An IEC-compliant Engineering Tool for Distributed Control Applications

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

***Abstract*— To address the requirement of modern manufacturing plants to quickly respond to market demands 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, which is based on our enhanced 4-layer architecture, automates the development process of distributed control applications. To be close with the latest trends in the development of CASE tools, we integrated a well-known general-purpose CASE tool with a custom Function Block development tool. The resulting ESS is described, the issue of portability is addressed and implementation details are considered.**

***Index terms*-- ESS, IEC-61499, IPMCS, Function Block, CASE tool, Engineering tool.**

I. INTRODUCTION

Competition in the area of Industrial Process Measurement and Control Systems (IPMCSs) as well as today's rapidly changing market requirements impose 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.

To address these problems the evolving standards, IEC-61499 and the IEC-61804 [1][2] define the basic concepts for the design of modular, re-usable, distributed IPMCSs. The Function Block (FB) 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 FBs that run on different processing resources. Complete applications, can be built from networks of FBs, formed by interconnecting their inputs and outputs. New generation, FB-oriented, Engineering Support Systems (ESSs), are highly required to support the whole life cycle of IPMCS applications. These ESSs should support the design, implementation, commissioning and operation of IPMCSs that are constructed according to the architecture defined in Annex C1 of the IEC61499-Part 1. 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 FBs provided by intelligent field devices, such as smart valves, but also to assign functionality into physical resources such as PLCs, instruments and controllers.

In our attempt to implement the IEC model, we have defined and have under development the CORFU framework, that is a Common Object-oriented Real-time Framework for the Unified (CORFU) development of distributed IPMCS applications [13]. We have also defined in [5] a process for the development of distributed IPMCSs. Since the analysis phase is slightly addresses by the IEC standards we have adopted the widely accepted use case driven approach of Ivar Jacobson [7] and the UML [6] notation to capture the requirements of the IPMCS system. The proposed process also defines the evolution of these requirements through a set of well-defined transformation rules to Function Block design diagrams.

In this paper we present the CORFU ESS, i.e. a prototype ESS that intends to automate the development process of IEC-compliant distributed IPMCSs. This ESS is based on the enhanced 4-layer architecture that is defined in [11] and also implements the extensions and modifications to the IEC-model that are proposed in [10]. To exploit the latest trends in the development of CASE tools, we decided to utilize a commercially available general-purpose CASE tool and integrate it with the CORFU Function Block Development Kit (CORFU-FBDK). The general-purpose CASE tool is used to support the early phases of the development process while the CORFU FBDK supports the remaining phases by automatically importing the diagrams produced by the general-purpose CASE tool applying a set of well defined transformation rules.

Nowadays, the first ESSs that attempt to support the engineering phase of industrial processes by following the IEC61499 are under development. The Function Block Development Kit (FBDK) [3] and the Verification Environment for Distributed Applications (VEDA) [4] are the most important tools today. The FBDK, which is an effort of the Holonic Manufacturing Systems consortium, allows the definition of Function Block types, and the design of Function Block diagrams. Function Block types and Function Block diagrams are described by means of XML as is specified by the IEC61499. A Java interface lets the engineer to visually test his diagrams. However, FBDK does not address the capture of requirements and lacks the capability of downloading the Function Block types and distributing Function Blocks networks in filed devices and field buses. VEDA on the other hand mainly focuses on the modeling and verification of the Execution Control of Function Blocks following the IEC61499 model.

The rest of this paper is organized as follows. In section 2 we briefly describe our CORFU development process. In

Electrical & Computer Engineering, University of Patras,
265 00 Patras, Greece, thrambo@ee.upatras.gr

section 3 we present our prototype ESS. We describe the CORFU FBDK and the way that this tool exploits a popular general-purpose CASE tool to elaborate to an IEC-compliant Engineering tool. In section 4 we discuss the compliance of the CORFU ESS with the IEC model. Implementation issues are discussed in section 5 and we finally conclude the paper in the last section.

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 Modeling Language (UML). 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 Behavior", engineers cope with the examination of the dynamic behavior 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 behavior. 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

To automate the above-described development process an Engineering tool is highly required. In our attempt to design and implement such a tool we considered the development from scratch as a 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. Since we have adopted the use case concept and the UML notation, existing CASE tools that support the UML notation may be used to elaborate to modern ESSs. 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.

