (IEC) was founded in 1906, as a non-profit and non-governmental organization, in which the members are National Committees, also named member country, consisting of experts and delegates from industry, government bodies, associations and academia (IEC, 2012), and the members are divided into two levels: full members or associate members. Its central office is in Switzerland, however, different offices can be found in other four countries of the world (Singapore, United States, Brazil, and Australia).

The IEC is the world's leader for the elaboration of new standards for the electrical, electronics and related technologies, also called "electrotechnolgy". When taking a decision, every member country has a vote, so that the standards are consensus based (IEC, 2012).

The IEC has published different standards such as IEC 61850 (standard for design of electrical substation automation), IEC 61508 (standard that fulfils the safety integrity level on electrical, electronic or programmable electronic systems), IEC 61970 (standard for interfaces for energy management systems) or IEC 61131 (standard for PLCs) (IEC, 2012). The first three standards mentioned above are not relevant to this work whereas the IEC 61131 is relevant to the IEC 61499. The IEC 61499 is a new Standard developed by this commission that offers an open standard for distributed control and automation.

3.2 The IEC 61499

The standard IEC 61499 was developed by the International Electrotechnical Commission in order to provide better solutions to the requirements of the manufacturing and automation world (Lastra, et al., 2005). The first version of IEC 61499 was published at first in 2000, it was called IEC/PAS 61499-1, but after some technical improvements, a new version was published in 2005 replacing the previous version, and it was called IEC/PAS 61499 (IEC, 2005). This standard proposes an open architecture for distributed Industrial Process Measurement and Control Systems (IPMCS) (Lastra, et al., 2005), offering interoperability, portability, configurability and distribution control applications (Minhat, et al., 2009) (Doukas, et al., 2006).

The basic functional software unit of the IEC 61499 is the Function Blocks (FB) (Lastra, et al., 2005). A FB is a piece of software that has its own data, state transitions and internal algorithms (Lastra, et al., 2005) (Yoong, et al., 2009) that are able to modify or change data depending on the different situations. In (Wang & Yijun Song, 2009), a FB is described as an "*IEC standard for distributed*

industrial process measurement and control systems, particularly for PLC control". The FBs will provide easier program distribution, integrated visualization and improved real-time control characteristics (Minhat, et al., 2009). The execution model of an IEC 61499 FB is based on events that trigger algorithm's execution, named event driven execution control, adopting a different execution model of a PLC or different embedded controllers instead of using cyclic scan based execution model (Lastra, et al., 2005).

The IEC 61499 is divided into four parts:

IEC 61499-1, where the architecture of the standard is described.

IEC 61499-2, where the software tool requirements are described.

IEC 61499-3, where there are some tutorial information.

IEC 61499-4, where some rules for compliance profiles are explained.

3.2.1 Why use the IEC 61499?

As it was explained before, the IEC 61499 has some benefits that offers an open standard for distributed and automation industry. Interoperability, portability and configurability are the most important ones as well as other benefits.

Different embedded devices can cooperate together in order to fulfil different distributed application tasks thanks to **interoperability** (Zoitl, et al., 2009) which allows the exchange information between the devices (Sünder, et al., 2006). For that, communication standards are used (IEC 61131-5), based on communication protocols and buses as for example Modbus/Transmission Control Protocol (TCP) or ModBus (Zoitl, et al., 2009).

Thanks to the **portability** a saved program can be opened in another environment. Syntax of the IEC 61131-3 is the same for all the software tools. Standardization can offer exchanging components from a software environment to another, but many times, each software environment interprets the standard in one way or in another, being the same code as in other environment, but the result is not the same (Zoitl, et al., 2009). PLCopen has defined a XML format standard for exchange defined in IEC 61131 (Zoitl, et al., 2009). Thanks to this format of the PLCopen, it is possible to exchange configurations, resources, tasks, program instances, global variables for communication and the access to inputs and outputs. However, some vendors do not offer this format or they do it in another way (because of the license, etc...), so the configuration is not admitted in other

environments (Zoitl, et al., 2009). The IEC 61499 offers the possibility of saving and opening the configuration in XML and in Java class file format.

The developed environment will be able to open and save the configuration in both formats, offering portability in order to be used with other software tools. The XML for the developed environment is being discussed in (Anasagasti, 2012), whereas this project will be focusing on Java class file. XML (eXtensible Markup Language) is a language that human and machines are able to read and understand (Anasagasti, 2012). It is being more useful and it is becoming more popular because its simplicity on the structure and because it is easy to understand (Anasagasti, 2012). Another language that allows portability in the IEC 61499 is Java class file, which is going to be developed in this project as it was mentioned before. The java class file is a defined format for compiled Java and it can be executed in any Java Virtual Machine (JVM) (Venners, 1996), and it uses the ".class" extension. As soon as the configuration is saved the Java code for FB is generated, allowing execution of FB. Each java class file has the definition of a single class or an interface (Su, et al., 2009).

Configurability gives the ability to the devices and software to be configured from different software environments; Configuration ability includes: ability to be selected, assigning location, interconnected and parameterised (Sünder, et al., 2006).

The IEC 61499, as it was described before, offers interoperability, portability, and configurability, as well as it is compliant with **distributed industrial-process measurement and control systems** (IPMCSs), devices, its life cycle supports system will give high benefits to those who will use the standard, thanks to the next reasons (Lewis, 2001):

-It will reduce engineering costs; the model includes the control and processing of the information and the communications and interfaces between the hardware.

-The embedded firmware and software is one of the most expensive items in the control of hardware nowadays, but as the software and the firmware is common, the costs of the hardware should be reduced.

-The engineers will reduce the implementation time once they adopt the programming skills, thanks to the object oriented technology, and the simple programming way.

-The use of the standard will have a bigger reliability and maintainability over the system and these is thanks to the interoperability and portability, erasing patchware and glueware.

The use of FBs offers also some advantages to model device or resource management applications: a consistent model of all applications in the system, thanks to encapsulating and re-using all functions, also in management functions, as well the re-use of existing data-types.

The execution control of the IEC 61499 is an event driven model, depending on how it is implemented, if the execution could be real time; The IEC 61131-3 uses a Cyclic scan model, which is continuously reading the inputs and updating the outputs, not being highly effective; in the last decade, the automation industry has being investigating so solve this problem. Event driven execution model provides real time execution control, permitting better scheduling functions and providing a higher flexibility. Figure 2 depicts a diagram that describes the different processes of the cyclic scan mode.

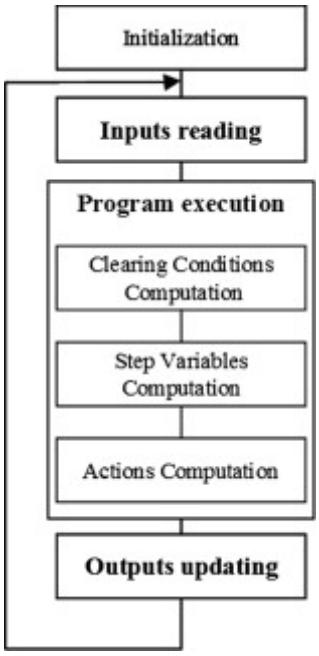


Figure 2: Flux diagram of scan based execution control (Machado, et al., 2011)

Another advantage of FBs is that they are object oriented, so that this programming environment will have all those benefits (Lewis, 2001):

-Objects reflect the real world: using the entities as objects can be more intuitive, for example, in an assembly line you will have different instruments as a conveyor, robot_1, robot_2, a testing device... and each of those instruments´ operations (moving, putting, inspecting) could be represented as objects.

-Objects are stable: objects can be defined as software units that usually don´t change, so that the user of the environment could use the same object classes more than once. For example, in the same assembly line mentioned before, the robot_1 could be in maintenance, but not the robot_2. An object used in robot_1 could be used in robot_2 not to stop the manufacturing process while the robot_1 is not working. That object could be already created so that it could be used to program another robot for that job.

-Objects reduce complexity: once that an object is created, the user of the object does not need to know how it works internally because an application could be created joining different objects together. If we continue with the same example as mentioned before, the assembly line could be used for product A and for product B, and as they are different products the application should be different (different working time, different velocity of the conveyor, different robot movements...).

However, FBs have not taken many concepts of object-oriented technology, such as inheritance, and it has been criticised for that, and in the near future, the IEC could develop an extensible version of FBs including this technology (Lewis, 2001).

3.2.2 Reference models

As the IEC 61499 is not only a programming language, it is also an open architecture and model for distributed systems, i.e. "*a set of devices interconnected by some networks to form a set of co-operating application at a physical level*" (Lewis, 2001). In the IEC 61499 (IEC, 2005) four different reference models are defined: (1) system model, (2) device model, (3) resource model and (4) application model. The next figure represents an idea of where each reference model is situated:

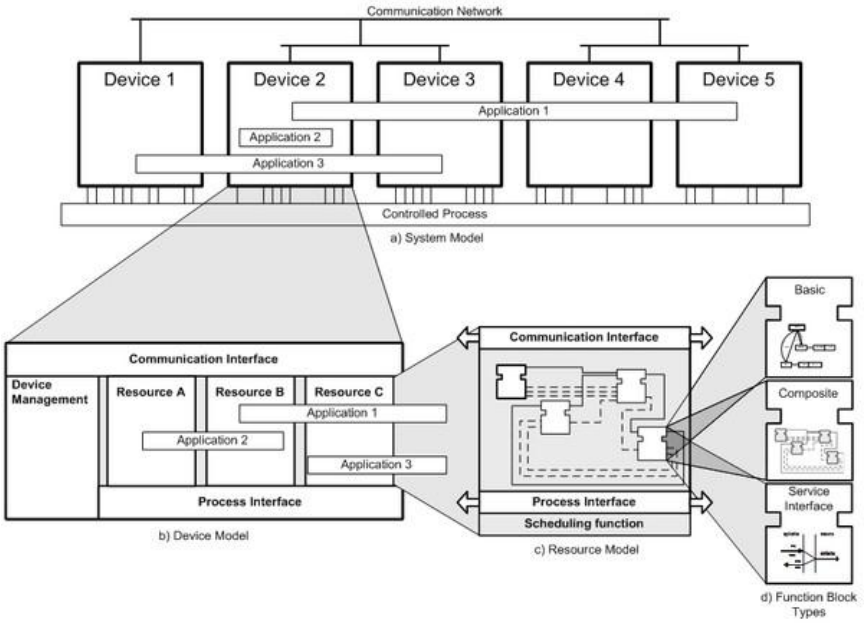


Figure 3: General view of the reference model of IEC 61499 (4DIAC, 2008)

The reference **system model** defines a relationship between communicating devices and applications (Lewis, 2001), interconnecting some devices and communicating them by a communication network consisting by some segments and links (used to connect to network) (IEC, 2005). Each device will have its own link and, and will be joined with the rest of Hardware with a segment. An application can reside in a single device or in several devices, i.e. distributed application as it is shown in the figure below, (Lewis, 2001) (IEC, 2005):

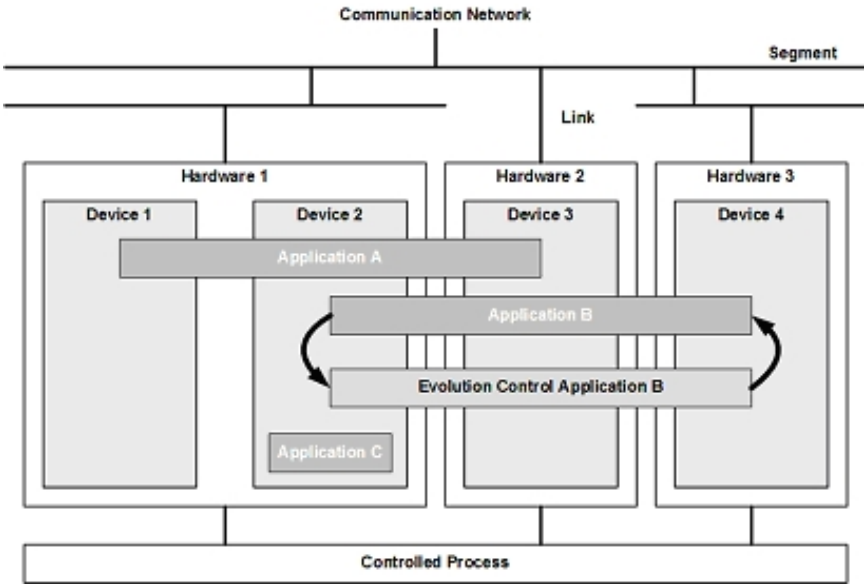


Figure 4: The reference system model (4DIAC, 2008)

An application would have one or more control loops consisting of (1) input sampling, (2) control processing and (3) output conversion, and each one is performed in a different device (IEC, 2005).

Inside the system model, it can be found the **Device Model**. A device is defined by (IEC, 2005) as an *"independent physical entity capable of performing one or more specified functions in a particular context and delimited by its interface"* and can support zero or more resources (a device that does not contain a resource is considered to have a function as a resource (IEC, 2005)), and will contain at least one interface, process interface or communication interface (IEC, 2005). *"A resource provides independent execution and control of networks of a Function Block (FB)"* (Lewis, 2001).

The process interface permits to read and change data from the physical process, i.e. from the Inputs and Outputs whereas the communication interfaces provide a communications exchanging information between resources in remote devices (IEC, 2005) (Lewis, 2001).

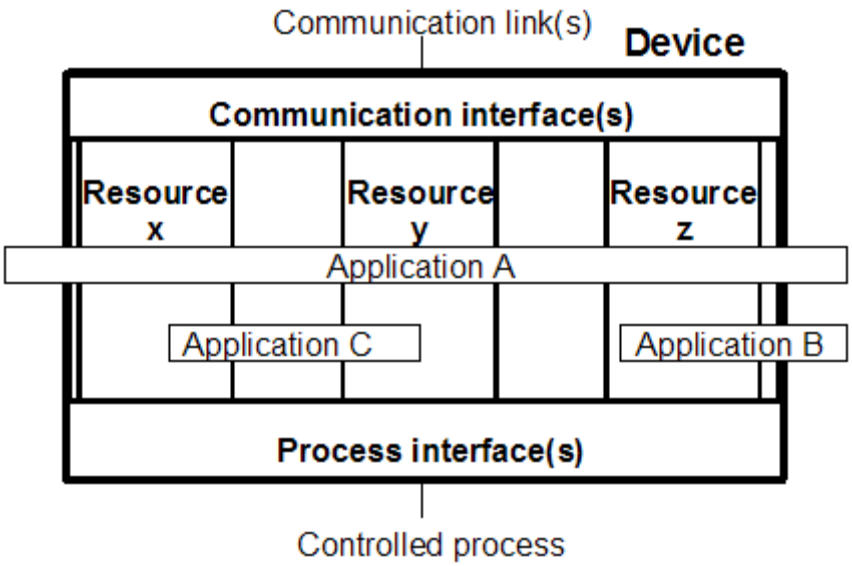


Figure 5: Reference Device Model (Christensen, 2012)

Under the device model, as Figure 3 shows, there is a **Resource model**. A resource is *"a functional unit that has an independent control of its operation and can provide services to applications, including the scheduling and execution of algorithms"* (IEC, 2005). The functions of a resource are supporting the execution of one or more function block (FB) application fragments (Lewis, 2001).

As it can be seen in Figure 6, a resource model is divided into (1) local applications, (2) process mapping, (3) communication mapping and (4) scheduling function (IEC, 2005).

