An IEC-compliant Engineering Tool for Distributed Control Applications

Christos Tranoris, *Member, IEEE,* and Kleanthis Thramboulidis, *Member, IEEE*

***Abstract*-- To address the need of modern manufacturing plants to quickly respond to market requirements by designing competitive products and modifying existing ones, evolving IEC standards like 61499 and 61804 define a methodology to be used by system designers to construct distributed industrial control applications. New generation IEC compliant Engineering Support Systems (ESSs) are highly required to support the whole development process. In this paper we present an IEC-compliant ESS that we have under development. This ESS is based on our 4-layer architecture and helps the control engineer to automate the development process of distributed control applications. To be close with the latest trends in CASE tools development, the proposed ESS is a merging of a well-known general-purpose CASE tool and a custom Function Block design tool. The issue of portability is addressed, implementation details are considered and an example of using our tool is presented.**

***Index terms*-- ESS, IEC 61499, IPMCS, Function Blocks, CASE tool**

I. INTRODUCTION

Competition in the area of Industrial Process Measurement and Control Systems (IPMCSs) as well as today's rapidly changing market requirements imposes the need of improving the agility of manufacturing systems. Concepts like agile manufacturing and interoperability between products, devices, utilities and vendors, are partially addressed by proprietary solutions. Even more, the most of the traditional products and tools are far away from the new challenging technologies of Software Engineering. These technologies must be considered to provide the basis for the definition of new methodologies for the design and development of the always-growing complexity distributed IPMCSs.

Towards this direction the evolving standards, IEC-61499 and the IEC-61804 [1][2] define the basic concepts for the design of modular, re-usable, distributed industrial process, measurement and control systems. The function block construct is defined as the main building block of IPMCS applications, in a format that is independent of implementation. The above standards define also a methodology to be used by system designers to construct distributed control systems. It allows systems to be defined in terms of logically connected function blocks that run on different processing resources. Complete applications, can be built from networks of function blocks, formed by interconnecting their inputs and outputs.

New generation, function block oriented, Engineering Support Systems (ESS), are highly required to support the whole life cycle of IPMCS applications. ESS should support the design, implementation, commissioning and operation of IPMCSs constructed according to the architecture defined in Part 1 of IEC61499 Annex C1. Using such a tool, the engineer must be able to start with the analysis of the plant diagram so as to capture the control requirements. Then, he should be able to define the major areas of functionality and their interaction with the plant. During this task, he can exploit function blocks provided by intelligent field devices, such as smart valves, but also to assign functionality into physical resources such as PLCs, instruments and controllers. All the above should be accomplished independent of the underlying communication subsystem and in the extreme case, where it is an aggregation of interconnected independent fieldbus segments, even from different vendors.

Nowadays, the first ESSs that attempt to support the engineering phase of industrial processes by following the IEC61499 are appearing. The Function Block Development Kit (FBDK)[3] and the Verification Environment for Distributed Applications (VEDA) [4] are the most important tools today. FBDK is an effort of the Holonic Manufacturing Systems consortium to this direction. The tool is capable of defining Function Block types, and designing Function Block diagrams, resources and devices. These Function Block types and Function Block diagrams are described by means of XML as is specified in IEC61499. FBDK provides also a Java interface that lets the engineer to visually test his diagrams, but lacks the capability of downloading the Function Block types and distributing Function Blocks networks in real devices. Additionally, FBDK does not address the capture of requirements and is not mature enough to be used for the design or implementation of actual control systems. Additionally with FBDK, the Function Block RunTime Environment (FBRT) provides a runtime platform for testing of software tools for compliance with some of the configurability provisions of the Technical Agreement for IEC 61499 Feasibility Demonstrations. This platform may also be used for experimental and pre-commercial testing of devices with appropriate embedded Java implementations. VEDA mainly focuses on the modeling and verification of the Execution Control of Function Blocks following IEC61499.

The rest of this paper is organized as follows. In section 2 we briefly describe our CORFU development process. In section 3 we present our prototype Engineering Support System and the way our CORFU Function Block design tool interacts with a popular general-purpose CASE tool. In section 4 we discuss the compliance of the proposed tool with IEC 61499. In section 5 some implementation details are also discussed. In section 6 we present an example using our Engineering tool. We finally conclude the paper in the last section.