However, to support a) the transformation of UML requirements to Function Block design specifications and b) the remaining workflows of our development process, we defined the architecture of the CORFU FBDK and we have already developed a first prototype. The CORFU FBDK is able to interact with a general-purpose CASE tool in order to ease and automate our development process. The general-purpose CASE tool will be used to support the "capture requirements", "capture behavior" and "capture static view" workflows. In a subsequent step, the CORFU FBDK imports automatically the diagrams produced by the above workflows, applies the transformation rules, and supports the remaining workflows of our development process.

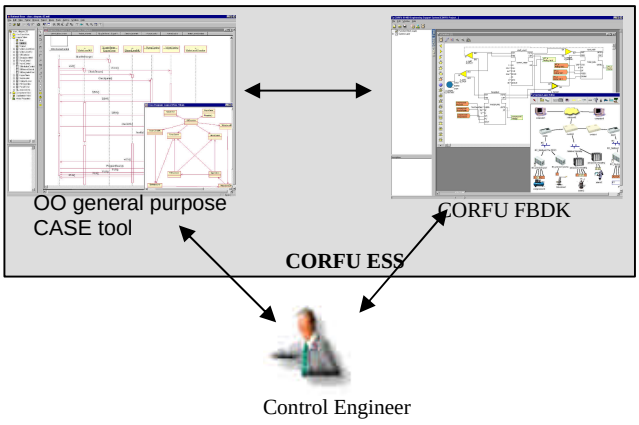

Figure 1. The Control Engineer interacting with the CORFU ESS.

Thus, the CORFU ESS consists of two subsystems as is shown in Figure 1: A general-purpose CASE tool, which in our case is Rational's Rose and the CORFU FBDK. The control engineer interacts both with the general-purpose CASE tool and the CORFU FBDK during the process of engineering his IPMCS application.

A. Using a general purpose CASE tool

We examined several CASE tools and finally we selected Rational's Rose, because a) 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, b) it has several extension mechanisms and c) it is widely used. Rose supports two extension 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 that is provided by Rational. We have exploited and made use of these extension mechanisms to properly support the development process.

1) Extending Rose through its scripting language

While the engineer designs the diagrams in Rose, it is assumed that he is familiar, uses and declares properly, provided stereotypes when designing class and interaction diagrams. Such stereotypes are for example the Function Block stereotype and the Industrial Process Terminator (IPT) stereotype. In order to simplify the design of object interaction and class diagrams, we have extended Rose's

toolbars using scripting language. , The engineer can use the application specific stereotypes with just a few clicks.

The 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 control application.

2) Using Rose's automation interface

In order to enhance the communication with Rose, we used the automation interface to read the internal Rose object model i.e. its classes, properties and diagrams. Additionally, the automation interface allows 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 from the Transformation Facility Manager presented below, during the phase of applying the Transformation Rules of the CORFU development process.

B. Main components of the CORFU FBDK

The CORFU FBDK consists of the following components:

1. a FB type library
2. a FB type editor
3. a FB network editor
4. a System Layer (SL) editor
5. a Transformation Facility Manager (TFM)

We next describe these components as well as the way they are used in the development process.

1) FB type library

The FB type library is the repository of FB types of the CORFU FBDK. A number of predefined FB is already contained in this library. A utility to import FB types defined by other vendors using the IEC61499 XML specification has been developed. This utility has already been used to import in our ESS all the FB types defined in FBDK. FB types already contained in the CORFU FBDK are: E_RESTART, E_DELAY, FB_ADD_INT, PID_PRE, DERIVATIVE, TANK_MDL, etc. Moreover new FB types may be constructed and existing ones modified using the FB type editor.

2) The FB type editor

The FB type editor allows the control engineer to modify existing FB types of the FB type library as well as to create new types to satisfy the specific requirements of the control application. For the construction of a new FB type the engineer has to select between using the default primitive FB template (New basic)[1], the default composite FB template (New composite), or select any other FB from the FB type library to be used as template (New, based on...).

[1] refers to the corresponding main menu entry.

The graphical representation of the selected template appears in the FB editor as is shown in figure 2. The engineer extends the functionality of the template using the form shown in figure 2. To obtain more flexibility in the FB type editor we selected to support both the IEC specifications i.e. the textual and the graphical one. In Fig. 3 two screens of the FB type editor are given to show the tree-structure and the XML specification of a FB type. An ECC graphical editor allows the definition of the Execution Control Chart of the FB type and an Algorithm editor allows the definition of the FB's algorithms. Options such as "Import", "Export", "Test", "Compile" and "Composition" complement the functionality of the FB type editor.