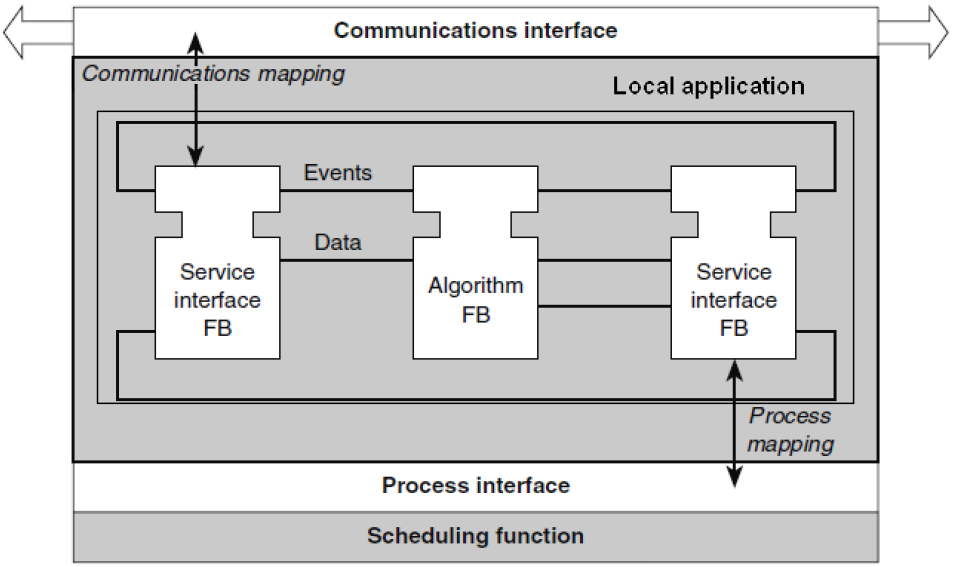


Figure 6: The reference resource model (Lewis, 2001)

-Local application: one or more local applications could be found in Resource model. The application in this part handles variables (inputs and outputs) and events (inputs and outputs) of FBs that perform the operations of the application (IEC, 2005).

-Process mapping: this part´s function is a data exchange between process interface(s) and application(s), and this mapping could be modelled thanks to the Service Interface FBs, who actuates as drivers (IEC, 2005).

-Communication mapping: part that performs a data exchange between communications interface(s) and the local application(s), which is allowed to make this data exchange thanks to the before mentioned Service Interface FB (IEC, 2005).

-Scheduling function: It is a function that allows the execution of, and data transfer between the FBs in the application depending on the timing and sequence requirements determined by (1) the occurrence of events, (2) FB interconnections and (3) scheduling information such as periods and priorities (IEC, 2005).

As explained before, inside the resource model there is a local application, and can be named as **Application model** and it can contain a FB network with FBs or subapplication instances linked together by data and event connections (IEC, 2005). As it was explained before, an application can be distributed in more than one resource and in the same or different devices (IEC, 2005).

The next level after the Application model is the **Function Block model**. This project will be focused on FBs, and the main characteristics and specification of FBs are explained in the next paragraphs.

3.2.3 Characteristics of Function Blocks

A FB is a block that encapsulates functionality. According to the IEC 61499 (IEC, 2005), a Function Block instance has the following characteristics: (1) type name and instance name, (2) event inputs, interface of a FB which may receive events from an event connection, and can affect the execution of one or more algorithms, (3) event outputs, interface of a FB which may issue events to an event connection, (4) data inputs, interface of a FB which may receive data from a data connection and corresponds to the input variables, (5) data outputs, interface of a FB which may supply data to a data connection and may correspond to output variables, (6) internal data mapped also as internal variables, (7) functional characteristics, which are divided into the Execution Control and the Internal Algorithms. Figure 7 shows how all these characteristic features can be mapped on a FB:

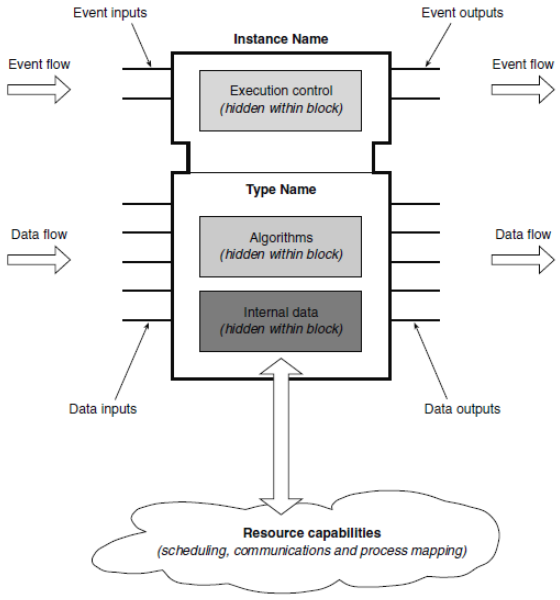


Figure 7: Characteristics of a Basic FB (Lewis, 2001)

Taking into account the IEC 61499 (IEC, 2005), the Function Block type specifications should include its (1) type name, (2) the number, names, type names and order of events inputs and events outputs, (3) the number, names, data type and order of input, output and internal variables.

Events and data can be differenced because events are *"an instantaneous occurrence that is significant to scheduling the execution of an algorithm"* and data is *"a reinterpretable representation*

of information in a formalised manner suitable for communication, interpretation or processing" (Lewis, 2001).

3.2.4 Function Blocks types

There are different types of FBs: (1) Basic Function Blocks, (2) Composite Function Blocks and (3) Service Interface Function Blocks. A Basic FB is the simplest FB, in which the executions of the algorithms are controlled by the Execution Control Chart (ECC). A Composite FB "*provides a way to encapsulate a network of FBs within another block"* (Yoong, et al., 2009). A Service Interface FB *"serves as a device driver to bind the FB application to a specific hardware target"* (Yoong, et al., 2009).

The **Basic FB** (Figure 8) type cannot be decomposed into other FB (IEC, 2005). The Basic FB uses an Execution Control Chart (ECC), which will be explained later, in order to control algorithms execution. These algorithms can be written in different programming languages such as Java, C, Delpi… or the IEC 61131-3 programming languages (Ladder Diagram, Structured Text and Instruction List (Sousa, 2008)) so that the algorithms could be executed; however the most used language is the Structured Text. The executions of the algorithms are executed by the Execution Control Chart (ECC). A Basic FB cannot be distributed because it can be run only on a single resource (Lewis, 2001).

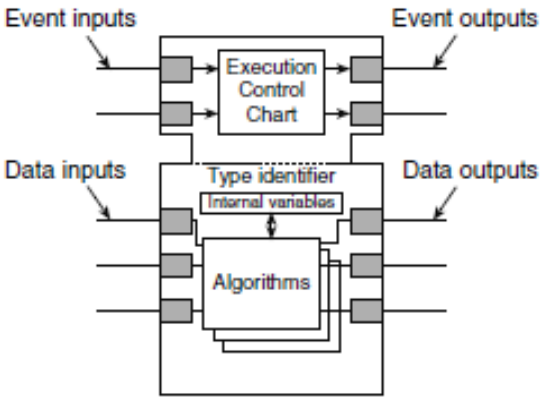


Figure 8: Basic Function Block (Lewis, 2001)

The FBs can be connected between them, establishing FB networks. The behaviour of the network is being determined by the event connections, also named instances and every single block (Vyatkin, 2011). These networks may be encapsulated into a single block called **Composite FB** (Figure 9) and these FBs can be used in other networks; these networks can also be encapsulated into other Composite FB developing different hierarchies (Vyatkin, 2011).

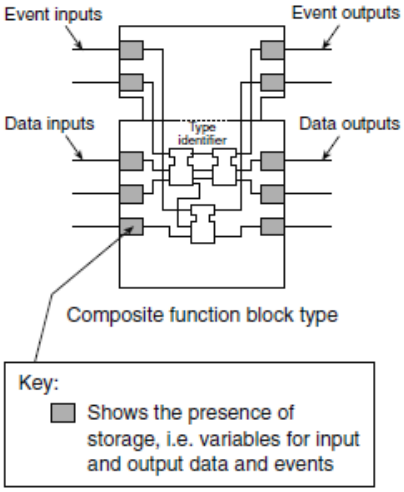


Figure 9: Composite Function Block (Lewis, 2001)

The Composite FBs are similar to the basic FBs: both have event inputs, event outputs, data inputs and data outputs (Vyatkin, 2011). However, the basic FBs do not have internal variables nor ECCs, but another component can take this role as it can be seen in the Figure 10 (Vyatkin, 2011).

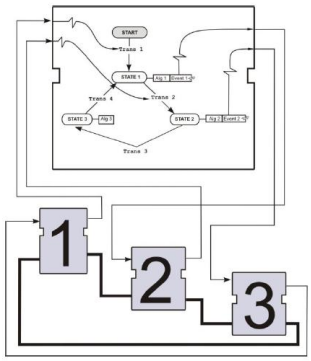


Figure 10: How a component can play the ECC´s role in a Composite Function Block (Vyatkin, 2011)

The **Service Interface FB** (Figure 11), as the name says, it is an interface Function Block (FB), which allows interfering between the FB domain and external services (Lewis, 2001) such as hardware target, or remote device (PLC, microcontroller...). Explained in an easy way, they can be described as reader and writers.

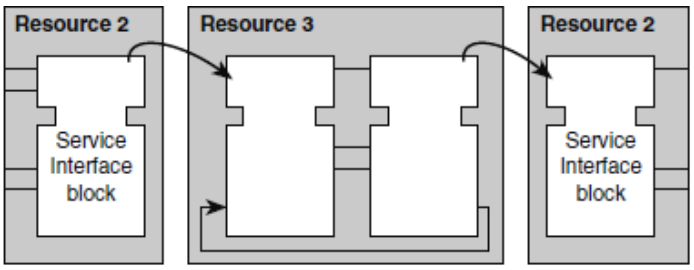


Figure 11: Example of an application of the Service Interface FB (Lewis, 2001)

A Basic FB is able to represent a small task, having a similar behaviour as an electronic device or circuit (Minhat, et al., 2009), and can solve simple problems, but joining different FBs (Composite FB), a more complex problem can be solved (Lastra, et al., 2005).

Subapplication (Figure 12) is another category of Function Block (FB), which is similar to Composite FB, but they can be distributed to run on more than one resource (Lewis, 2001). It is constructed from networks of Basic FBs and Composites FB that also could contain subapplication of lower level inside (Lewis, 2001). This type of Block can be distributable (Lewis, 2001).

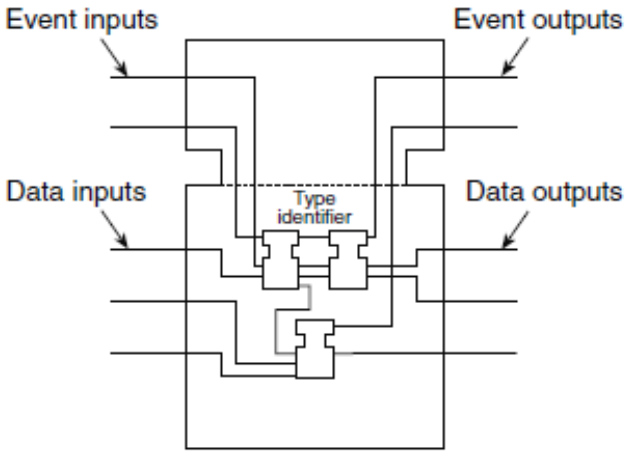


Figure 12: Subapplication FB (Lewis, 2001)

3.2.5 Execution model

The execution model of a function block (FB) describes eight phases that defines the behaviour of a Basic FB (Lastra, et al., 2005) (Lewis, 2001) when it is running. In this case, a scheduling function is used in order to ensure that each phase is executed in a correct order and at the correct priority (Lastra, et al., 2005) (Lewis, 2001). The phases are the next ones (Lastra, et al., 2005) (Lewis, 2001):

1. Data inputs of the function blocks have a stable value.

2. An event which is associated with a data input is arrived to the event input of the FB.
3. The execution controller indicates to the scheduling function that it has a signal and it is ready to execute an algorithm.
4. After a time, the scheduling function executes the algorithm.
5. Algorithm produces an output value after processing the input values and if there are internal variables, which also can be changed.
6. The internal algorithm sends a signal to the scheduling function notifying that the execution is finished.
7. The scheduling function invokes the FB's Execution Control, informing that the algorithm has finished its execution.
8. The Execution Control creates an output event in the FB's output event interface according to the execution of the internal algorithm.

The steps are graphically described in the next figure:

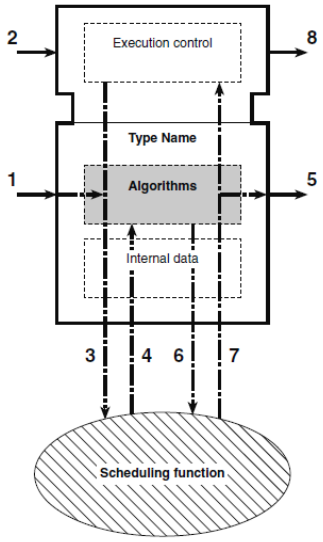


Figure 13: Phases of a basic FB (Lewis, 2001)

Each phase needs an execution time that usually is not perceptive for the human, but that could be important in some applications (Lewis, 2001). The IEC 61499 (IEC, 2005) defines the next durations:

Tsetup = T2-T1 (Time between the phase 2 and 1)

Tstart = T4-T2 (Time between the phase 4 and 2)

Talgorithm = T6-T4 (Time between the phase 6 and 4)

Tfinish = T8-T6 (Time between the phase 8 and 6)

Figure 14 (Lewis, 2001) shows the relationship between the different timing points, and when they are changed, depending on the different inputs and outputs:

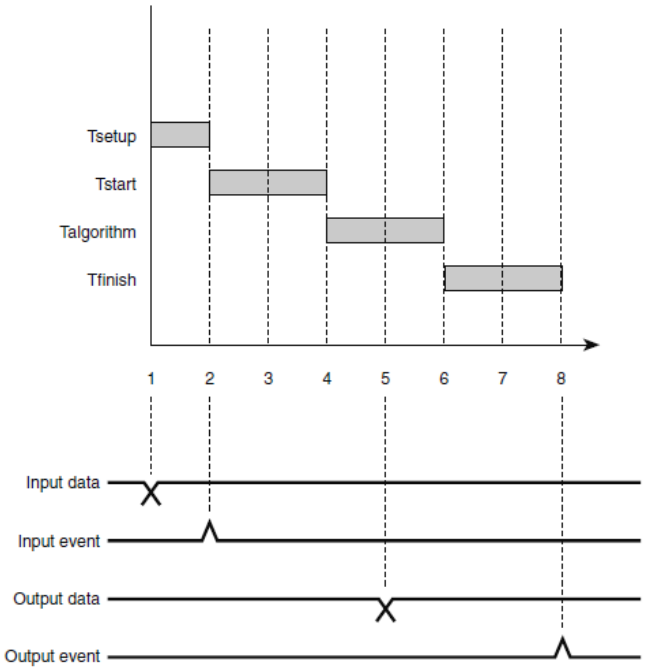


Figure 14: Relationship between the different timing points (Lewis, 2001)

The execution model explained before can be used just in Basic FB. In Sünder et al. (Sünder, et al., 2007), an execution model for Composite FBs and Subapplication is explained.

3.2.6 Internal behaviour

The internal behaviour of a Basic Function Block (FB) takes into account the algorithm bodies and the algorithm execution control (Lewis, 2001). A basic block usually contains one or more algorithm (Lewis, 2001); however there are cases in which the block just uses the ECC without any algorithm. Each algorithm is invoked by the scheduling function that depends on an event input (Lewis, 2001). As it is explained in (Lewis, 2001), one of the most important parts of the behaviour of a FB consists of the relation between the event that invokes the algorithms and the algorithms, and this is joined thanks to a concept named Execution Control Chart (ECC), its configuration of which was developed in this project.

The **Execution Control Chart (ECC)** is the graphical or textual representation between the relation that the Event Inputs, Event Outputs and Function Block's (FB) algorithms (IEC, 2005), thanks to some Execution Control States, transitions and actions, which will be explained in the next paragraphs. An example of an ECC is presented in Figure 15.

The IEC 61499 (IEC, 2005) defines the following characteristics for the ECC: (1) it resides in the upper portion of the FB, (2) it will have one Execution Control (EC) initial state, which is represented graphically with a double outlined shape, (3) there will be one or more EC states, which are represented graphically with a single outlined shape, and this states could have one or more associated EC actions, and (4) the ECC can use but not modify variables declared in the FB type specification.

Taking all these characteristics into account, it can be said that an ECC is divided into EC states, EC transitions and EC actions.

EC states: An EC state is a part of the ECC which is defined by the IEC 61499 (IEC, 2005) as the *"situation in which the behaviour of a basic FB with respect to its variables is determined by the algorithm associated with a specified set of execution control actions."* There are two types of EC States: (1) Initial EC state and (2) Common EC state. The Initial EC state is the state in which the ECC starts when it is executed in the beginning, and after an EC transition, the state will be changed to a Common EC state, but also can be returned to the Initial EC state. An EC state can have one or more EC action.