Elec. & Comp. Eng. Dept. , University of Patras,
265 00 Patras, Greece, tranoris@ee.upatras.gr

II. THE CORFU DEVELOPMENT PROCESS

The CORFU development process has been defined as a series of workflows, and is described in detail in [5]. This process is our attempt to ameliorate the development process defined by IEC61499 adopting best practices from component-based development, Object Technology and the Unified Modelling Language (UML) [6]. We next briefly describe this development process to make the paper self-contained.

For the "capture requirements" that is the first workflow, we have adopted the well-accepted use-case concept introduced by Ivar Jacobson [7]. During this workflow, control and field engineers properly define the use cases of the system, i.e., the responses of the system to external events that originate either from devices or humans. During the next workflow, namely the "Capture Behaviour", engineers cope with the examination of the dynamic behaviour of the system. Object Interaction Diagrams are considered as the first realization of system's use cases and are used to show the system's internal objects and the way they collaborate to provide the required behaviour. During the subsequent "capture static view" workflow, engineers deal with the design of the static view of the system in terms of class diagrams. Since the diagrams produced through the above process must be consistent, the engineer has to go back and forth through the workflows, in order to better specify the analysis and early design models of the system. As soon as the above models have been defined, the engineer is ready to move to FB design diagrams. A set of transformation rules were defined to automatically transform the UML-based system model to a FB-based design model that is better understood by control engineers. "Refinement and evaluation," "model-verification," and "FB-distribution" are among the main workflows that complement the development process.

III. A PROTOTYPE ENGINEERING SUPPORT SYSTEM

As far as the development of our prototype ESS, the development from scratch was considered as waste of time. Such an approach could make the development of the ESS much more complicated and there is a danger to lose our focus from the actual problem. Existing CASE tools that support the UML notation may be used to elaborate to modern ESSs. The most of the modern commercially available CASE tools support the UML notation and a lot of this know how can be successfully utilized for the development of our ESS. We next came to the dilemma to adopt an open source CASE tool to form the basis for the development or to base the development of our ESS on a commercially available OO CASE tool. For the first case, we had to consider the Argo UML open source CASE tool that supports the UML notation and is entirely written in Java. However, we made the choice to build on a commercially available CASE tool although the use of an open source CASE Tool provides more flexibility, but at the same time longer development time.

Our ESS is able to interact with a general-purpose CASE tool in order to ease and automate our development process and give the possibility to the engineer to smoothly follow the workflows of our process from the analysis of the system to the design with Function Block diagrams. Thus, the CASE tool will be used to support the workflow of Capturing the requirements, the workflow of capturing behaviour and the workflow of capturing system's static view. In a subsequent step, the CORFU FB design tool imports automatically the above information, applies the transformation rules and supports the workflow of designing FB diagrams.

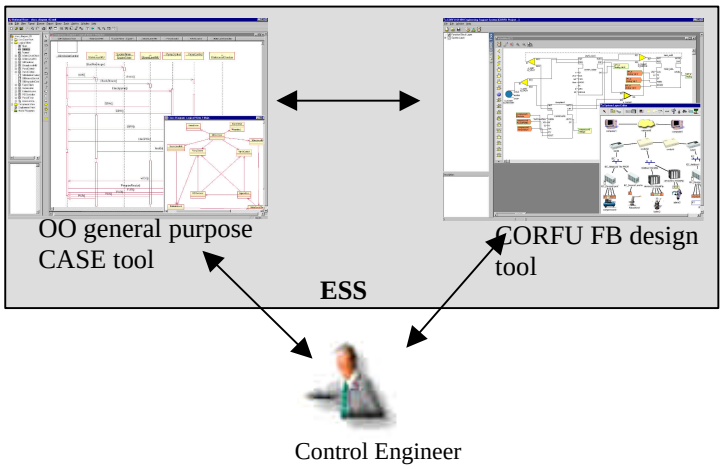


Figure 1. The Control Engineer Interacting with the ESS

Figure 1 shows the above concept. The control engineer interacts with the general-purpose CASE tool and the FB design tool during the process of engineering his IPMCS application. Thus, the CORFU ESS consists of two subsystems that co-operate and exchange information: A CASE tool, which in our case is Rational's Rose and the CORFU FB design tool.