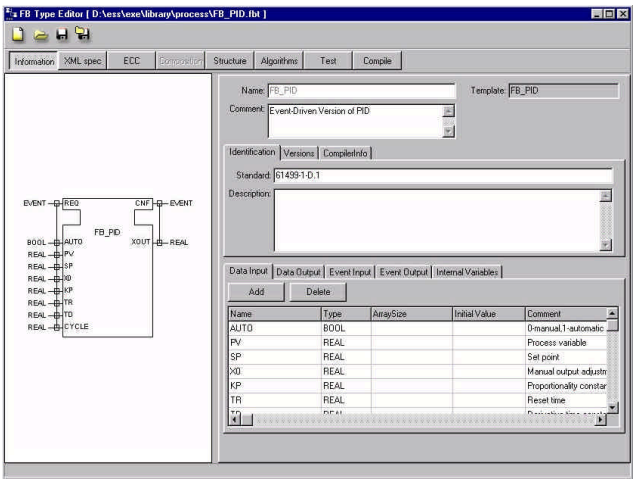

Figure 2. Editing a FB type in CORFU FBDK

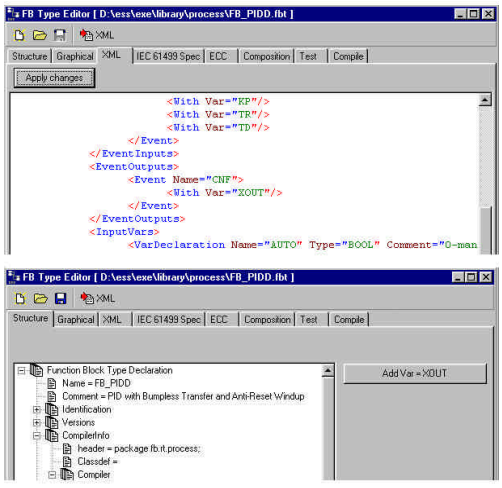

Figure 3. XML and structured specification in the FB type editor

3) The FB network editor

The FB network editor enables the engineer to graphically construct and refine the FB network diagrams of the control application. In figure 4 an example of a FB network diagram created with the FB network editor is given. For the construction of the FB network the engineer has the following options:

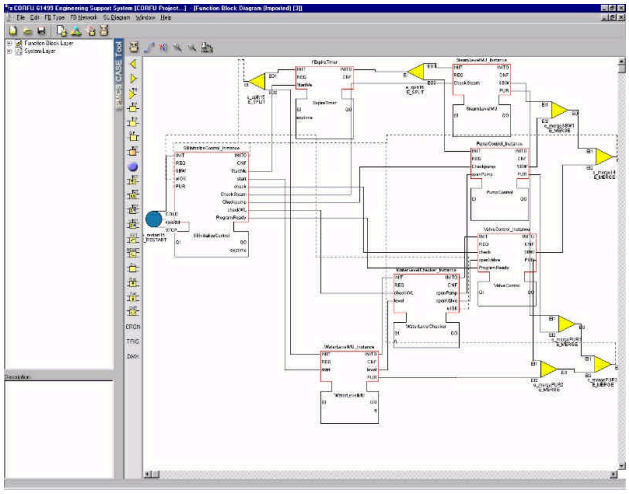

Figure 4. A sample FB network diagram.

a) Use the UML diagrams of the analysis model of the control application (New from UML). The analysis should have been already completed using the general-purpose CASE tool. The Transformation Facility Manager (TFM) is used by the CORFU FBDK to transform the UML design diagrams to corresponding FB networks. These diagrams are then refined to move into the specific implementation.

b) Import FB network diagrams (Import) from other tools, such as the FBDK. These diagrams should of course be compliant with the XML specification defined in IEC 61499.

c) Design the FB network from scratch (New). The engineer can i) insert in the diagram FB instances using the FB types of the library ii) insert IPPs from the SL diagram and iii) draw the connections between FB instances or between FB instances and IPPs.