EC transitions: An EC transition is a Boolean expression, part of the ECC that allows "jumping" between an EC State to another. This Boolean expression can be used with Event Input variable, input variable output variable or internal variable (IEC, 2005).

EC action: As defined in (IEC, 2005), an EC action is an *"element associated to EC state that identifies algorithm(s) to be executed and event(s) to be issued on completion of execution of the algorithm"*. The differences between EC states and EC actions can be seen in Figure 16.

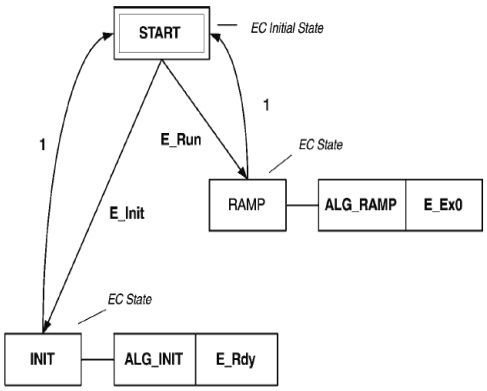


Figure 15: Example of an ECC (Lewis, 2001)

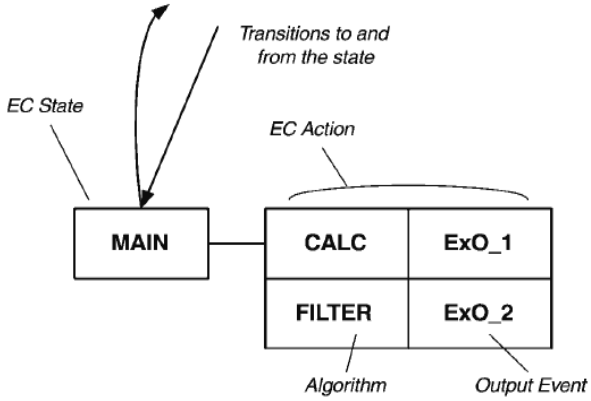


Figure 16: Example of an ECC's EC State (Lewis, 2001)

The ECC will execute **Algorithms**, which is defined by (IEC, 2005) as *"finite set of well-defined rules for the solution of a problem in a finite number of operations"*. The algorithms are invoked following some rules. When a FB is not executing any algorithm and an Event Input occurs, EC transitions in the ECC are evaluated following the active EC state. If there is no a true condition no action will be performed, and the FB will wait until another Event Input is performed. If there is a true condition, EC action will be performed, and an algorithm could be invoked after a request to the scheduling function that will schedule the execution of the algorithm´s operation (IEC, 2005). Once the actions are completed, EC transitions will be evaluated again. All this process can be better seen in the following state diagram:

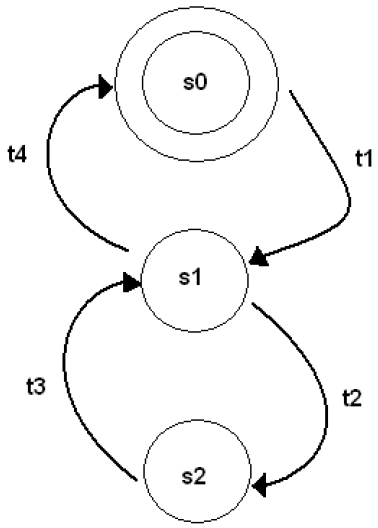


Figure 17: State Diagram of the execution of algorithms (IEC, 2005)

STATE	OPERATIONS
s0	Idle
s1	Evaluate transitions
s2	Perform actions

Table 1: States of the State Diagram of the execution of algorithms (IEC, 2005)

TRANSITION	CONDITION
t1	Invoke ECC
t2	There are transitions
t3	Actions are completed
t4	There is no transition

Table 2: Transitions and conditions of the State Diagram of the execution of algorithms (IEC, 2005)

The standard does not specify any programming language that the algorithm should be written, but the most common ones are the IEC 61131-3 programming languages such as Structured Text (ST), and also higher level languages such as Java, C or Delpi. However, some FB programmer could use other languages such as Ladder (LD) as it is shown in Figure 18, because they are used to program in another language.

The algorithms should fulfil the next characteristics described in (Lewis, 2001):

-The input and output variables should be assigned precisely and without ambiguity to the variables of the algorithms.

-The algorithm should be encapsulated, i.e. it can just read and write variables inside the FB´s body.

-The execution time should be short in relation to the velocity of the arrival of events that executes their execution.

-The initial state should be well defined.

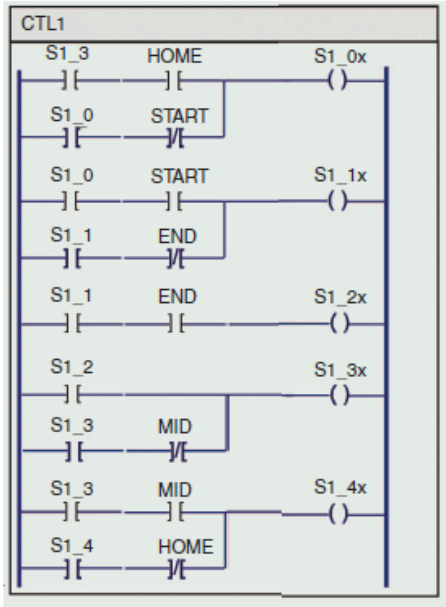


Figure 18: Example of an algorithm programmed in LD language (Vyatkin, 2009)

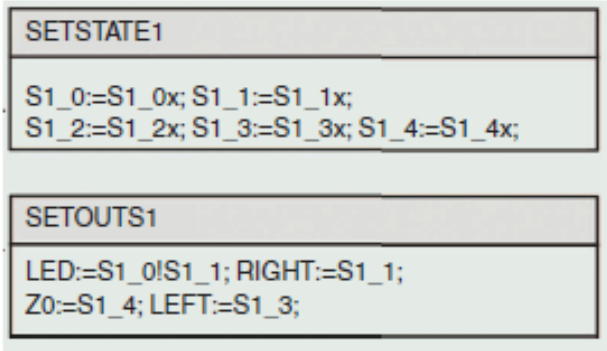


Figure 19: Example of an algorithm programmed in ST language (Vyatkin, 2009)

3.2.7 IEC 61131-3 vs.- IEC 61499

In (Wenger, et al., 2009), some differences of Execution Sequence are explained between the IEC 61499 and IEC 61131-3. The first difference is that whereas the IEC 61131-3 is based on a cyclic execution order, the IEC 61499 is based on an event triggered execution concept, i.e. it is based on events, supporting asynchronous execution (Strasser, et al., 2006). The second difference is that in IEC 61131-3 a resource represents a signal processing function, but in IEC 61499 a resource is responsible for the independent execution of FBs.

In (Wenger, et al., 2009) differences between both IEC 61131-3 and IEC 61499 standard libraries are also mentioned, focusing on semantic correct transformation from IEC 61131-3 to IEC 61499 where the differences are explained in (Wenger, et al., 2009), and can be seen in Figure 20 bellow, in which the FBs according to the IEC 61311-3 are translated to the FBs of the IEC 61499, and some mistakes of these are denoted:

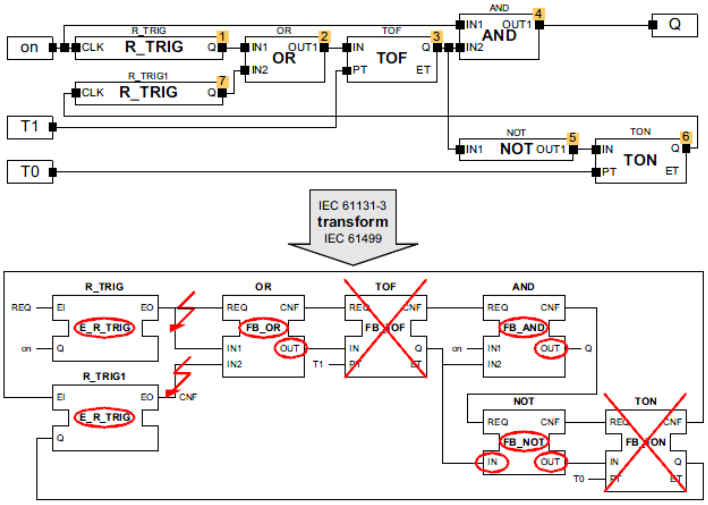


Figure 20: Semantic correct transformation from IEC 61131-3 to IEC 61499 (Wenger, et al., 2009)

In (Zoitl, et al., 2009), the main characteristic´s differences of both standards are given in a table (Table 3).

	IEC 61131-3	IEC 61499
Datatypes	Defined	Adopted from IEC 61131-3
Programming languages	IL, LD, FBD, ST, SFC	No specific language defined, the use of the IEC 61131-3 languages is recommended
Structuring means	SFC, FBD, ST	FB network
Global variables	On different levels	Not possible
I/O access	Direct addressed variables	Encapsulated in service interface FBs
Engineering approach	Application centred, in practice mostly device centred	Application centred
Execution concept	Mainly cyclic	Event driven
Dynamic reconfiguration	Not provided	Interface and instruction set, not for data
Distribution concept	Not provided	Arbitrary distribution of applications

Table 3: Differences of the main characteristics between IEC 61131-3 and IEC 61499 (Zoitl, et al., 2009)

In (Lewis, 2001) there is a graphic that shows how the technology has been advancing since 1950s, and compares each technology with the amount of functionality of each. In that graphic both standards, IEC 61131-3 and IEC 61499 are situated in a term of a square that takes a surface in the graphic comprising advancing technology and functionality. Taking this graphic into account it is possible to see that both standards take together a piece of surface on the graphic and it refers to the Function Distribution taking as reference technology the Industrial Communications. The difference between them is that the IEC 61131-3 is in a lower level in term of functionality comprising the digital devices as functionality and taking as technology the Micro-processor, whereas the IEC 61499 takes as technology younger ones that the IEC 61131-3 does not take, the Data modelling and also the before mentioned object oriented technology, taking as functionality the Tool Integration, and part of Defence Independent Functionality. It is possible to see it in the graphic of Figure 21:

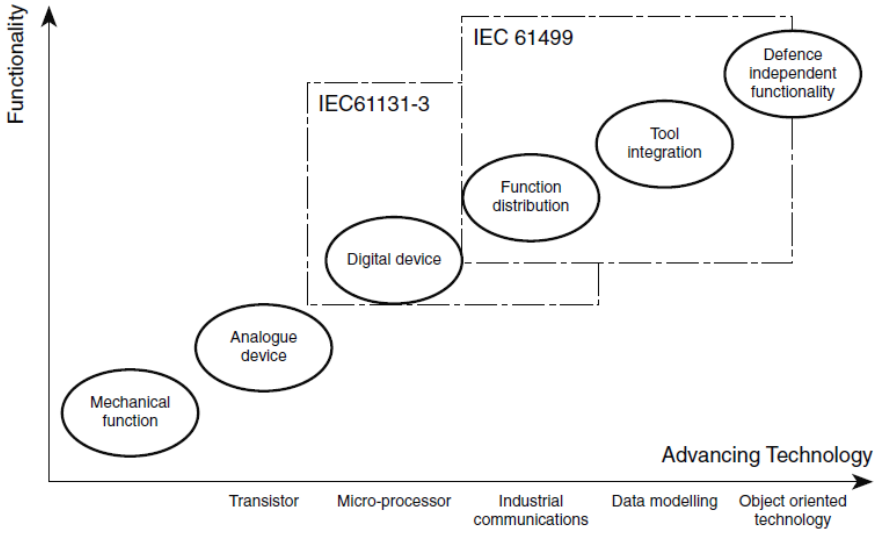


Figure 21: Graphic of the difference between IEC 61131-3 and IEC 61499 focusing on functionality and technology type (Lewis, 2001)

3.2.8 Event driven execution control

As it was mentioned before, one of the main differences between IEC 61499 and IEC 61131-3 is the execution model control used: IEC 61131-3 is scan based, i.e. it uses cyclic scan whereas the IEC 61499 uses event driven model execution control. The executions of the algorithms are done just when at least one event is triggered, and only the necessary algorithms are executed (Sünder, et al., 2006), because each task has to be associated with an event (Lastra, et al., 2005). Event driven is a priority based execution control (Lastra, et al., 2005), i.e. some tasks will be more important than

other so that they will have a bigger priority. Thank to this, more than one algorithm are not executed together.

This execution control model is done in real time, saving computing power making it more effective and reduces the computing power (Gerber, et al., 2008), whereas the cyclic scan is continuously reading inputs, executing the program and writing outputs.

3.2.9 Applications

The applications of the IEC 61499 are oriented to distributed processes in automated environments that could be executed in real time. The FB model is quite new with respect to the IEC 61131-3, so it is still growing. However, some projects that use the IEC 61499 have been found: in (Doukas, et al., 2006) a PID-based control application is examined for robotic arms; there are some research articles that apply this technology to CNC machines such as (Minhat, et al., 2009) and (Xu, et al., 2006); moreover, there are two projects that have been developed in collaboration with this (Díaz, 2012) (Palomeque, 2012) in which a CNC-robot is controlled with FB technology, the overview of these projects can be seen in Figure 22. In (Arriouach, 2012), a Scara robot is being controlled using the FB technology.

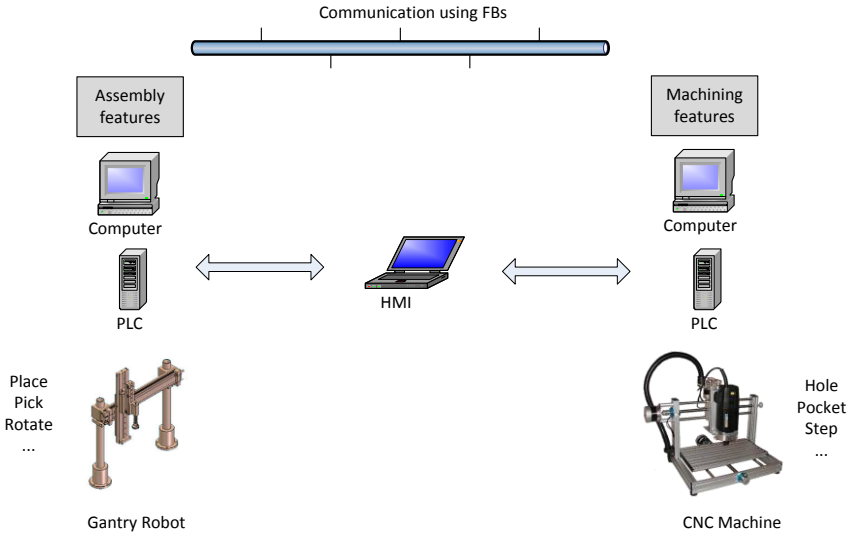


Figure 22: Project overview of (Palomeque, 2012)

3.3 Web based environments

Automation environments for multiple purposes and the use of these remotely are increasing in industry, and the solution for this is on Internet, permitting that the software tools to habit on the network through web based environments (Schwab, et al., 2005). More than one user could share information (Wang & Yijun Song, 2009), working from any part of the world.

The developed environment will be web based thanks to Java Virtual Machine (JVM), offering the next advantages: the updates are done on a "central location" so that when a user is connected, he does not have to download and install anything, saving memory in the hard disk because the access is direct, furthermore, the requirements of the RAM memory are quite low. Another advantage is real time data availability; the data that is generated can be seen by other users in real time, without the need of making phone calls or sending the information in another way. Data is safer, the hardware of a computer can have failure easily, but the data that is on internet is safer thanks to the different backups that the servers are used to doing. However, one of the most advantages that a web-based environment can offer is that you can access from any place where internet connection is available.

3.4 Saving methods

Portability is one of the main advantages that the IEC 61499 standard offers. The configurations can be saved in different methods and these can be opened in any FB environment that allows opening if the file is compatible. In (Anasagasti, 2012) the design and implementation of opening and saving files in XML is explained for the environment developed in collaboration with this project. This project will implement for the previously mentioned Web-Based environment a Java class saving method. These different methods are generated code into different languages that represent a FB configuration.

Some research articles show different files that the IEC 61499 can generate. In (Yoong, et al., 2009) it explains how to generate C code, whereas (Yoong, et al., 2009) focuses on how to generate Esterel code.