A. *Using a general purpose CASE tool*

We examined several CASE tools and finally we selected Rational's Rose, because it comes with a suite of specific tools for software engineering, to cover the needs from requirements capture through the final implementation of software systems, it has several extension mechanisms and it is widely used. Rose supports two mechanisms and can be extended either with its custom scripting language, or can be used as a COM automation server [9] through a type library provided by Rational. We exploited and made use of both extensions mechanisms in the following way:

1) *Extending Rose through its scripting language*

We have used scripting language to extend Rose's toolbars in order to ease the design of the diagrams. While the engineer designs the diagrams in Rose, it is essential the way that he assigns types and stereotypes. It is assumed that the engineer is familiar, uses and declares properly, our provided stereotypes when designing class and interaction diagrams. Such stereotypes are for example the Function Block stereotype and the Industrial Process Terminator stereotype. Thus, by extending Rose's toolbars and forms, the engineer can use the stereotypes with a few clicks.

The Industrial Process Terminator (IPT) construct is used to represent in the design space the industrial process entities that are monitored or controlled by the application. IPTs are inserted into the design space of the system layer, to properly define the interaction of the control system with the

plant. IPT instances must directly be mapped to the actual devices that interface the IPMCS with the controlled processes of the industrial environment. Each IPT has a number of Industrial Process Parameters (IPPs) which are the inputs and the outputs of the control application.

2) Using Rose's automation interface

We used the automation interface to read the internal ROSE object model i.e. its classes, properties and diagrams. Additionally, the automation interface lets the programmer to manage (create, edit, delete) externally, classes and diagrams, thus giving the possibility to enable in our extension tool round-trip engineering capabilities. The automation interface is mainly used during the phase of applying the Transformation Rules of the CORFU development process.

B. Main components of the CORFU-ESS

The CORFU FB design tool, consists of the following components:

- a FB type editor
- a FB type library
- a FB diagram editor
- a System Layer editor
- a Transformation Facility Manager

We next describe these components.

1) The FB type editor

For the construction of FB diagrams, the engineer should be able to create instances not only from pre-defined FB types, but also from custom types. So, a FB type editor is provided with the tool, in order the engineer to be able to expand the existing FB type library. In Fig. 2 where a screen of the FB editor is given, the tree-structure and the XML specification of a FB type are shown. Both IEC specifications i.e. the textual and the graphical one are also supported. The interface shown in figure 3 is used by the control engineer to add new functionality and edit the properties of the FB type. Options such as "Test", "Compile" and "FB composition" complement the functionality of the FB editor.

