A Function Block Based Approach for the Development of Distributed IPMCS Applications

K. Thramboulidis, *Member IEEE,* C. Tranoris, *Member IEEE*

***Abstract*- Today's rapidly changing market requirements impose the need of improving the agility of manufacturing systems. The function block concept, promises to improve productivity in terms of re-use, reliability, flexibility and interoperability. IEC, with standards like 61499, defines the basic concepts and a methodology, for the design of modular, re-usable distributed Industrial Process Measurement and Control Systems (IPMCSs). In this paper we present our approach for the development of function block oriented distributed IPMCS applications. This approach is based on a 4-layer architecture that was defined to facilitate the design and development of the new generation function block oriented Engineering Support Systems (ESSs). The approach greatly exploits the modular architecture of the interworking unit that is used to interconnect different types of fieldbus segments. We adopt the steam boiler system as a case study and we consider the development of a function block based IPMCS application for the control of it.**

***Index Terms*--Fieldbus, Interoperability, Interconnection, distributed IPMCS, Profibus**

I. INTRODUCTION

Today's rapidly changing market requirements impose the need to improve the agility of manufacturing systems. The always growing need for innovative products, forces manufacturing plants to improve their ability to quickly respond to market demands by designing competitive products and modifying existing ones. Until recently, most of the industrial control systems have been based either on traditional distributed control systems or on programmable logic controllers. In both cases, the systems are composed of monolithic applications that are almost impossible to integrate and even to expand. Modularity, flexibility, extensibility, reusability and interoperability are dimensions slightingly addressed by many traditional proprietary engineering tools and products. Even more, the most of the traditional products and tools are far away from the new challenging technologies of Software Engineering. The software industry of Industrial Process Measurement and Control Systems (IPMCSs) increasingly faces today, the challenge of creating complex custom-made distributed systems within time and budget, while high competition forces prices down.

Evolving standards, like IEC 61499 and the more recent IEC61804, define the basic concepts as well as a methodology for the design of modular, re-usable, distributed industrial process, measurement and control systems [1][2]. They define, the function block construct as the main building block of IPMCS applications, in a format that is independent of implementation. They also define, the way that function blocks can be used to define robust, re-usable software components that constitute the distributed IPMCSs.

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 network 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. An ESS must support the engineer, in both the analysis and design phase as well as in the implementation phase. Using such a system, 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 should be able, to 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 underline communication subsystem and in the extreme case, where it is an aggregation of interconnected independent fieldbus segments, even from different vendors.

In the context of this work, we consider the process of developing function block based IPMCS applications that must be distributed over heterogeneous fieldbuses. We exploit the architecture of the interworking unit that was defined to obtain interoperability in fieldbus level, to allow a uniform development of applications, independent of the underlying profibuses. This architecture enables the interworking unit to:

- Meet the real-time constraints imposed by the fieldbus-to-fieldbus communication channel
- Provide a uniform, independent of the underlying communication subsystems, interface to the ESS
- Provide a uniform interface to SCADA client systems
- Provide the interoperability mechanisms required by any communication channel in the IPMCS environment.

Our intention is to highlight the main steps an engineer has to follow for the development of distributed IPMCS applications using the function block construct. We use the steam boiler control system as a running example, and demonstrate the process of developing a distributed IPMCS application using the function block construct.

We assume the existence of an interoperable IEC 61499-compliant interworking unit that will provide the

Electrical & Computer Engineering Dept., University of Patras, 265 00 Patras, Greece, Tel: +30-61-997304; Fax: +30-61-997316 ,
e-mail: {tranoris, thrambo}@ee.upatras.gr

ability to utilize heterogeneous software environments, with IEC 61499-compliant software tools. We utilize our 4-layer IPMCS architecture, to capture the key abstractions, that will become the basis for the development of the function block oriented IPMCS. This architecture identifies the main layers of abstraction involved in the development of distributed IPMCS applications and provides the framework for the required assignments from layer to layer.

The rest of this paper is organized as follows. In section 2 of this paper, we briefly describe the steam boiler control problem. In section 3, we introduce the function block construct as the primary building block for the design of IPMCS applications. We next, in section 4, use the function block construct to build a draft function block interaction diagram of the steam boiler control application. In section 5, we proceed to the mapping of the resulting function block interaction diagram to the constructs of the appropriate system layer diagram. Section 6 refers to the architecture of the interworking unit and the way this architecture automates the mapping and configuration process of the IPMCS application. We finally conclude the paper in the last section.

II. THE STEAM-BOILER CONTROL SYSTEM

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. An informal specification of the original problem is presented in detail in [3]. The specification 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:

- the steam boiler: A water tank of total capacity C in liters, which generates steam.
- a water sensor: A device to measure the quantity q in liters of water in the steam boiler
- four pumps to provide the steam boiler with water. Each pump has its capacity P in liters/sec.
- four devices to supervise the pumps (one controller for each pump)
- a steam sensor: A device to measure the quantity õ in liters/sec of steam which comes out of the steam boiler.