4 Methodology

As it was mentioned in the report, the scope of this project covers: (1) development of the FB algorithm editor, (2) FB serialization for saving and opening Java Class Files and (3) preparing library of exemplary basic FBs. The first step for developing the project correctly has been to make a **research about FBs** to familiarize with the performance of these ones with respect to the IEC 61499 standard. This has been important to know what it has to be developed and have an idea of the final result. To reach the goal, different information has been found in articles and books, sources that ensure a big security about the information that is written; however, internet has also been used to familiarize with basic concepts.

In collaboration with projects (Díaz, 2012) and (Palomeque, 2012), a small research of different software tools has been done to know how other environments look like. In (Díaz, 2012) and (Palomeque, 2012), a commercial software named nxtControl is used to develop the projects; another software tools such as 4DIAK or FBDK have been studied also.

There are different **programming languages** to develop software tools, such as C, C++ or Java. The developed environment must be a web-based environment and Java programming language is preselected because the previous version of the environment was started in Java programming language in Canada (Wang, et al., 2003). However, Java has different advantages compared with other programming languages; Java can be divided into two parts: Java programming language, and Java Virtual Machine (JVM). JVM is available for all commercial operating systems for computers (Windows XP, Windows vista, MAC, linux...), and it can be installed for free; once a developed program in Java language has been implemented, it is possible to run in any computer with any operating system if the JVM is installed, without changing any code line, whereas a program in C or C++ can be run just in the developed operating system.

The environment in which the code of this project is developed is named **Netbeans**, and the version that is used is the “IDE 7.1”. Netbeans is an open source IDE developed in Java programming language, and can be used to develop any type of applications (Netbeans, 2012). Different programming languages may be used in Netbeans, Java, C/C++, Ruby, Groovy, PHP (Anasagasti, 2012)...

One of the advantages of Java is that classes can be classified by **hierarchy**; each section of the environment (such as the interface, declaration, ECC editor) may have one or more classes. In (IEC, 2005), it shows an example of how the classes of the different declarations of a FB environment

could be classified hierarchically. However, that design has not been followed in the developed environment, because not all the classes shown in (IEC, 2005) are needed to fulfil the requirements of the project.

The developed environment may be able to save and open files either in XML or in Java class, allowing portability with other FB environments. The implementation of opening and saving in XML file is developed in (Anasagasti, 2012), whereas this project will be in charge of opening and saving in **Java class**, so that the saved configuration could be run in any JVM. In (Yoong, et al., 2009), an algorithm is developed to generate C code for basic and composite FB (Figures 23 and 24 respectively). In this project something similar could be developed, because C programming language's syntax is similar to Java programming language.

```
1  procedure GenerateBFB(fb)
2      S := set of all states in fb;
3      IE := set of all input events in fb;
4      OE := set of all output events in fb;
5      ID := set of all internal input data in fb;
6      ID' := set of all interface input data in fb;
7      OD := set of all internal output data in fb;
8      OD' := set of all interface output data in fb;
9      generate code to clear all output events in OE;
10     foreach ie ∈ IE, associated with id ∈ ID do
11         generate code to update id with id' whenever ie occurs;
12     end
13     foreach s ∈ S do
14         generate new case for s in switch-statement;
15         foreach action, a, of s do
16             if a has algorithm, alg then
17                 generate call to function[alg];
18             end
19             if a has oe ∈ OE then
20                 generate code to set oe;
21             end
22         end
23         foreach transition condition, t, of s do
24             if t leads to next state n ∈ S then
25                 generate code to test for t and assign next state
                   to n;
26             end
27         end
28     end
29     generate code to clear all input events in IE;
30     foreach oe ∈ OE associated with od ∈ OD do
31         generate code to update od' with od whenever oe occurs;
32     end
33 end procedure
```

Figure 23: Algorithm to generate c code for a Basic FB (Yoong, et al., 2009)

```
1  procedure GenerateCFB(fb)
2      P := set of all input ports in the network of fb;
3      foreach p ∈ P do
4          I := p.connectionSet();
5          if p.type = EVENT then
6              foreach i ∈ I do
7                  generate code to assign p to event, i;
8              end
9          else if p.type = DATA then
10             generate code to assign p to data, i ∈ I;
11         end
12     end
13     foreach component block, b, in the network of fb do
14         make call to execute b;
15     end
16     Q := set of output ports at the interface of fb;
17     foreach q ∈ Q do
18         O := q.connectionSet();
19         if q.type = EVENT then
20             foreach o ∈ O do
21                 generate code to assign q to output event, o;
22             end
23         else if q.type = DATA then
24             generate code to assign q to output data, o ∈ O;
25         end
26     end
27 end procedure
```

Figure 24: Algorithm to generate c code for a Composite FB (Yoong, et al., 2009)

There are different parameters that have to be taken into account when translating a FB into a Java class file. Firstly, the parameters to take into account are the set of all states, input and output

events, internal input data, interface input data, internal output data and interface output data in FB. Secondly, the ECC is translated into Java code, taking into account the transitions (that are represented in the Java code with "if" and "if else" statements) and the EC States. In Figure 25, a translation of an ECC of a cruise control FB is translated into C code, but as mentioned before, the syntax of C programming language is the same of Java programming language.

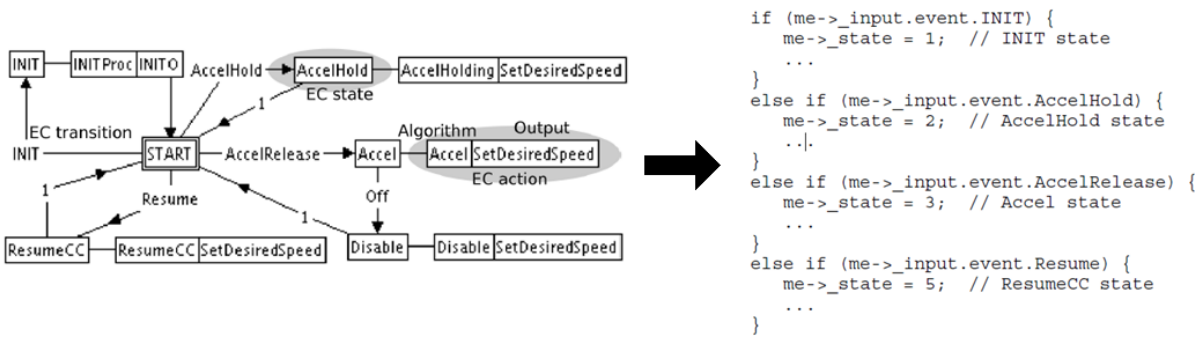


Figure 25: Translation of an ECC into c code (Yoong, et al., 2009)

In (Yoong, et al., 2009), some pseudo-code is explained, focused basically on the ECC to generate Esterel language and different processes are explained such as generating different types of transitions, handling unstructured transitions in an ECC and generating a "run" signal.

In this project, all the data added in the user´s interface is stored in a class named "FeatureData". This data is read by the program developed in this project so that the FBs can be drawn. When the Java code is to be generated, this data is also taken from the "FeatureData" class. Figure 26 depicts a diagram of how the data is taken and then used. Apart from the Java class, the FB environment is able to save the files in XML, part implemented in (Anasagasti, 2012).

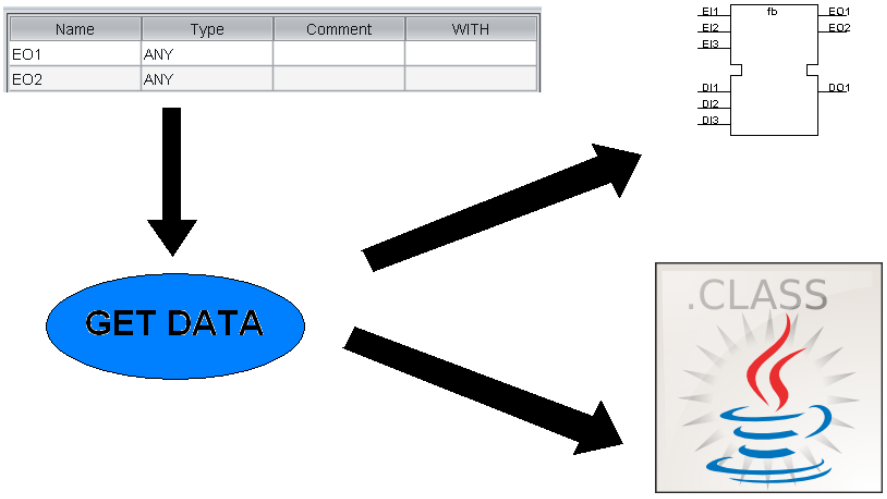


Figure 26: Diagram of how data is taken and then used

After generating the Java class, it should be compiled; this Java class is used in the environment for the runtime, in which the FB networks are simulated, and executions are visualized thanks to compiled Java that allows execution of FBs. This is better explained in the next section.

5 Implementation

5.1 Algorithm parser

As it was mentioned before, algorithms parser was one of the main tasks to be developed in this project. This task is divided in two: the implementation of the **Execution Control Chart**, and the **algorithm editor**. The development of the ECC editor has been carried out firstly; the ECC part of a FB and its main function is to invoke the algorithms. The ECC should be able to add or delete EC states, EC actions and inside the EC actions the algorithms or events output to be executed. Apart from that, it should also be able to join EC states with some transitions, linking them with an arrow. This is implemented thanks to hierarchy between Java classes, so that each class could have some tasks in the program.

5.1.1 ECC editor interface

As mentioned before an **ECC editor** was developed in the environment. Different steps have been done; firstly, the ECC editor was developed as a part of the FB platform, but later as a part of the Basic FBEditor. In Figure 27, it shows the previous hierarchy of the classes for the FB interface´s and in Figure 28, the result of the previous implementation of the FB-environment and how the different editors are illustrated on the left side of the program. In the ECC editor it is possible to work with two tools: the selection tool and the connection tool. The selection tool will be able to add or delete the EC states as well as move them for a better organization. With this selection tool, EC actions can be also modified, adding or deleting algorithms and event outputs. The tool also will be able to select and modify the connections between the EC states. The class that represents the selection tool of the ECC editor is named “ECCSelectionTool.java”. The connection tool is the tool that will connect the EC states with an arrow, representing transitions between states, and the class that will represent this tool is named “ECConnectionTool.java”.

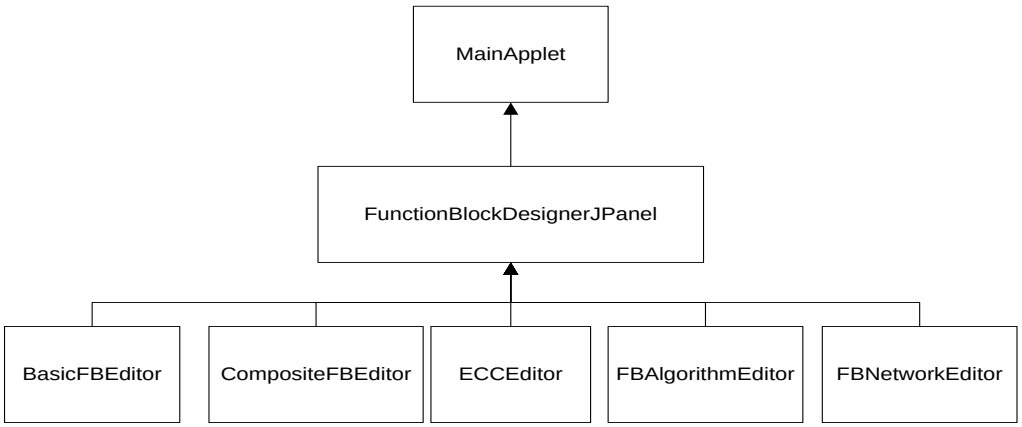


Figure 27: Class hierarchy of the FB-environment´s interface

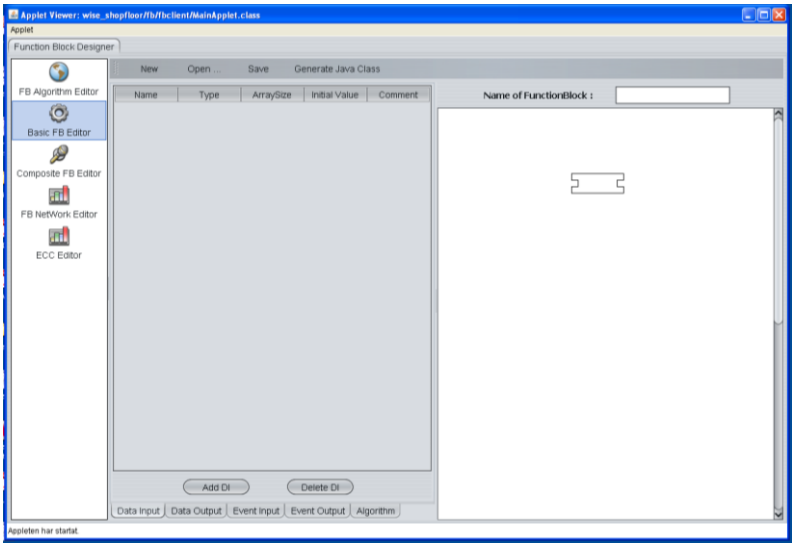


Figure 28: Interface´s overview of the FB-environment

There will be another class that represents the EC state´s blocks figures, named "ECCBlockFigure.java". The methods of this class will have different tasks. The method panel has two inputs (x and y), that represents the coordinates of the mouse when working with the selection tool; this method will show one menu or another depending on where the right click has been done. The method "createConnectors" creates seventeen connecting points in the upside and downside of the EC state when the mouse is inside the EC state with the connection tool; for that, the method "findConnectors" has two inputs (x and y), and its function is to find a connector depending the mouse´s position. Figure 29 illustrates the interface of the previous implementation of the ECC editor. However, Figure 29 is the result of some of the classes and methods described before and it is not a final result.

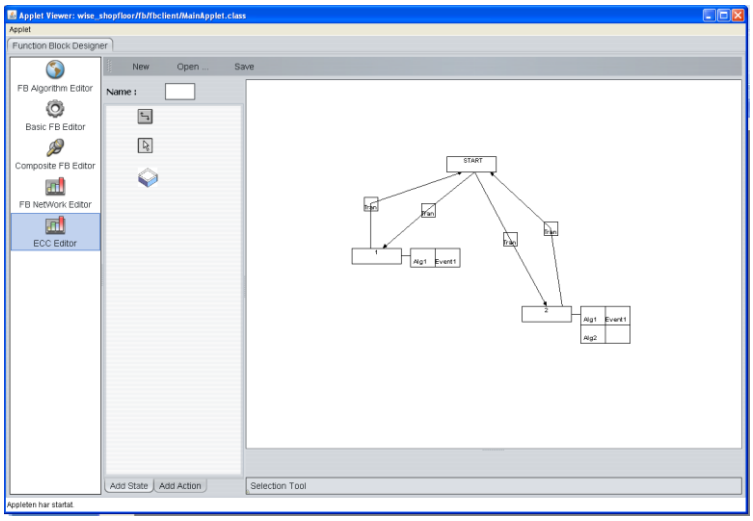


Figure 29: Interface of the previous version of the ECC editor

In addition, once the ECC editor was implemented, it was decided to implement it as a part of the Basic FB Editor, so that when designing a FB, the ECC of that FB could be implemented inside that FB, making the user´s interface more intuitive. Another reason to carry out these changes was that the Basic FB editor needs its own ECC editor, so that each Basic FB could have its ECC with the list of algorithms to be triggered and the list of Events Outputs to be executed. Apart from that, it was decided to erase the composite FB Editor, as well as the FB Algorithm Editor (the latter was implemented inside the Basic FB Editor). Figure 30 depicts the new interface´s class hierarchy, whereas Figure 31 shows the ECC Editor´s interface class hierarchy.