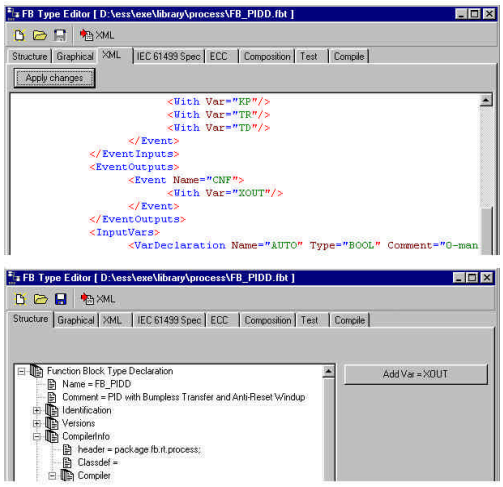


Figure 2. The FB type editor

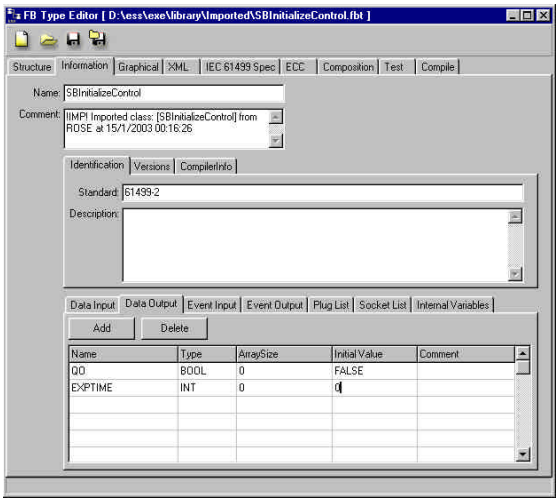


Figure 3 Creating a new FB type

2) FB type library

The FB type library is used as a repository of predefined FB types. New FB types are constructed and existing ones are modified using the FB type editor. Since we support the IEC61499 XML specification a utility to import FB types defined by other vendors is also provided. This utility was used to import in our tool all the FB types defined in FBDK.

3) The FB diagram editor

The FB diagram editor (see figure 4) enables the engineer to construct and refine the FB design diagrams of his control application. Using the FB editor the engineer has two alternatives:

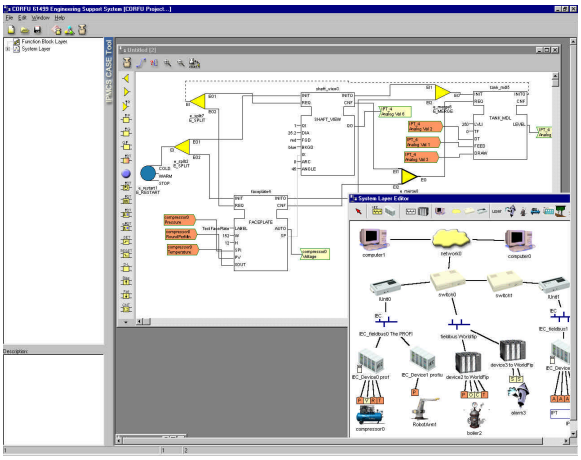


Figure 4. Our prototype CORFU FB design tool

a) He can automatically create from the already analysis model created in ROSE the first draft FB design diagrams. These diagrams are then refined to move into the specific implementation. The Transformation Facility Manager (TFM) utility of the tool, that is described following is used. He can import FB diagrams from other tools, such as FBDK, if the diagrams are compliant with the XML defined in IEC 61499.

b) He can design from scratch the FB diagram. The engineer can pick from available toolbars FB instances from pre-defined FB types in our library and design connections between FB instances or between FBs and IPPs. Additionally the engineer can import FB diagrams from

other tools, such as FBDK, if the diagrams are compliant with the XML defined in IEC 61499. Event FBs are supported in a special manner and depicted with our chosen format. Such FBs are: “event splitter”, “event merger”, “event rendezvous”, etc. The tool does not support the internal implementation of these FBs; instead an event-API has been defined that should be provided by an IEC-compliant device[10]. The tool automatically configures the device during the downloading process, through the configuration event-API, to provide the required behaviour during the operational phase. This approach results in simplified FB diagrams and improves the performance of the corresponding control application. Later on, the diagram can be exported as an IEC61499 compliant XML file and passed for testing or validation to another tool like VEDA.

4) The System Layer editor

The graphical System Layer editor, which is based on our 4-Layer architecture [11], is provided to automate the process of distribution of the control applications as well as its configuration and re-configuration. The engineer selects and configures available constructs of the system layer such as IEC compliant devices and fieldbuses, InterworkingUnits, IPPs and draws connections between them. Figure 5 for example shows the form where the engineer can edit the properties of an IEC compliant device. The system layer editor should fully support the process that we defined in [10].