The Event FBs of the IEC61499 model (i.e. “event splitter”, “event merger”, “event rendezvous”, etc.) are supported in a special manner and depicted with our chosen format as is described in [10]. We have defined a set of icons to represent the event FBs in a toolbar to simplify the FB network construction process. The event-API that should be provided by an IEC-compliant device has been defined. The CORFU FBDK utilizes this API to automatically configure the device, during the downloading process so as the device provide the required behavior during the operational phase. This approach improves the performance of the corresponding control application [10].

For the verification of the FB network we examine the possibility to export the FB network diagram in IEC61499 compliant XML specification. This specification can be used for testing or validation using other tools. VEDA is one of these tools.

4) The System Layer editor

The graphical System Layer editor is provided to automate the process of distribution of the control application as well as its configuration and re-configuration. This editor fully supports the design space of the system layer of our extended 4-layer architecture. The engineer selects and configures the constructs of the system layer i.e. the IEC-compliant devices and fieldbuses, the InterworkingUnits, and the IPPs, and draws connections between them. Figure 5 shows the form that is used by the engineer to edit the properties of an IEC-compliant device.

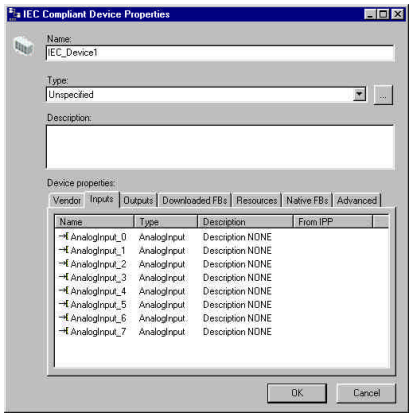

Figure 5. Editing the properties of an IEC-compliant device

Figure 6, displays the properties of an Industrial Process Terminator. The user can add to the IPT, the Industrial Process Parameters, i.e. sensors or actuators that are relevant to the specific industrial application.

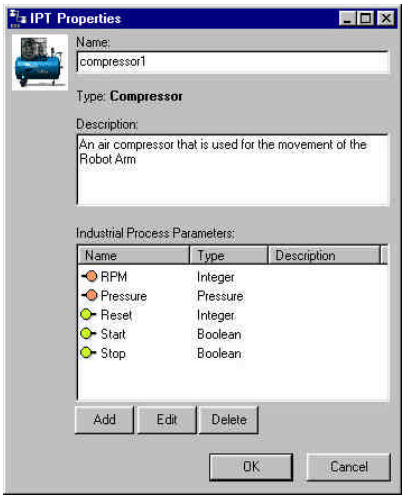

Figure 6. Editing the properties of an Industrial Process Terminator

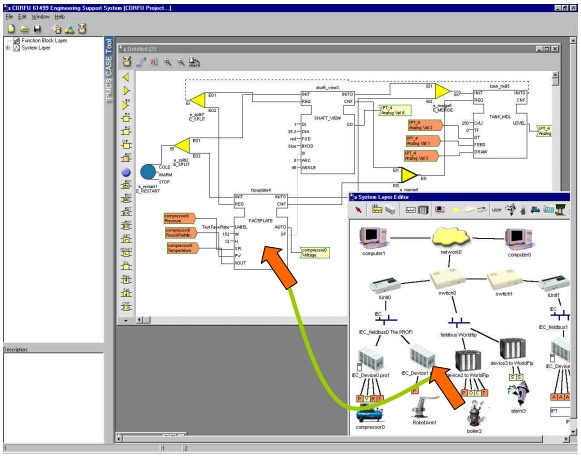

Figure 7. Assigning the FB instances to system layer devices.

Figure 7 shows that for the distribution of the control application to the filed devices of the system, the user has to drag FB instances from the FB network editor and drop them to System layer devices either IEC-compliant or non IEC-compliant. The CORFU FBDK automatically examines if the device supports the FB type of the downloaded instance. If the FB type is not supported the FB type is downloaded to the device. A "new instance" command is next issued to the device. 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. This greatly facilitates the process of connecting IPP sensors to FB data input and IPP actuators to FB data outputs.