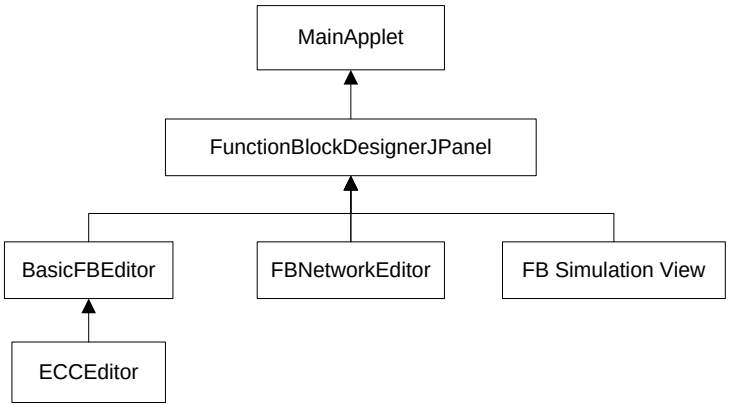


Figure 30: Redesign of the interface´s class hierarchy

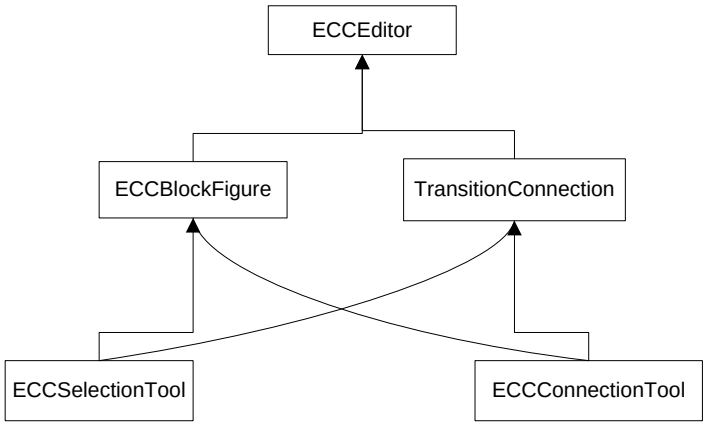


Figure 31: ECC´s interface class hierarchy

5.1.2 ECC data administration

Some classes have been created for the ECC´s data administration, with different vectors containing the data of the different parts of the ECC. For the design of the hierarchy of these classes, the structure of the ECC has been taken into account: An ECC will have different EC States, and an EC State will have different EC Actions, where the actions may have algorithms or event outputs. Apart

from that, the ECStates will be joined by ECTransitions. Being the structure in that way, the design of the hierarchy has been implemented as shown in Figure 32. However, the class that is on the top of the hierarchy is named “FeatureData.java”, this class was created before and contained the data of the hole Function Block editor, but as an ECC editor was added to the environment, this class will contain also the data of this editor. All this data is used because it is needed later to generate the XML and Java class files. Figure 33 represents the result of the ECC editor.

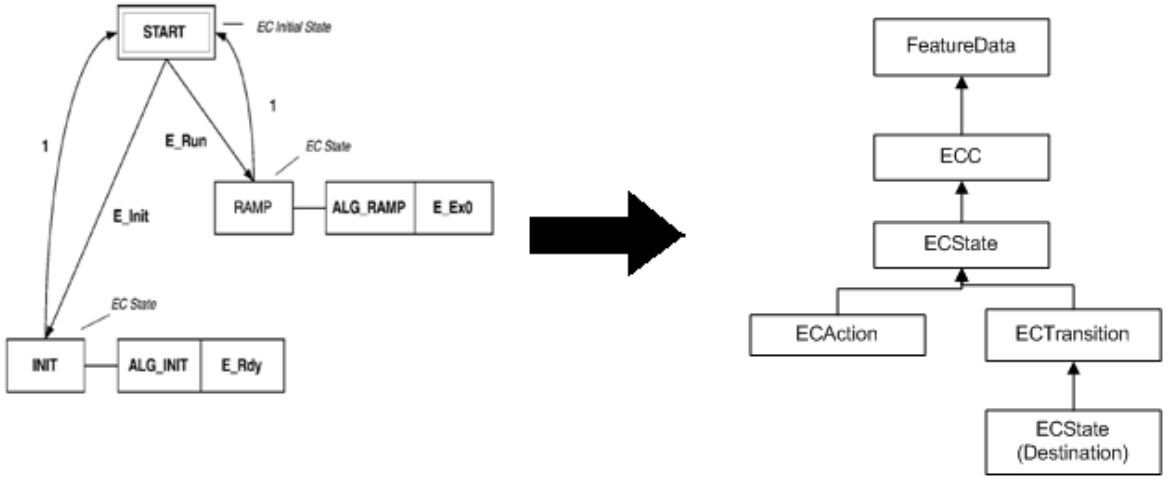


Figure 32: ECC´s class hierarchy taking into account the structure of the ECC (Lewis, 2001)

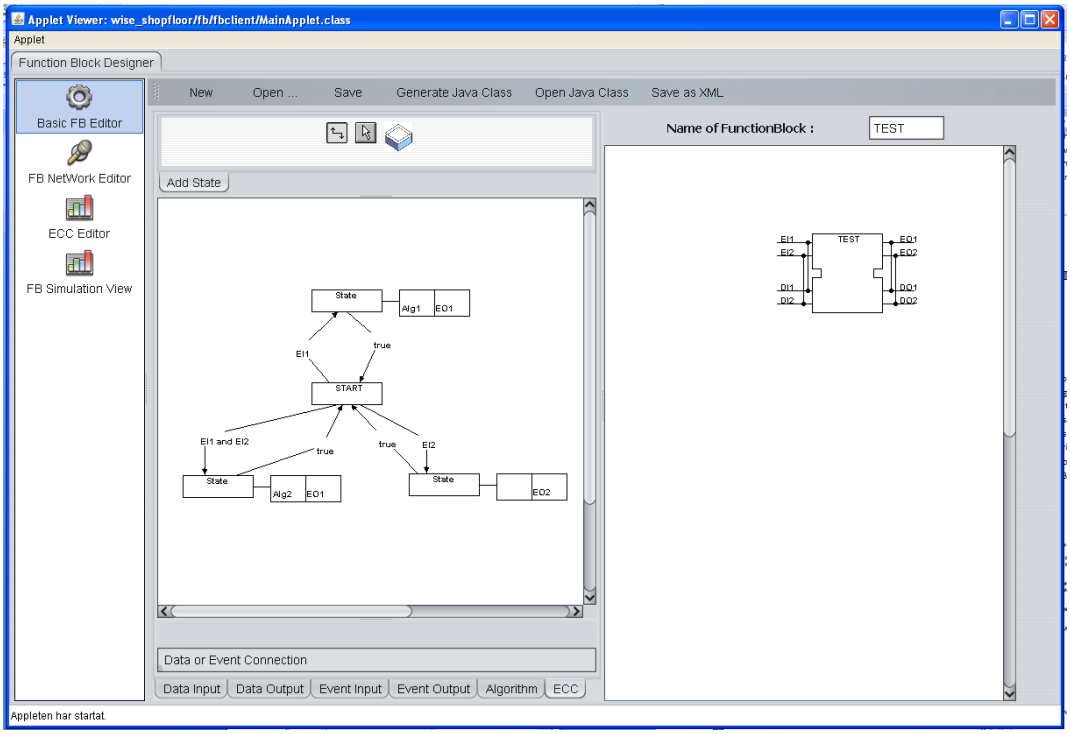


Figure 33: Example of the final result of the Basic FB editor while working with the ECC editor

5.1.3 Testing the ECC editor

Different tests have been carried out in order to ensure the good performance of the ECC editor. Most of the tests were launched while implementing, because any change in the interface could give problems, nevertheless, once that the hole environment was implemented, a general test have been done to be sure that a change in another part of the program did not have influence in the ECC editor. The tests have been important because in many cases any change of code line could have consequences in any part of the program.

To test this part the most common tests were associated to the interface, so most of the tests have been running the program. However, some tests have been carried out using the debug option of Netbeans. This option was mostly used at the beginning of the implementation and when saving data was important so that it could be ensured that all the data was correctly saved.

Most of the tests consisted in moving different EC states, adding actions, deleting these ones, ensure that the selected algorithm or event output was added in the correct EC action, etc... The part that has given more problems was the transition connection, but there were other problems with the EC States. The “ECCBlockFigure.java” class was based on a previously implemented “BasicFunctionBlockFigure.java” class, so this has generated some problems when some variables

and methods were mixed between both. However, it was not very challenging to solve these problems because the origin of the error was found easily.

5.1.4 Algorithm editor

The ECC editor is able to execute algorithms of the basic FB. There have been some improvements in the algorithm editor with respect to the previous platform. The previous algorithm editor could just add or delete algorithms, but it was not able to edit them. The first step taken for the algorithm editor was to add a "jTextArea" in order to be able to write the algorithm. So adding this area that allows writing strings, the algorithm editor will have a table with the name of the algorithms and the area, as well as three buttons (add and delete the selected algorithm and save the edited algorithm).

The algorithms are saved in a vector that contains the name of the algorithm and the written text in the previously mentioned area. The program has been implemented so that it detects which algorithm is selected from the table, so when an algorithm is selected, the edited text is uploaded to the text editor. Figure 34 shows how the algorithm editor of a Basic FB looks like.

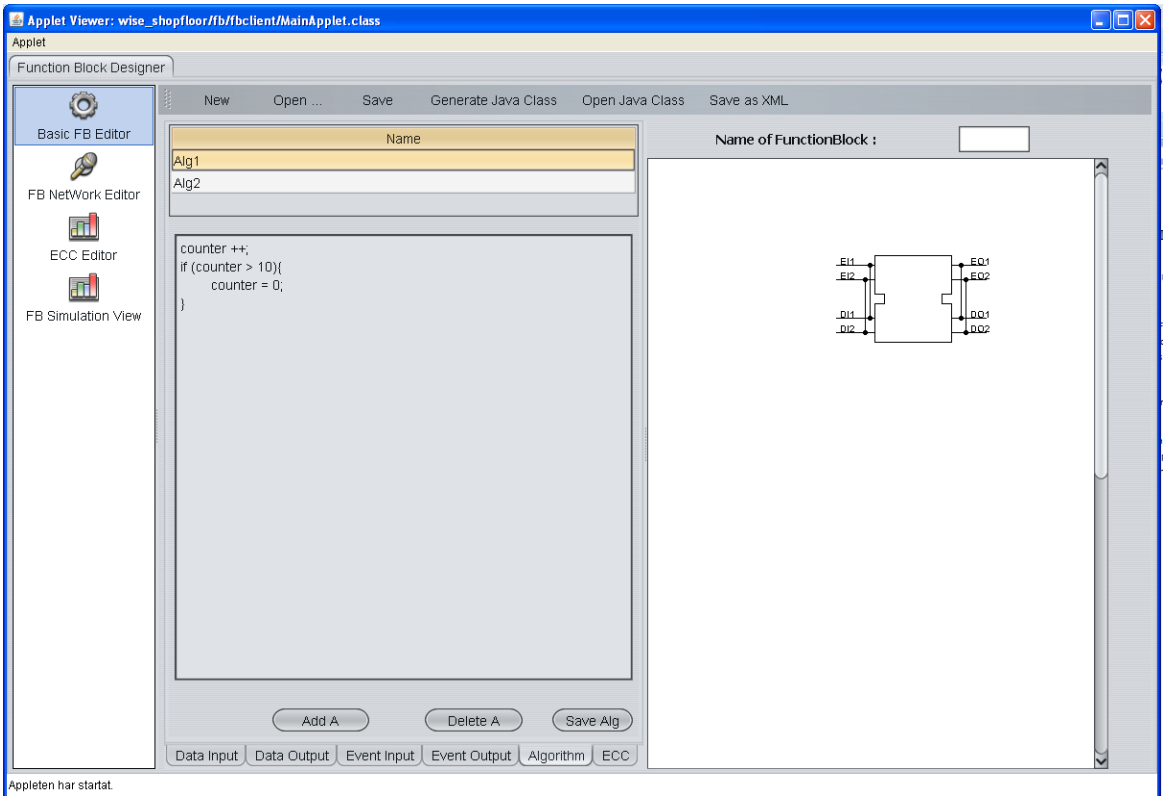


Figure 34: Interface of the algorithm editor

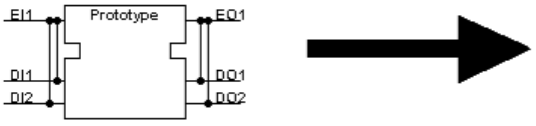
This part has not been very challenging, and the tests that have been accomplished have been using the debugger to be sure that all the data was stored in a correct way in the vectors. The problems

that have appeared in this part have been about selecting the algorithms with the mouse, because sometimes detected but other times not. This problem has been solved using another function.

5.2 Java Class file

5.2.1 Saving java class files

Different saving systems are possible in order to save and open a FB configuration, allowing portability with the same environments that could open and save using the same system. In (Anasagasti, 2012) a method to save in XML file is shown, in (Yoong, et al., 2009) a method to save the files in a ".c" fine is described, whereas in this project the saving system is going to be in Java Class file. Java Class is a ".java" file that represents a class programmed in Java language, and in this case a FB configuration could be in Java code saving the configuration in a Java Class file, so that execution of FB could be achieved thanks to compiled Java. Figure 35 shows an example of the Java class file representation of a Basic FB.




```
/**
 * File: Prototype.java
 **/

package wise_shopfloor.BFB;
import wise_shopfloor.FBClasses.*;

public class Prototype extends FBCore {
  //Input Data Variables
  FBData DI1 = new FBData("DI1","",this);
  FBData DI2 = new FBData("DI2","",this);

  //Output Data Variables
  FBData DO1 = new FBData("DO1","",this);
  FBData DO2 = new FBData("DO2","",this);

  //Input Event Variables
  FBEvent EI1 = new FBEvent("EI1","",this);

  //Output Event Variables
  FBEvent EO1 = new FBEvent("EO1","",this);
  // Default Constructor
  public Prototype( ){
    // constructor body
    init();
  addDI(DI1);
  addDI(DI2);
  addDO(DO1);
  addDO(DO2);
  addEI(EI1);
  addEO(EO1);
  }

  /* The following are event call back functions and internal functions*/
  // Process input event EI1
  public void processEI1(){
    // Method body
  }

  /* ECC function*/
public void ECC(){
}
  /* ALG_INI function*/
public void ALG_INI(){
}
  /* ALG_RUN function*/
public void ALG_RUN(){
}
}
```

Figure 35: Translation of a Basic FB into a Java class file

For saving and opening files in Java class code, two new buttons were created into the FB editors: “Open Java class file” and “Save in Java class file”. Pushing these buttons, it is going to be redirected into a direction of the Hard Disc of the computer so that the user can select where to save the configuration. After that, some tests are going to be designed to ensure that the portability of the environment works in a good way.

An algorithm has been designed so that all the parameters that have to be taken into account when translating a Basic FB into Java code could be written in the file correctly. Eleven steps has to be done so that the Java class file could be created, and nine of those will be in charge of the code generation, whereas the first step will create the class file into a folder in the hard disc and the last step will compile the generated code. Figure 36 depicts the flow diagram with the steps of saving a file in Java class code.