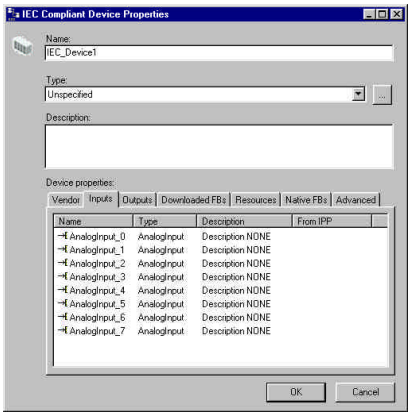


Figure 5. Properties of an IEC compliant device

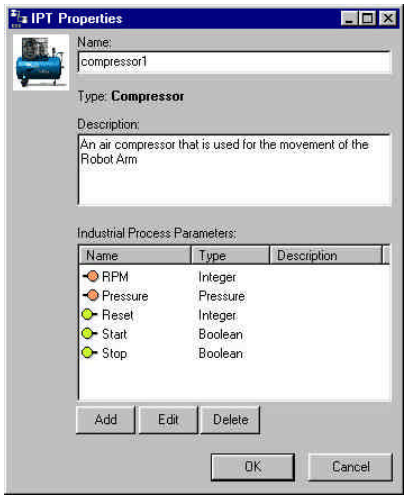


Figure 6. Properties of an Industrial Process Terminator

Figure 6, displays the properties of an Industrial Process Terminator. The user can add to the IPT, several Industrial Process Parameters, either sensors or actuators, that the industrial applications needs in order to solve the process.

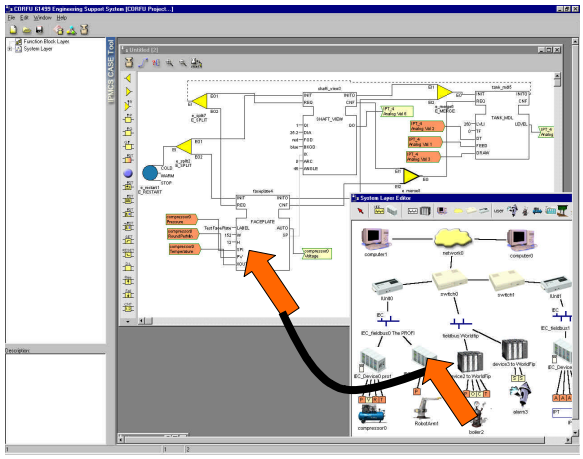


Figure 7. Drag and drop an FB instance to an IEC device

For the distribution of the control application to the system layer implementation system the user has to drag FB instances from the FB diagram editor and drop them to System layer devices either IEC-compliant or non IEC-compliant. The tool will automatically examine if the device supports the FB type of the downloaded instance. If the FB type is supported a new instance command is send to the device. Otherwise the FB type is first downloaded to the device. Additionally, the tool recognizes that the dragged FBs will be loaded in a device and makes available in the FB diagram editor all the IPPs that are connected to this device. After this the engineer can connect IPP sensors to FB data input and IPP actuators to FB data outputs.

5) The Transformation Facility Manager

Finally, in order to automate the transformation process, we have implemented in the ESS tool the Transformation Facility Manager (TFM) utility. TFM is a core utility of our tool since it incorporates and follows the transformation rules, reports the process and guides the engineer during the transformation. TFM is the utility that implements the interfacing with Rose and it's responsible for the proper creation of new types, events, etc from the analysis model in Rose as presented in the next section. The most important task of the TFM is the creation of new FB types in the ESS, by properly parsing and transforming the class and interaction diagrams from Rose. Figure 8 shows the form where the user interacts with the TFM utility.

Rose allows us to use it as a COM server, through its type library as mentioned. Type libraries provide a way to get more type information about an object than can be determined from an object’s interface. The type information contained in type libraries provides needed information about objects and their interfaces, such as what interfaces exist on what objects, what member functions exist on each interface, and what arguments those functions require.

Figure 9 shows on the left-hand side the class ValveControl with some methods like OpenValve(), CloseValve() and Start() as it appears in the CASE tool. The right–hand side, shows the equivalent Function Block that is created from the FB prototype tool when it applies our proposed transformation rules. The event inputs and outputs of the Function Block have been designed automatically. For example the input events Start() and OpenValve() on the Function Block, are extracted from the message exchange of the objects on an interaction diagram while a use case is realized.