5) The Transformation Facility Manager

To automate the transformation process of UML diagrams to FB network diagrams, we have designed and implemented in the CORFU FBDK the Transformation Facility Manager (TFM). TFM is a core utility of our tool since it incorporates and applies the transformation rules, informs and guides the engineer during the transformation process. TFM implements all the interface of the CORFU FBDK with Rose. It is responsible for the creation of the proper new types, events, etc from the analysis model in Rose. The most important task of the TFM is the creation of new FB types in the CORFU FBDK, by properly parsing and transforming the class and interaction diagrams from Rose. Figure 8 shows a sample interaction diagram that we have used with TFM. This diagram is extracted from an Object Oriented solution we gave for the steam boiler case study [12]. We have used extensively this case study in conjunction with our tool, in order to examine how the tool can be applied in the development process of a control application.

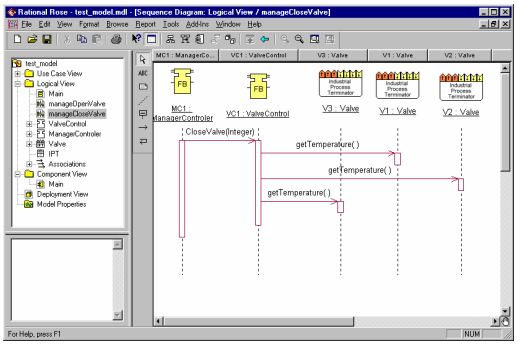

Figure 8. Sample interaction diagram produced with Rose

Rose is used as a COM server, through its type library. 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 the required information for the 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 TFM 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, were extracted from the message exchange between the objects of the corresponding interaction diagram that realizes the use case.

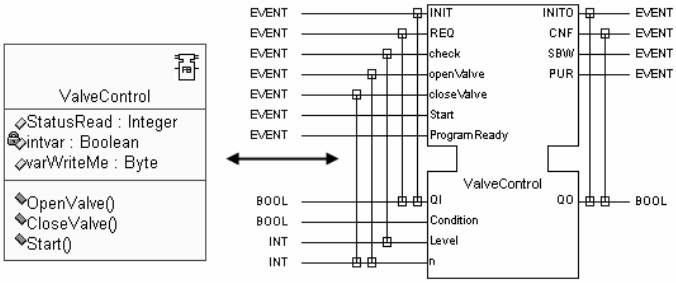

Figure 9. Class to Function Block type mapping

IV. IEC61499 COMPLIANCE

The prototype CORFU FBDK complies with the portability agreement proposed in IEC 61499 Industry Technical Agreement. The 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. The tool for example supports certain file types for element exchange like .fbt for Function Block types, etc.

In order to check the portability of our tool, we have managed to exchange library elements such as Function Block types and diagrams, with the FBDK. We are also using FBDK to test via the Java library that it provides, our Function Block design diagrams.

V. IMPLEMENTATION DETAILS

The CORFU FBDK was developed entirely from scratch with the Borland's Delphi IDE. 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. As we have already noticed we use 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*
Then, we access the scenario diagrams from Rose, as the following code fragment shows:

```
for i:=0 to theSD.Count-1 do
  begin
    vCurrentSD := theSD.GetAt(i+1);
    vCurrentSDName := theSD.GetAt(i+1).Name;
    vInstanceCollection := theSD.GetObjects;
    vSDMessagesCollection:= theSD.GetMessages;
    vInstancesCnt := vInstanceCollection.Count;
    vFirstInstance:= vInstanceCollection.GetAt(i+1);
…..
```

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. 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 FBDK. The CORFU FBDK is capable of communicating with the CASE tool and it can follow specific proposed transformation rules, for transforming 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
- ? examining the possible integration with the VEDA tool

VII. 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] FBDK - The Function Block Development Kit, http://www.holobloc.com/fbdk/README.htm

[4] Modeling and Verification of Execution Control of Function Blocks (VEDA) http://at.iw.uni-halle.de/~valeriy/project/proj_descr.htm

[5] C. Tranoris, K. Thramboulidis, "From Requirements to Function Block Diagrams: A new Approach for the Design of Industrial Applications", 10th Mediterranean Conference on Control and Automation, 9-12 July, 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] Distributed Component Object Model (DCOM) - Downloads, Specifications, Samples, Papers, and Resources for Microsoft DCOM, http://www.microsoft.com/com/tech/dcom.asp

[10] K. Thramboulidis, “An Architecture to extend the IEC model for Distributed Control Applications” submitted.

[11] K. Thramboulidis, C. Tranoris, “Developing a CASE tool for Distributed Control Applications", The International Journal of Advanced Manufacturing Technology, Springer-Verlag (forthcoming).

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

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