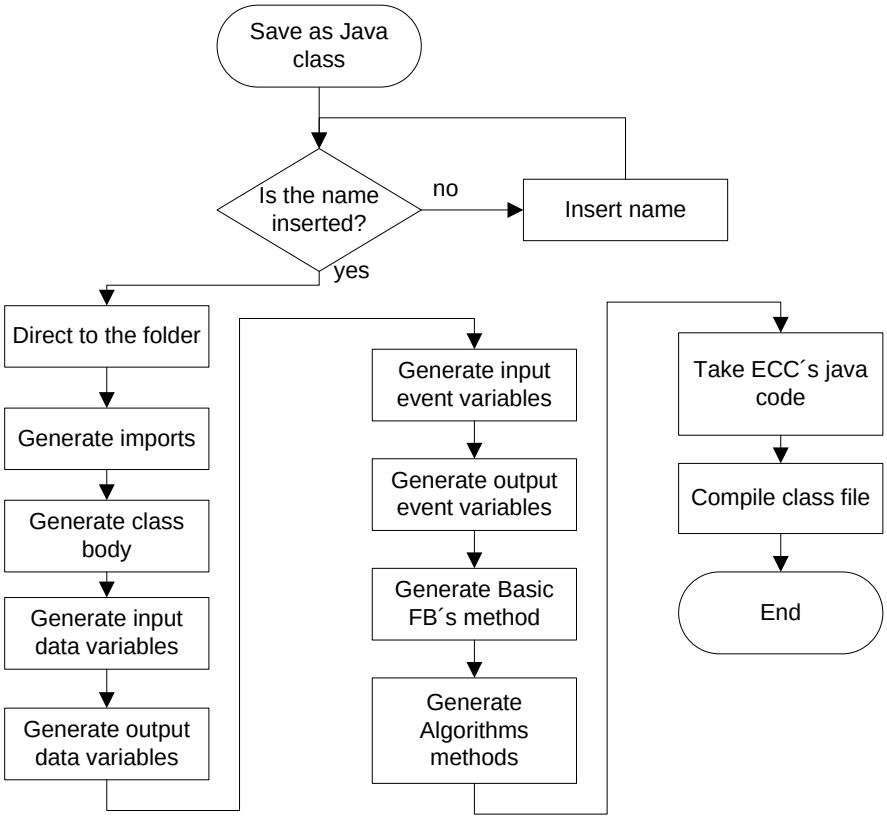
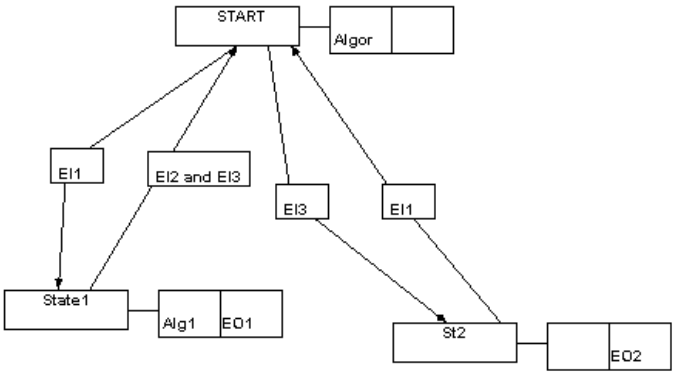


Figure 36: Flow diagram of the designed algorithm to generate Java class file

The generation of the ECC´s Java code has been implemented in the “ECC.java” so that the implementation could be easier. Java code for the IEC 61499 is not implemented in any FB

environment, so firstly a design of how the Java code was going to look has been designed. The ECC is going to be the chart that will trigger the implemented algorithms and the event outputs, so the design of the ECC Java code has been done so that this compiled code could work on a runtime environment. Each ECC will have its own method in the class that the Java code is generated; the environment was designed so that each EC State could have an ID different to the rest of the EC States, and it is not going to be changeable, so that this ID could allow the program difference between more than one EC State even if they have the same name. The ID number of the "START" type EC State will be "0", and the Java code will be configured so that the ECC could start from this previously mentioned EC State. Two "switch-case" statements will allow controlling the ECC in which EC State is running. The first statement will be in charge of checking if any transition in accordance to that EC State is activated, and if it is, the ID is changed corresponding to the EC State that the ECC should be "jumped" and will reset the transition. The second statement will execute the algorithms and trigger the event outputs corresponding to the EC State of that ID. An example is shown in the Table 4, where the Java class code of an ECC is shown.

The IEC 61499 allows using some logical statements as conditions of the EC Transitions, in the terms of "AND", "OR"...and the result of these statements, a Boolean condition is given for each transition. The syntax of Java code does not allow using these statements in the same syntax as the IEC 61499 does, so these statement´s conditions had to be translated to Java syntax. For example, the "AND" statement was translated to "&&", so that the Java compiler could understand the logic statement. The implemented code can be found in the Appendix B of the project.




```
int id = 0;
public void ECC(){

int previousState;
while(true){
  previousState=id;
  switch(id){
       case 0:
       {
         if(EI1.check()){
            id=1;
            EI1.reset();
         }else if(EI3.check()){
         id=2;
         EI3.reset();
       }

       break;
       }
       case 1:
       {
         if(EI2.check() && EI3.check()){
                    id=0;
                    EI2.reset();
                    EI3.reset();
         }
       break;
       }
       case 2:
       {
       if(EI1.check()){
                    id=0;
         EI1.reset();
       }
       break;
            }
  }
  switch(id){
     case 0:
            {
       //Action:1
       Algor();
       break;
            }
            case 1:
            {
       //Action:1
       Alg1();
       EO1.trigger();
       break;
            }
            case 2:
            {
                    //Action:1
                    EO2.trigger();
                    break;
     }
}
if(previousState==id) break;
}
}
```

Figure 37: Java Class code for an ECC

5.2.2 Opening files

For opening a Basic FB both XML and Java class file are needed if the complete FB is needed. As mentioned before, a FB can have event input and output, data input and output, the ECC and algorithms. The basic FB´s XML file, implemented in (Anasagasti, 2012), contains almost all the information of a basic FB, so it is possible to open a big part of the file without using the Java class file. However, the XML file does not contain information about the algorithms; the algorithms are stored in the Java class file, so this project has implemented an opening method that updates the algorithms. Figure 38 shows the steps taken when opening a new file.

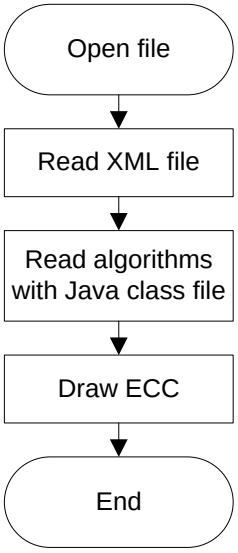


Figure 38: File opening method

Once that the files where opened, the figures are drawn. This project has been in charge of drawing the ECC. When saving files, all the times we added an EC State, a vector was created with all the information containing from the “ECCBlockFigure.java” class (interface class for the EC States). The same happened with the transition, when a transition was added, some vectors were created with the information taken from the “TransitionConnection.java” class. A new method was created in “ECC.java” class, that instead creating new vectors depending on what the information are in the interface classes, the interface classes updated their information depending on the data of the vectors. Figure 39 presents the steps that have to be taken into account for drawing the ECC. The used code can be seen in Appendix C.

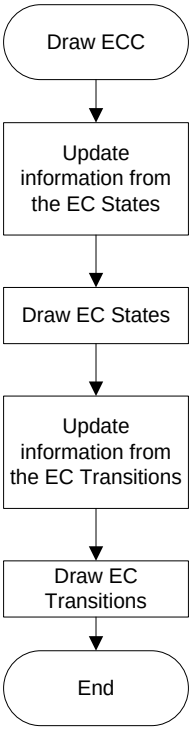


Figure 39: Flow chart for drawing the ECC

5.2.3 Testing the saving and opening options

Some tests have also been carried out in this part of the project to ensure possible problems not to appear and ensure everything was working properly. Most of the tests of the XML file have been implemented in (Anasagasti, 2012), but this project also have implemented some of them because the ECC was saved and opened with the XML. The first tests were about implementing some basic FBs that and ensuring that the files, Java class and XML, were correct: the Java class file according to our design, and the XML according to the standard IEC 61499 (IEC, 2005).

Once ensured that the saving options were working properly, some XML files were taken and edited manually. When the code was correct, the file was opened in the program and it was ensured that the data in the environment was accorded to the edited file.

The algorithms were read from the Java class file, so in the Java class file the algorithms were written and after that uploaded. Those algorithms were displayed in the algorithm editor´s text area. As a conclusion, it is possible to say that in some cases, when the algorithms of the FB are quite complex, it is better to change them from the Java class file using NetBeans environment, because it is more intuitive, and the errors are easier to be detected.

6 Case Study

(Part implemented simultaneously with Mikel Anasagasti (Anasagasti, 2012)*)*

After ending any kind of work, a thesis in this case, it is always essential to perform some validations 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. 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. On it 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 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 40) already integrated in Wise-ShopFloor (Wang, 2009). 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"*.

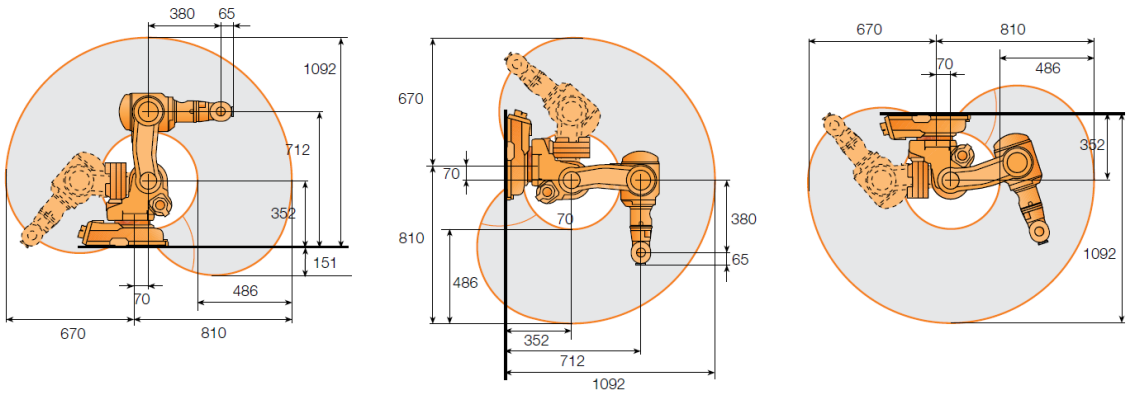


Figure 40: Sizes, configurations and the region of space the robot can reach for each configuration (ABB, 2012)

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 measure will be known and depending on the detected measure, the robot will perform 3 different paths. So, the measure inspection workstation 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 41 shows the robot working during the assembling process.

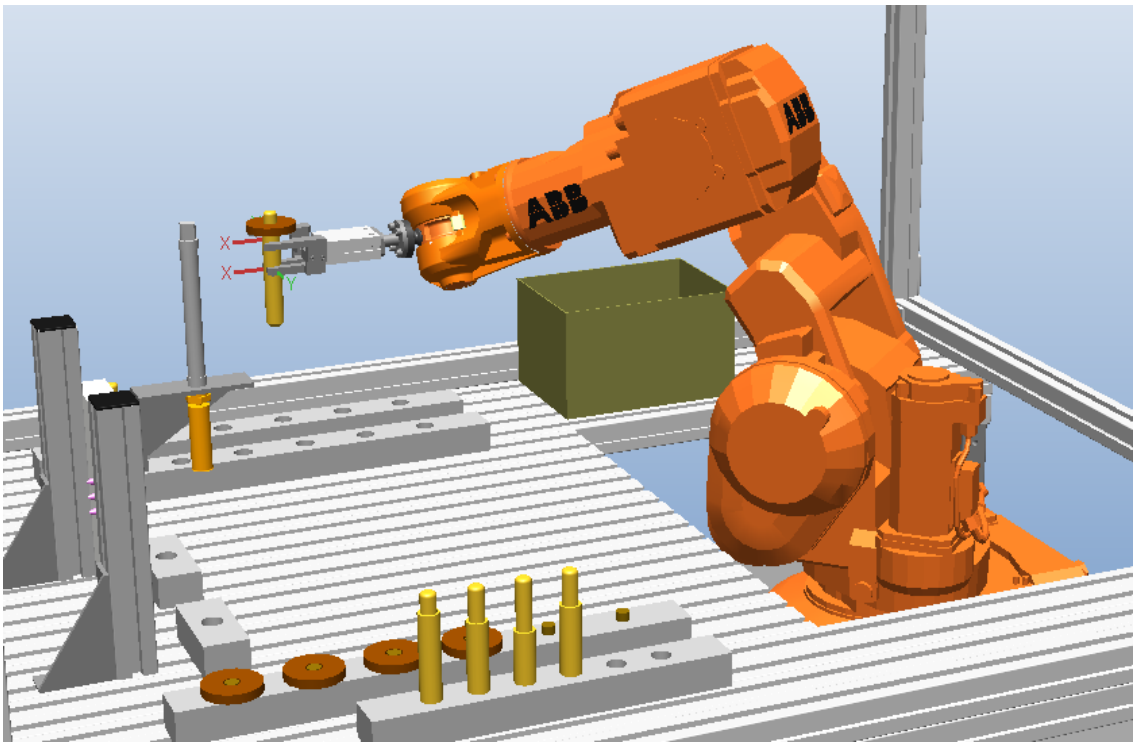


Figure 41: Robot working during the assembling process

This process will be performed one by one in 6 occasions (total number of shafts and washers). On the other hand, Figure 42 presents the flow chart of the assembly process whereas Figure 43 shows the entire mini-cell 3D model.

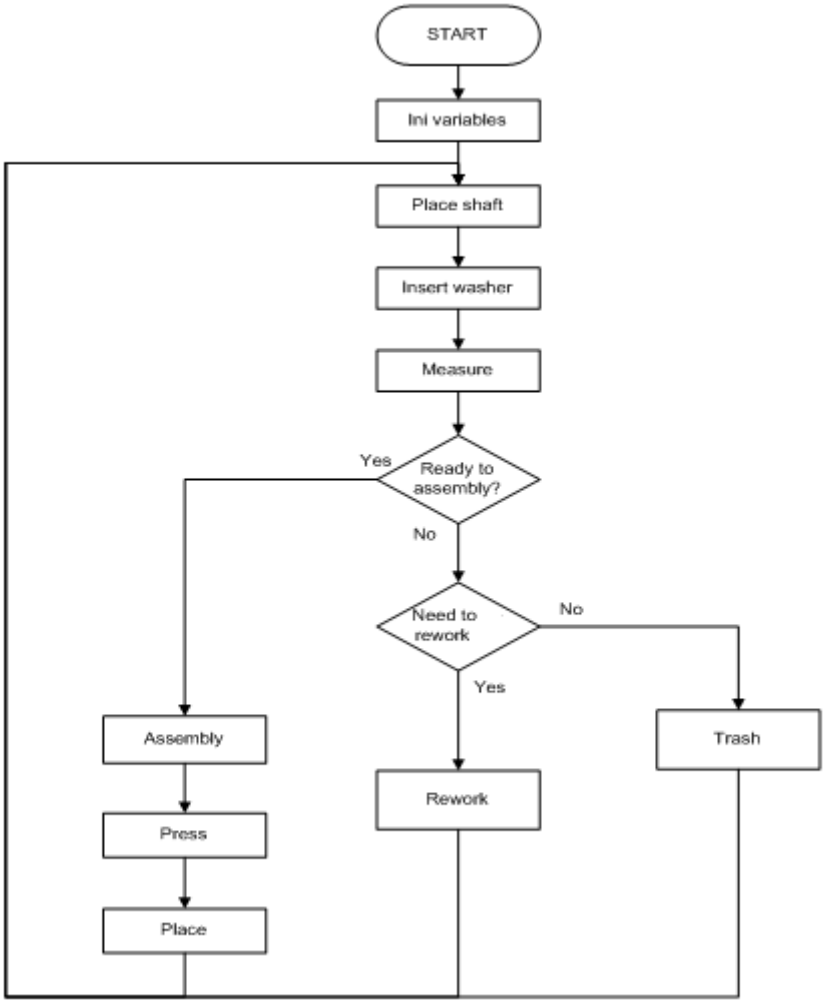


Figure 42: Flow Chart of the simulated assembly line

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-ShopFloor, 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).

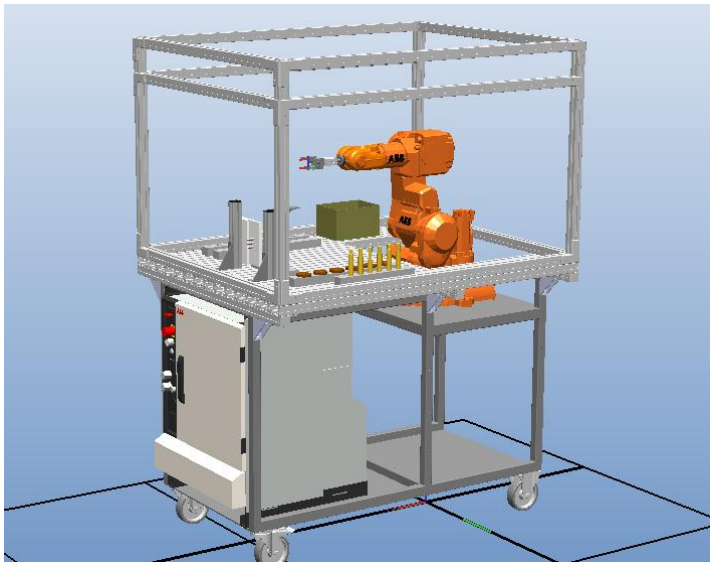


Figure 43: 3D model mini-cell robot assembly station

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 44. 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 Figure 45.