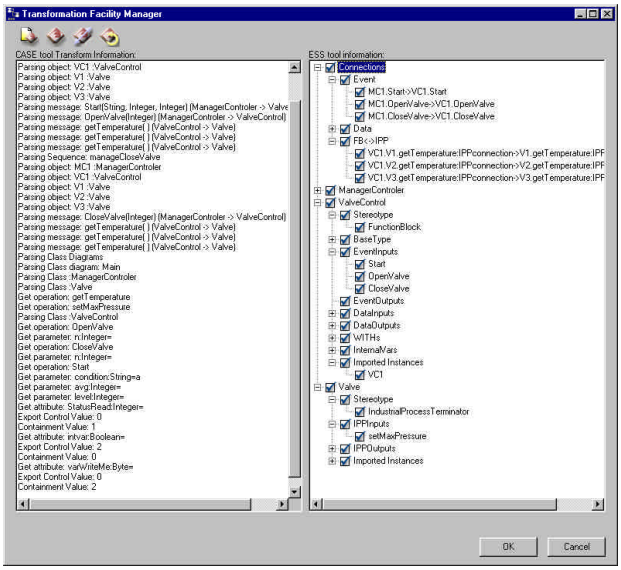


Figure 8. The Transformation Facility Manager utility

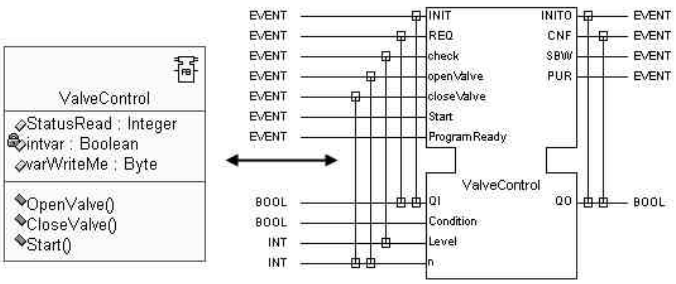


Figure 9. Class to Function Block

IV. IEC61499 COMPLIANCE

The implementation of our tool complies with the portability agreement proposed in IEC 61499 Industry Technical Agreement. So our tool is capable of:

- producing library elements like data types, function block types, resource types, device types, function block diagrams, system configurations, etc using the syntax and the semantics defined in Annex A of IEC 61499-2
- correctly parsing and interpreting elements in the XML DTDs, which are defined in Annex A of IEC 61499-2
- utilizing files for the exchange of library elements. For example the tool supports certain file types for element exchange like .fbt for Function Block types, etc.

With the above provided interfaces, the tool is capable to browse in the diagram. In order to check the portability of our tool, we have managed to exchange library elements such us Function Block types and diagrams, with the FBDK mentioned above. Additionally, FBDK gives us the ability to test via the Java library that it provides, our Function Block design diagrams.

V. IMPLEMENTATION DETAILS

The CORFU FB design tool was developed entirely from scratch with Borland's Delphi. For the XML processing we used Microsoft's XML 4.0 parser SDK, in conjunction with Delphi's XML components. Additionally, Delphi's XML binding tools helped us to create the necessary interfaces for correctly parsing XML files according to the IEC 61499 standard. In section 3.1 we discussed that we used the type library of Rose in order to access its object model. So, the first step was to import the type library in Delphi and have access to the interface of Rose automation. Delphi provides a type library editor and a tool that translates the type library file to Delphi constructs and creates the equivalent dispatch interfaces. Dispatch interface declarations are used to describe the methods and properties a COM Automation object implements through its IDispatch interface. After importing the type library, we have full access through our programming language to all the interfaces and Rose 's object model. Rose is active and hidden in the background, with our project opened.

The first step is actually to create the application, so we invoke a certain call to class factory's CreateInstance method, which (through windows API) creates the instance of the CoClass and thus the application. An example is the following extract from the code (in Delphi):

fRoseApp:= CoRoseApplication.Create;

where

fRoseApp variable is an instance of the *IRoseApplication* interface and is used as the interface to Rose.