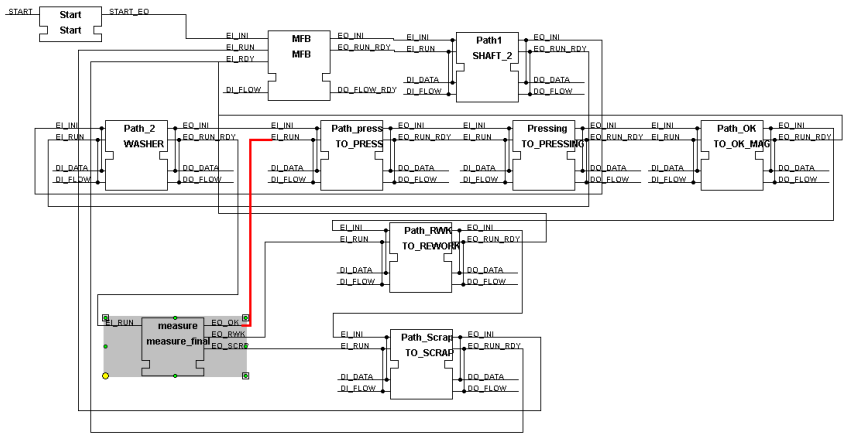


Figure 44: Simulation of the assembly process

On the one hand, Figure 44 shows the created FB Network while it is running. The red line is the indicator of what part of the network is simulating in each moment. The algorithm in measure FB decides which of the three paths to follow. On the other hand, Figure 45 presents 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.

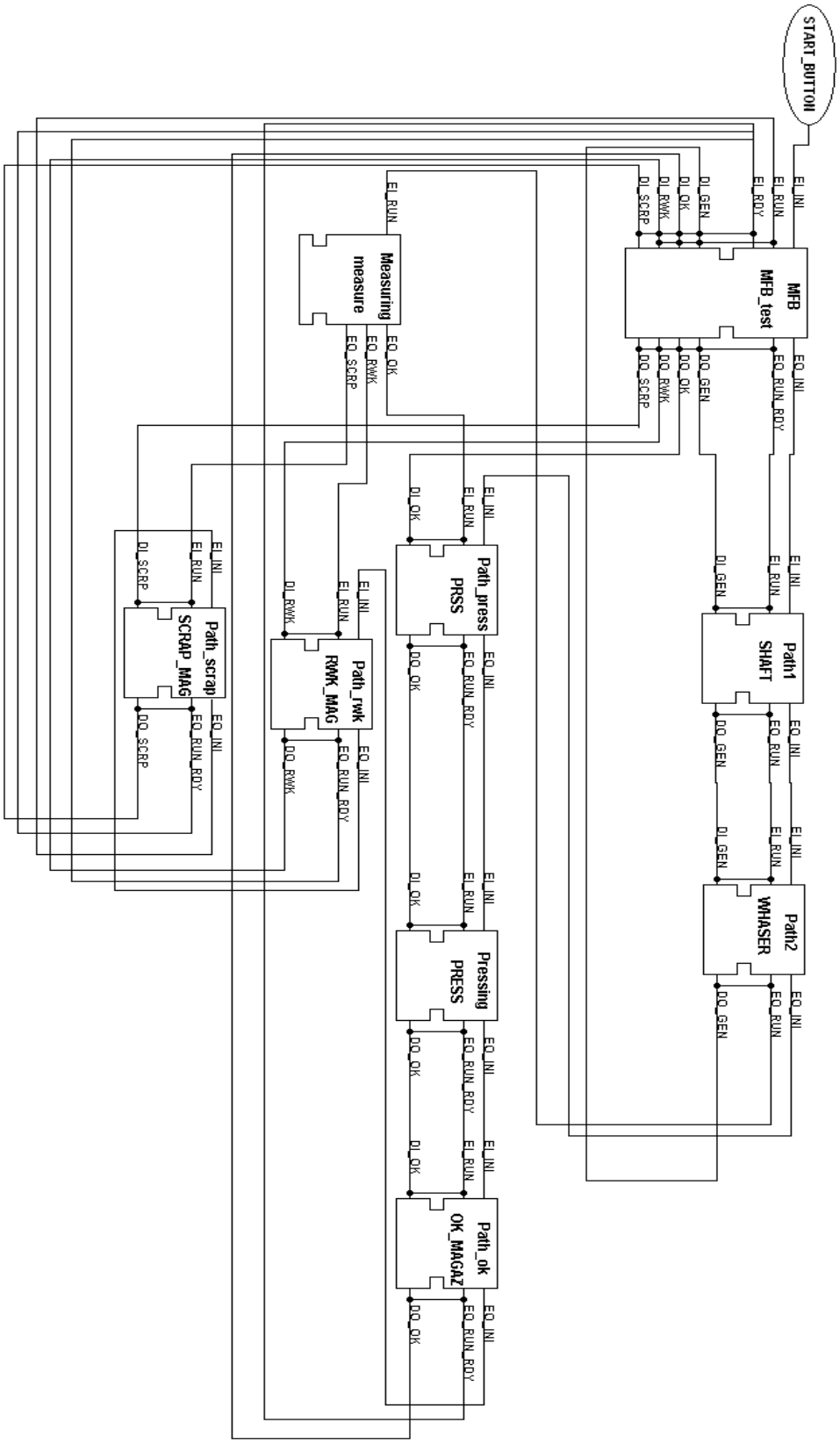


Figure 45: Function Block network for the control of the ABB IRB 140 robot in mini-cell assembly station

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 measure 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 choose. 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 convenient 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 45, 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.

7 Discussion

In the previous pages the implementation and design of a Web-based environment that fulfils the requirements of the IEC 61499 are explained. The IEC 61499 is a new standard provided by the International Electrotechnical Commission published in the 2005 for the use of Function Blocks in the industry. This new technology allows interoperability, portability and configurability as well as other advantages explained before. However, the IEC 61131-3 is a standard that is more used nowadays in the industry due to its wide acceptance. Even if the IEC 61499 has some advantages over IEC 61131-3, it is going to take some time to replace the old standard. The IEC 61131-3 has been used since long time ago, the engineers and technicians are used to program PLCs with this standard, and for workers and companies it is difficult to be "recycled", moreover when they have to implement a program for a client that does not have the knowledge of a new technology.

The saving systems that allow portability to the IEC 61499 have been discussed before. This project has been focused on saving the configuration of the FBs in Java class file. Nevertheless, it has to be said that there are not many research articles that focuses on Java class code saving method, moreover, the commercial environments that have been used for the research are able to save their configuration in XML file.

Referring to the problems that have been appearing during the implementation of the project, it has to be said that they were technical problems that could be solved by taking some time redesigning the environment and programming again. However, these problems and errors have been part of the project, because every day there have been new problems.

It is important to mention that programming skills have been important during the implementation of the project. When the project was started, Java programming language has been learnt, because it was never used before by the author. This environment is an automation tool and can be developed by an automation engineering student because the profile of these engineers is joined with programming, however, a computer science student would have developed this project easier, but he could have problem with aspect directly related to automation.

The case study has been useful to ensure that the program was working on a correct way, and to fix the mistakes we did. It has also been useful to take some conclusions about the advantages and disadvantages of using Function Blocks.

8 Conclusions and Future Work

This report is a bachelor degree project, which shows the research of the IEC 61499 standard, the state of the art of a software web-based environment that fulfils the requirements of the IEC 61499 standard, as well as the methodology and implementation to develop the web-based Java platform. The first months have been used for the work in the research of this new standard; the investigation has been very important in order to start with the design and implementation of the environment, and thanks to some books and different research articles a good basis has been adopted. The last months have been invested for the implementation of the web-based environment and for the simulation of the case-study.

Different Java tutorials and exercises have been developed to learn how to program in Java programming language that it was never used by the developer of the project before. However, it has to be said that this programming language is similar to another that was used for other projects, and gives a lot of advantages thanks to its facilities. It was important also to take some time to implement some designs of the program before starting to write the code. Apart of that, the program provided by the University of Skövde has helped when developing some parts of the environment.

The work in team has also been very important. Firstly, the collaboration between the projects that were similar to this one, in which the authors of (Anasagasti, 2012), (Arriouach, 2012), (Díaz, 2012) and (Palomeque, 2012) have been collaborating together with recommendations of books and research articles, different programming techniques and other technical parts that have been mentioned in the previous paragraphs. Secondly, the different seminars that have been given by Prof. Lihui Wang and others during the first months have been very useful, as well as the meetings with some teachers of the university to be recommended about the report. Finally, the different meetings with the supervisor have been determinants during the project.

A Mid-Term report was handed in during the project where the future work was explained in form of tasks for the following months: the ECC and the Algorithm editor, the Java class saving mode, and the case study. Those tasks were developed in the last months of the project successfully. The project was also joined with another bachelor degree project, (Anasagasti, 2012), and with the work of Bernard Schmidt, that has provided to this project the FB simulator and the FB Runtime that was not in the scope of the project.

Nevertheless, some more improvements could be done in the future. According to the main scopes of this project, the improvement suggested is linked with the algorithm editor, in which new options of programming in IEC 61131-3 standard´s languages could be developed (mainly Structured Text and Ladder) and then translated this into Java class, so that the environment could fulfil better the requirements of the IEC 61499.

According to the ECC editor, some improvements could be done, in which the editor could indicate to the user if the written transitions are allowed or not. Following the same line, another improvement could be to edit the transitions clicking twice in it, and not deleting the whole transitions.

The environment could be integrated in the project Wise-ShopFloor, and some FBs could be developed to control the robots already integrated in this project.

The IEC 61499 is oriented to distributed control systems, which means that the program could be run in an embedded system. Another improvement could be that the developed program could be downloaded into a PLC. For this, different communication protocols should be investigated depending on the market of PLCs, but usually the program is downloaded via USB, Ethernet or RS 232. It would be interesting to test the program working in an embedded system.

As it was mentioned before, this project could be implanted in the near future in Volvo Cars and in Sandvik AB. The user's interface could be implemented for these special clients with their logotype, as well as with some special FB libraries that they could use for a concrete work.

9 References

4DIAC, 2008. *4DIAC.* [Online]
Available at: http://www.fordiac.org
[Accessed 09 02 2012].

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

Anasagasti, M., 2012. *Function Block Environment in Wise-Shop floor: Graphical user interface,* Skövde: University of Skövde.

Arriouach, K., 2012. *SCARA ROBOT - FB,* Skövde: University of Skövde.

Cana, J. & Gil, C., 2012. *3D Model- driven distant assembly,* Skövde: University of Skövde.

Cervera, A., 2012. *Design of an Information Carrier Device for a Decision Support System,* Skövde: University of Skövde.

Christensen, J. H., 2012. *Holobloc.* [Online]
Available at: http://www.holobloc.com/papers/iec61499/ovw_body.htm
[Accessed 09 02 2012].

Díaz, R., 2012. *DEVELOPMENT OF A MANUFACTURING CELL IN COMPLIANCE WITH IEC 61499: Implementation of a function block network for controlling a CNC-based system,* Skövde: University of Skövde.

Doukas, G. S., Thramboulidis, K. C. & Koveos, Y. C., 2006. Using the Function Block Model for Robotic Arm Motion Control. *Mediterranean Conference on Control and Automation.*

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.*

IEC, 2005. *International Standard, IEC 61499.* Geneva: International Eletrotechnique Commission.

IEC, 2012. *International Eeletrotechnical Commision.* [Online]
Available at: http://www.iec.ch/
[Accessed 08 02 2012].

IEC, 2012. *International Electrotechnical Commission.* [Online]
Available at: http://www.iec.ch/about/profile/
[Accessed 05 02 2012].

IEC, 2012. *International Eletrotechnical Commission.* [Online]
Available at: www.iec.ch/about/
[Accessed 05 02 2012].

Lastra, J. L. M., Godinho, L., Lobov, A. & Tukko, R., 2005. An IEC 61499 Application Generator for Scan-Based Industrial Controllers. *IEEE International Conference on Industrial Informatics,* pp. 80-85.

Lewis, R., 2001. *Modelling Control Systems Using IEC 61499.* London: The Institution of Engineering and Technology.

Machado, J. et al., 2011. Safe controllers design for industrial automation systems. *Computers & Industrial Engineering,* Volume 60, p. 635–653.

Minhat, M., Vyatkin, V., Xu, X. & Al-Bayaa, Z., 2009. A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks. *Robotics and Computer-Integrated Manufacturing,* pp. 560-569.

Netbeans, 2012. *Netbeans.* [Online]
Available at: http://netbeans.org/
[Accessed 14 February 2012].

Palomeque, J. E., 2012. *DEVELOPMENT OF A MANUFACTURING CELL IN COMPLIANCE WITH IEC 61499: Implementation of a function block network for controlling a CNC-based system,* Skövde: University of Skövde.

Schwab, C., Tangermann, M. & Ferrarini, L., 2005. Web based Methodology for Engineering and Maintenance of Distributed Control Systems: The TORERO Approach. *3rd IEEE Intemational Conference on Industrial Informatics,* pp. 32-37.

Sousa, M. d., 2008. Restricting IEC 61131-3 Programming Languages. pp. 361-368.

Strasser, T. et al., 2006. Modeling of Reconfiguration Control Applications based on the IEC 61499. *Proceedings of the IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications.*

Sünder, C. et al., 2007. Execution Models for the IEC 61499 elements Composite Function. pp. 1169-1175.

Sünder, C. et al., 2006. Usability and Interoperability of IEC 61499 based distributed automation systems. pp. 31-37.

Su, Q., Cao, Y. & Liang, G., 2009. The Strategy of Java Class File's Modification. *Second International Workshop on Computer Science and Engineering,* pp. 322-326.

Transpaderne, T. & Vidal, E., 2012. *Wise-Shop Floor decision support system,* Skövde: University of Skövde.

Venners, B., 1996. *Java World.* [Online]
Available at: http://www.javaworld.com/javaworld/jw-07-1996/jw-07-classfile.html
[Accessed 2 March 2012].

Vyatkin, V., 2009. The IEC 61499 Standard and Its Semantics. *IEEE INDUSTRIAL ELECTRONICS MAGAZINE,* pp. 40-48.

Vyatkin, V., 2011. *IEC 61499 Function Blocks for Embedded and Distributed C Control Systems Design.* Second ed. Auckland: s.n.

Wang, L., 2009. *University of Skövde.* [Online]
Available at: http://www.his.se/english/university/contact/staff/lihui-wang/research-interests/wise-shopfloor/
[Accessed 31 01 2012].

Wang, L., Feng, H. Y. & Cais, N., 2003. Architecture Design for Distributed Process Planning. *Journal of Manufacturing Systems,* Volume 22, 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. & Yijun Song, Q. G., 2009. Designing function blocks for distributed process planning and adaptive control. *Engineering Applications of Artificial Intelligence,* pp. 1127-1138.

Wenger, M., Zoitl, A., Sünder, C. & Steininger, H., 2009. Semantic correct transformation of IEC 61131-3 models into the IEC 61499 standard.

Xu, X. W., Wang, L. & Rong, Y., 2006. STEP-NC and Function Blocks for Interoperable Manufacturing. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING,* Volume III, pp. 297-308.

Yoong, L. H., Roop, P. S. & Salcic, Z., 2009. Efficient Implementation of IEC 61499 Function Block. *Industrial Technology, 2009. ICIT 2009.*

Yoong, L. H., Roop, P. S. & Salcic, Z., YEAR. Efficient Implementation of IEC 61499 Function Block.

Yoong, L. H., Roop, P. S., Vyatkin, V. & Salcic, Z., 2009. A Synchronous Approach for IEC 61499 Function Block implementation. *IEEE TRANSACTIONS ON COMPUTERS,* 58(12), pp. 1599-1614.

Zoitl, A., Strasser, T., Sünder, C. & Baier, T., 2009. Is IEC 61499 in Harmony with IEC 61131-3?. *IEEE INDUSTRIAL ELECTRONICS MAGAZINE,* pp. 49-55.

10 Appendices

10.1 Appendix A: Manual of the Basic FB Editor

It is possible to run the program from NetBeans java programming environment, or from the internet using the URL http://wise-shopfloor.his.se/WebFB/. Using the web based environment, the user will be able to use the last updated version to the server. By default, the program will start running in the Basic FB editor as shown in Figure 46.