Then we can load our model with

theModel:=

fRoseApp.OpenModel('C:\test\class_diagram_01.mdl');

and get for example from the diagram categories, the Scenario Diagrams with

theRC := theModel.RootCategory ;

theSD := theRC.ScenarioDiagrams;

where *theRC* is type of *IRoseCategory*

and *theSD* is type of *IroseScenarioDiagram Collection*

Other useful interfaces are *IRoseObjectInstance* which provides access to all object instances on a diagram, *IRoseMessageCollection* which provides access to the collection of messages in a scenario diagram and the *IRoseMessage* which provides access to individual messages. Rose normally is active and hidden in the background. Another implementation issue was the description of the distribution of Function Blocks in devices of the System Layer. Since IEC 61499 does not standardize any syntax on system distribution we defined our own syntax for describing system distribution and FB assignment. We prepared for this an XML schema definition which fully describes the syntax concerning the distribution on the System Layer. Actually this XML syntax with the IEC FB diagrams XML specification is embedded into a file describing the CORFU project.

VI. USING THE CORFU-ESS

In order to examine how our tool can be applied in the development process of a control application we have considered the steam boiler case study [12]. The steam-boiler control specification problem has been proposed for testing the various design formalisms with respect to their versatility and applicability to embedded control system design. The specification in brief, concerns a control application that serves to control the level of water within some safe range in a steam boiler. The physical environment comprises the following units: A water tank which generates steam, a water sensor to measure the quantity of water in the steam boiler, four pumps to provide the steam boiler with water and four devices to supervise the pumps (one controller for each pump) and a steam sensor to measure the quantity of steam which comes out of the steam boiler.

Following our CORFU development process, during the first workflow of capturing the requirements, we created a set of Use cases describing the requirements and the actions of the system. Figure 10 shows the use cases and the actors of the system that we have identified. Some of them are “System setup”, “Handle Water Level Measurement”, etc.

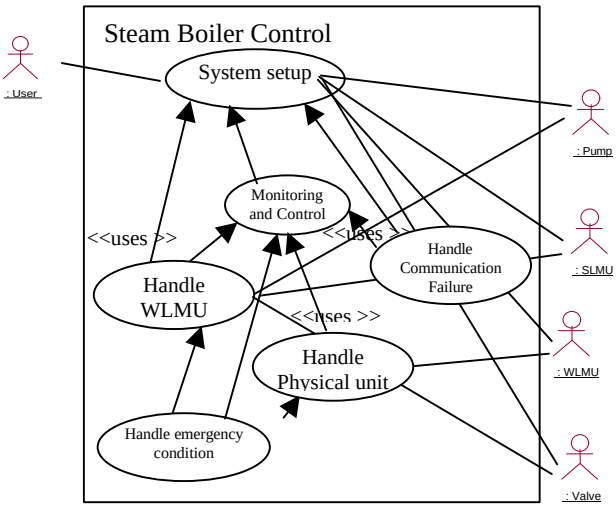


Figure 10. Steam Boiler Control Use case diagram

Next we proceed in the next workflow of capturing system’s behavior. For each use case we have implemented an Interaction Diagram such the one shown in figure 11 from Rose, that implements the Use Case Close Valves. Additionally, in the workflow of capturing system’s static view we created a class diagram describing the system.

Next, in order to move to the design workflow, we used the TFM utility in the CORFU FB tool, which translated every interaction diagram to an equivalent Function Block diagram. We made several iterations through this process, until coming to a satisfactory level of detail.

Finally, all the Function Block diagrams are merged together. The resulting function block diagram is shown in figure 12.We made here all the proper connections and eliminated any redundancy of Function Blocks among the diagrams.