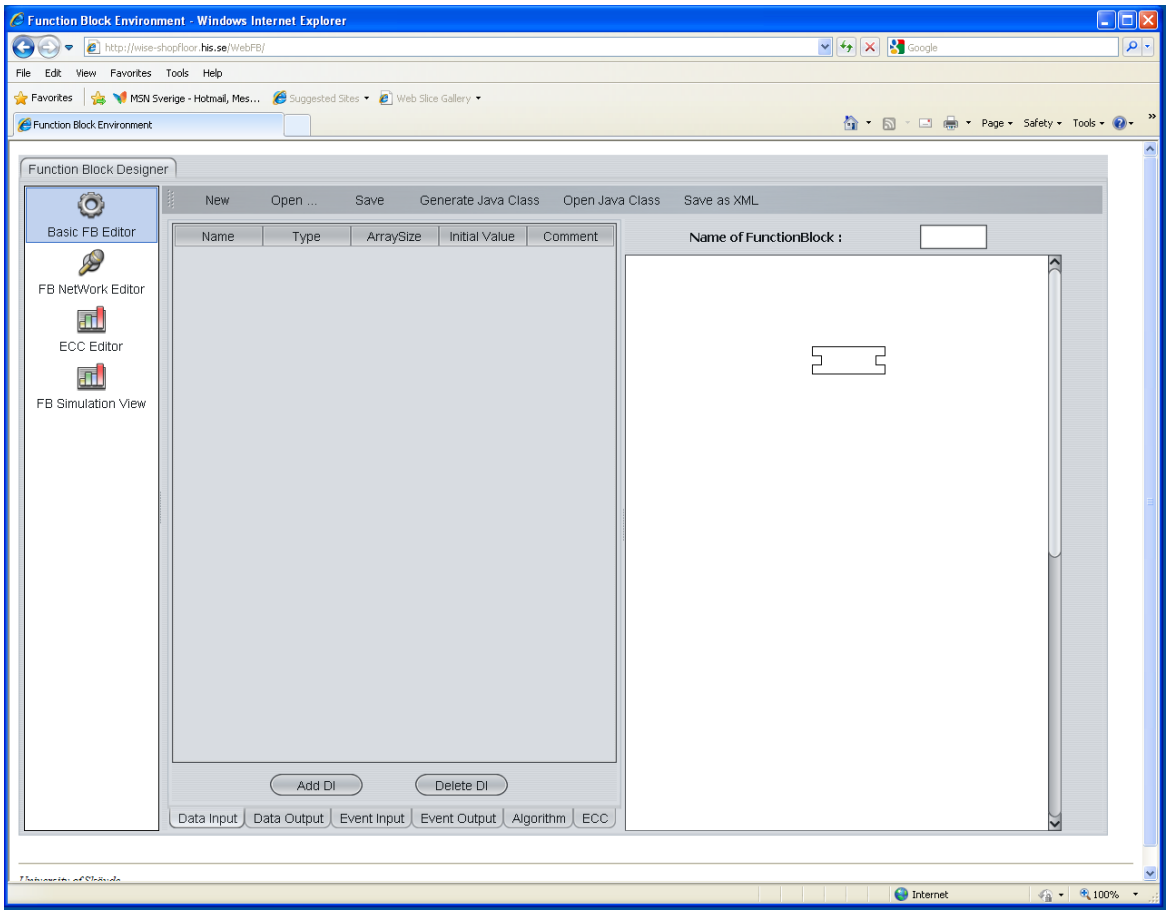


Figure 46: General overview of the Web-Based environment

In the figure, it is possible to see a menu in the left side, that allows to work with different editors, Basic or network Function Blocks. Focusing on the Basic FB editor, two panels can be seen. The right panel shows how the FB looks like from its outside appearance, which are the event and data inputs and outputs and the name. The left panel can be used to edit the basic FB, to add or delete the data of the data IOs, event IOs, algorithms and ECC. The IOs can be added or deleted using the buttons of ADD or DELETE. To use the DELETE button, the data that is wanted to be deleted has to be deleted.

Regarding the algoriths, it is not possible to write them in Structured Text programming language, the algorithms have to be written in Java code, and it is important to save the algorithm before starting to work with another configuration of the FB. Figure 47 shows the algorithm editor.

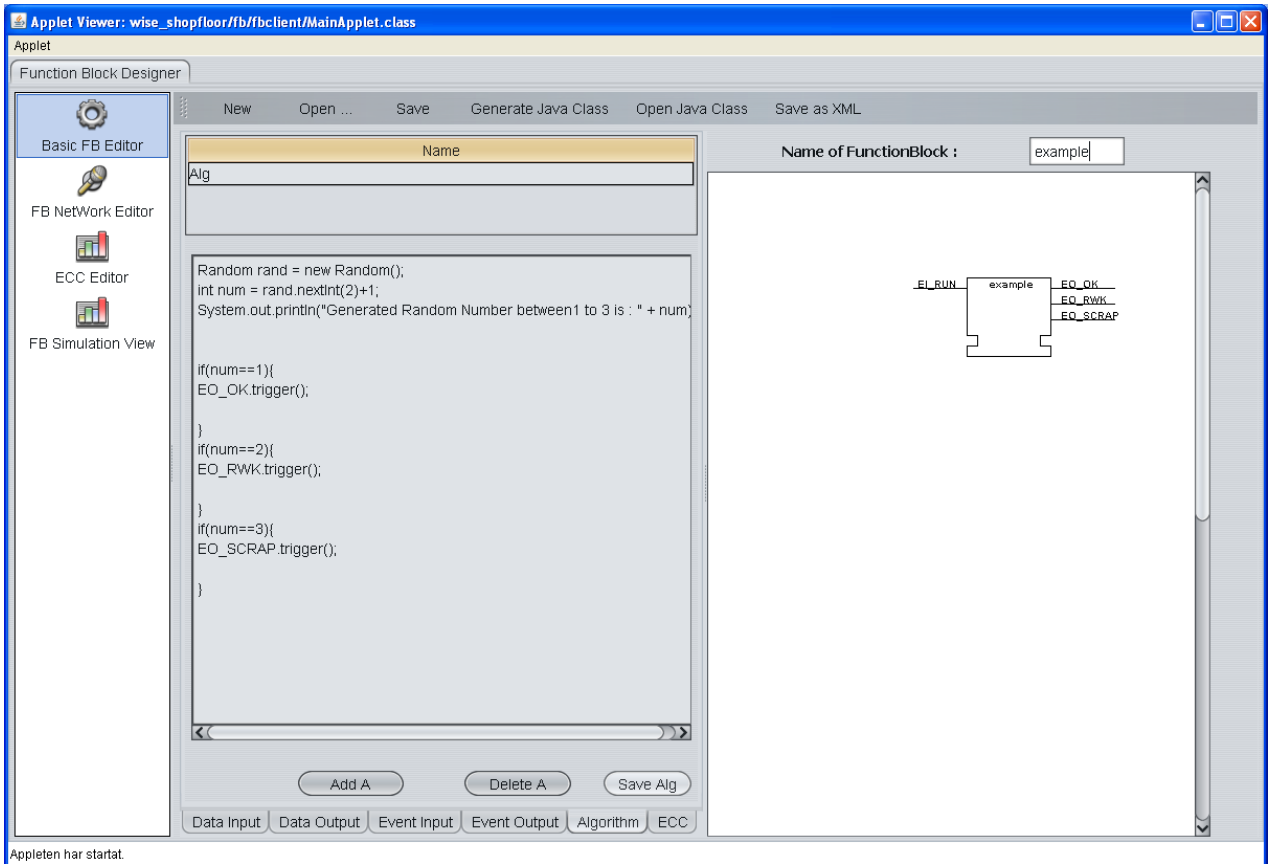


Figure 47: Algorithm editor of the Basic FB

Finally, the ECC will be in charge of executing the algorithms and triggering the event outputs. A new EC State is added using the selection tool (second in Figure 49) when dragging the block that is at the top part of the panel to the panel (Figure 48). After adding at least two EC States, the next step would be to add EC actions. To do that a right click inside the EC State is needed and a menu will appear, with the option of adding or deleting an action and changing the name of the EC State. Once that the user adds the action needed, if you right click inside the EC Action it will have the option of adding or deleting algorithms or event outputs associated to that EC Action. Using the connection tool (first in Figure 49), the user will be able of joining two EC States by a transition. To do this, the user should bring the cursor close to the EC State until the connectors appear, click there, and while clicked, drag it to another EC State´s connectors; after doing that, a message will be appeared asking you to write the EC Transition.

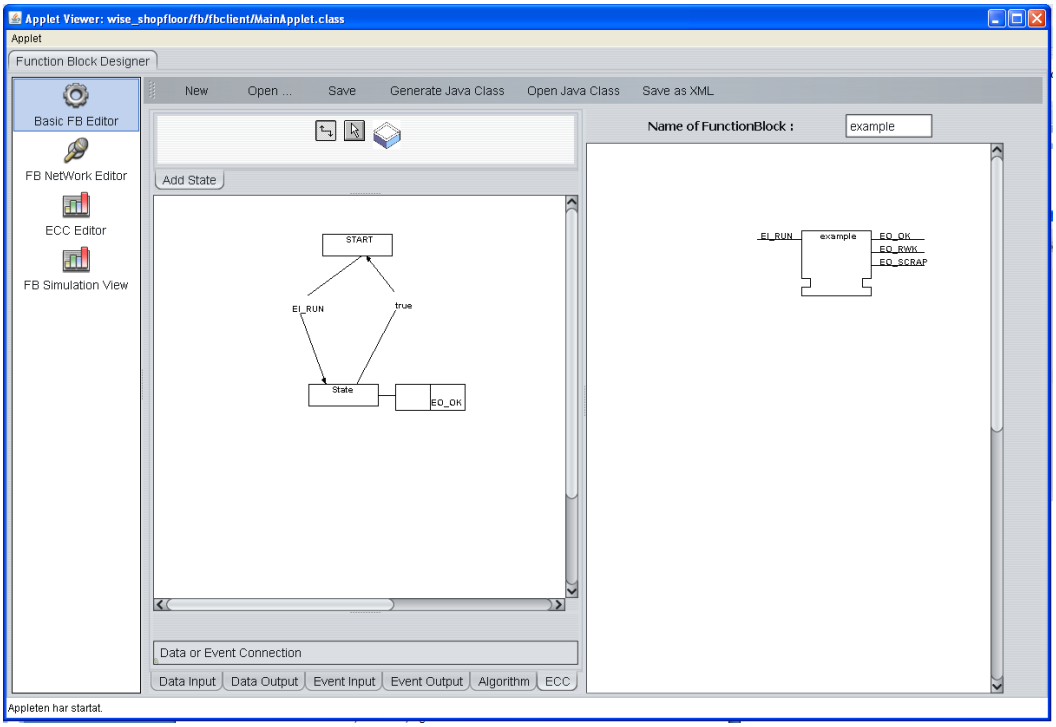


Figure 48: ECC editor of the Basic FB

Figure 49: From right to left: connection tool, selection tool, EC State block

10.2 Appendix B: ECC java code generation

```
public void writeECC(URLConnection con){
   try{
   update(getDrawing());
   OutputStream out = con.getOutputStream();
   PrintStream pout = new PrintStream(out, true);
   pout.print("int id = 0; \n");
   pout.print("public void ECC(){\n\n");
   pout.print("int previousState;\n");
   pout.print("while(true){\n");
   pout.print("previousState=id;\n");
   pout.print("\tswitch(id){\n");
   for(int i=0; i<States.size(); i++){
      pout.print("\t\tcase "+States.elementAt(i).id +":\n");
      pout.print("\t\t{\n");
      for(int j=0; j<States.elementAt(i).getTransition().size(); j++){
         if(j==0){
            Vector vec = featureData.getEventInput();
            String cond = States.elementAt(i).getTransition().elementAt(j).getCondition();
            for(int k =0; k<vec.size(); k++){
               EventData data = (EventData) vec.elementAt(k);
               String str_event = data.getName();
               cond = cond.replace(str_event, str_event+".check()");
            }
            cond = cond.replace("and", "&&");
            cond = cond.replace("or", "||");
            pout.print("\t\t\tif("+cond+"){\n");
```

```
            pout.print("\t\t\t\t
id="+States.elementAt(i).getTransition().elementAt(j).getDestination().id+";\n");
            for(int k = 0; k<vec.size(); k++){
               EventData data = (EventData) vec.elementAt(k);
               if(cond.contains(data.getName())){
                  pout.print("\t\t\t\t"+data.getName()+".reset();\n");
               }
            }
            pout.print("\t\t\t}");
         }else{
            Vector vec = featureData.getEventInput();
            String cond2 = States.elementAt(i).getTransition().elementAt(j).getCondition();
            for(int l =0; l<vec.size(); l++){
               EventData data = (EventData) vec.elementAt(l);
               String str_event = data.getName();
               cond2 = cond2.replace(str_event, str_event+".check()");
            }
            cond2 = cond2.replace("and", "&&");
            cond2 = cond2.replace("or", "||");
            cond2 = cond2.replace("=","==");
            pout.print("else if("+cond2+"){\n");
if("+States.elementAt(i).getTransition().elementAt(j).getCondition()+"){\n");
            pout.print("\t\t\t\t
id="+States.elementAt(i).getTransition().elementAt(j).getDestination().id+";\n");
            for(int k = 0; k<vec.size(); k++){
               EventData data = (EventData) vec.elementAt(k);
               if(cond2.contains(data.getName())){
                  pout.print("\t\t\t\t"+data.getName()+".reset();\n");
               }
```

```
            }
            pout.print("\t\t\t}\n");
          }
          //pout.print("Destination: " +
States.elementAt(i).getTransition().elementAt(j).getDestination().id);
        }
        pout.print("\n\t\tbreak;\n");
        pout.print("\t\t}\n");
      }
      pout.print("}\n");
      pout.print("\tswitch(id){\n");
      for(int l=0; l<States.size(); l++){
        pout.print("\t\tcase "+l+":\n");
        pout.print("\t\t{\n");
        for(int m = 0; m<States.elementAt(l).ecAction.size();m++){
          pout.print("\t\t\t//Action:"+(m+1)+"\n");
          if(States.elementAt(l).ecAction.elementAt(m).AlgName != null){
            pout.print("\t\t"+States.elementAt(l).ecAction.elementAt(m).AlgName +"();\n");
          }
          if(States.elementAt(l).ecAction.elementAt(m).EventName != null){
            pout.print("\t\t\t"+States.elementAt(l).ecAction.elementAt(m).EventName
+".trigger();\n");
          }
        }
        pout.print("\t\t\tbreak;\n");
        pout.print("\t\t}\n");
      }
      pout.print("}\n");
```

```
      pout.print("if(previousState==id) break;\n");

      pout.print("}\n");


      pout.print("}\n"); //ECC

      }catch (Exception e) {

         System.err.println(e.toString());

//          JOptionPane.showMessageDialog(this, e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);

      }

   }
```

10.3 Appendix C: Drawing the ECC

```
   public void draw()

   {

      drawing.removeAll();

      for(int i=0;i<States.size();i++)

      {

         ECCBlockFigure eccBlockFigure = new ECCBlockFigure();

         eccBlockFigure.setEcState(States.elementAt(i));

         eccBlockFigure.displayBox(new
Rectangle(States.elementAt(i).get_coord_state_x(),States.elementAt(i).get_coord_state_y(),0,0));

         eccBlockFigure.setFeatureData(featureData);

         States.elementAt(i).setEccBlockFigure(eccBlockFigure);

         //set t o ECState reference to ECCBlockFigure

         //....

         drawing.add(eccBlockFigure);

      }

      for(int i=0;i<States.size();i++)
```

```
    {
        Vector transition = ((ECState)States.elementAt(i)).getTransition();
        for(int j=0;j<transition.size();j++)
        {
            ECTransition ecTransition =(ECTransition)transition.elementAt(j);
            TransitionConnection tConnection = new TransitionConnection();
            tConnection.setTransition(ecTransition);
            int x=ecTransition.getCoordinate_x();
            int y=ecTransition.getCoordinate_y();
            tConnection.displayBox(new Rectangle(x,y,0,0));
            tConnection.setFeatureData(featureData);
            Connector fStartConnector = States.elementAt(i).getEccBlockFigure().findConnector(x,y);
            Connector fEndConnector =
ecTransition.getDestination().getEccBlockFigure().findConnector(x,y);
            tConnection.connectStart(fStartConnector);
            tConnection.connectEnd(fEndConnector);
            tConnection.updateConnection();
            //....
            drawing.add(tConnection);
        }
    }
  }
```