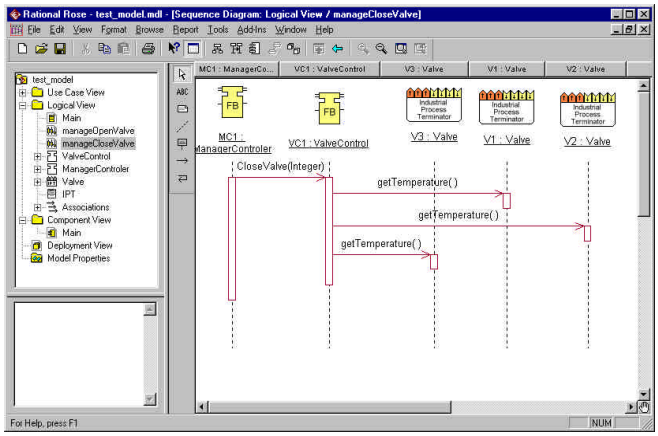
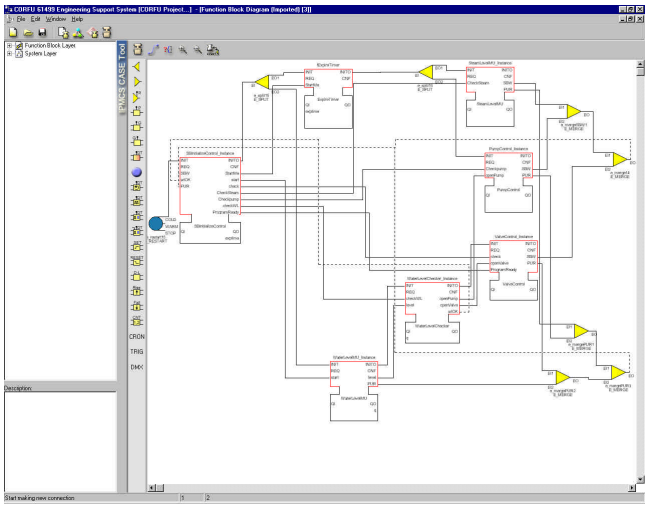


Figure 11. Steam Boiler Control Use case diagram



12.The resulting Function Block diagram

VII. CONCLUSIONS

In order to enhance the requirements capturing, the system analysis and the transition to the design phase of IPMCSs, we have defined a new development process towards a unified design methodology. Our development process adopts use cases and the UML notation to capture requirements of the IPMCS applications. It uses, the interaction diagrams for the first realization of use cases and a set of well-defined transformation rules for the subsequent evolution of requirements to Function Block Diagrams.

In this paper we have presented our prototype ESS tool that consists of two subsystems: Rose, a popular general-purpose CASE tool and the CORFU FB design tool. Our CORFU FB tool is capable of communicating with the CASE tool and it can follow specific proposed transformation rules, for mapping interaction and class diagrams to equivalent function block diagrams. By means of these subsystems, the engineer designs his diagrams on the CASE tool and our prototype FB design tool imports them automatically. Then he proceeds with the design of his application with Function Blocks, which are closer to the final implementation. We have presented also implementation details of our tool and how it complies with the IEC 61499 Industry Technical Agreement.

In our prototype tool still some missing functionality exist and further improvement is needed such as:

- ECC and test editor
- feasibility demonstration with FBRT
- the ability to download the Function Block types and Function Blocks networks to the interworking units and create all the Function Block data and event connections
- applicability of an example IPMCS application with remote configuration of personal computers acting as IEC devices
- examining the possible integration with the VEDA tool
- further operations that will help more the engineer during the use of our CORFU development process.

VIII. REFERENCES

[1] IEC Technical Committee TC65/WG6, "IEC61499 Industrial Process Measurement and Control – Specification", IEC Draft 2000

[2] IEC sub committee no. 65c: digital communications, working group 7: function blocks for process control, "IEC1804 General Requirements", IEC Draft 1999

[3] The Holobloc webpage, http://www.holobloc.com

[4] at.iw.uni-halle.de/~valeriy/project/proj_descr.htm

[5] K. Thramboulidis, "Development of Distributed Industrial Control Applications: The CORFU Framework", 4th IEEE International Workshop on Factory Communication Systems, Sweden, August 2002

[6] OMG UML specification version 1.3, March 2000

[7] I. Jacobson, Object-Oriented Software Engineering: A use-case driven approach, Addison Wesley 1992.

[8] Ivar Jacobson et.al. "The Unified Software Development Process", Addison, Wesley 2000 Ch.2 p.26

[9] www.microsoft.com/com/tech/dcom.asp

[10] K. Thramboulidis, "Towards an Engineering Tool for the Development of Distributed Control Systems", submitted

[11] Developing a CASE tool for distributed Control Applications, submitted to The International Journal of Advance Manufacturing Technology, Springer-Verlag.

[12] J.- R. Abrial, "Steam-boiler control specification problem", August 10, 1994., http://www.Informatik .unikiel.de /~procos /dag9523