There are two normal water quantity bounds N1 and N2 in liters, within which the water quantity should be maintained during regular operation. The plant provides the control application with information from various physical units and sensors, while the control application controls the plant based on the messages received from the plant. The control application communicates with the physical units i.e. the plant and the controllers through messages transmitted via communication channels. The time for the transmission can be neglected. The control application follows a cycle and a priori does not terminate. A sampling event occurs every five seconds. When it happens, the control application first receives messages coming from the physical units. Then the control application analyzes the information, which have been received and calculates the suitable pump control value p. Finally the calculated control value p is transmitted to the physical pump by the controller. The water quantity q, the steam output õ and the throughput of the pump p are continuous functions of time. The control application uses the corresponding sampling data.

We also assume that all messages coming from or going to the physical units are supposed to be received or emitted simultaneously by the program at each cycle and there is no communication failure during system operation.

There are two dangerous bounds for the water quantity in the boiler: The minimum limit quantity M1 and the maximum limit quantity M2. Below M1 the steam boiler would be in danger after five seconds, if the steam continued to come out at its maximum quantity without supply of water from the pumps. Above M2 the steam boiler would be in danger after five seconds, if the pumps continue to supply the steam boiler with water without possibility to evacuate the steam.

III. THE FUNCTION BLOCK APPROACH

The latest trends in the development of IPMCS are concentrated in the development of the evolving international standards IEC 61499 and IEC 61804. They both ameliorate the function block construct first introduced by the IEC 1131 standard on programming languages for programmable logic controllers, extending it to the new requirements of distributed control systems.

A function block may be primitive i.e. small enough to provide a solution to a small problem, such as the control of a valve, or composite, that is an aggregation of function blocks, i.e. big enough to control a major unit of plant such as the steam-boiler. The function block construct is an abstraction mechanism that allows industrial algorithms to be encapsulated in a form that can be readily understood and applied by industrial engineers who are not specialists in the implementation of complex algorithms. The functionality of the function block is provided by means of algorithms, which process inputs and internal data and generate output data. A function block consists of a head and a body. The head is connected to the event flows and the body to the data flows.

The function block concept, as is very well pointed out in [4], shares many of the well defined, and already widely acknowledged benefits of the object concepts introduced by the Object Technology. Objects are stable; they reduce the complexity of the system and are highly reusable. The function block shares these attributes and constitutes a well-established concept for defining robust, re-usable IPMCS software components.

The IEC 61499 further defines a general model and methodology for describing IPMCS applications in a form that is independent from a specific implementation. An application consists of one or more function block *instances*, interconnected by *event connections* and *data connections*. Process inputs and outputs define the interface of the application with the controlled industrial process. We have defined the Industrial Process Terminator (IPT) construct, to represent, in the function block diagram, the sources and sinks of the application's events and data.

IV. THE FUNCTION BLOCK DIAGRAM OF THE STEAM BOILER

To facilitate the development of the steam boiler IPMCS application we use the 4-layer architecture presented in [5]. The bottom layer, that is the real world layer, is the actual IPMCS system. It consists of the

controlled industrial processes, the fieldbus segments used to interconnect the field devices, the interworking units, the backbone, the enterprise intranet, the control room and the engineering workspace. For the development of an efficient, robust and reliable steam-boiler system, the higher layers of abstraction must be directly assigned to this layer and this assignment must be invisible to the engineer. The next layer, that is the system layer, includes the required abstractions, used by the ESS to support the function block's distribution and assignment process. It consists of appropriate abstract representations of field devices, fieldbuses, interworking units, the backbone communication subsystem and the related interconnections. These abstract representations are in the form of proxies of the actual real world objects in the developer's workspace. The appropriate system level editor allows the engineer to construct the system design and directly map it to the lower level. Device specification is an example of the means used to support such a direct mapping.

The third layer, that is called application layer, is used to represent the required constructs used in the design and implementation of the IPMCS applications. Function blocks, data and event connections, Industrial Process Terminators (IPTs), and industrial process parameters, are the key abstractions and correspondingly the main building blocks of the application layer diagrams that are constructed by means of an application layer editor. The IPT instances of the diagram, must directly be mapped into the actual devices that interface the IPMCS with the controlled processes of the industrial environment.

In this phase of the development process, we mainly concentrate in the application layer and we construct the function block diagram of the steam boiler control application. We assume that the function block types shown in figure 1 are available for the implementation of the system.

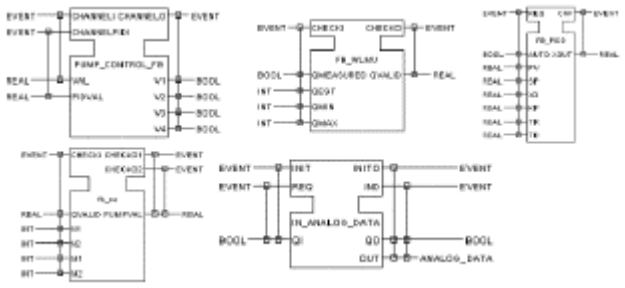

Figure 1. Function block types used in the steam boiler control application.

The PUMP_CONTROL_FB and IN_ANALOG_ DATA function block types implement a service interface for the pump control unit and the water level measurement unit, respectively. The FB_WLMU type checks for the following water level measurement unit failures:

a) if the unit indicates a value that is out of the valid static limits- i.e. between 0 and C and
b) if the unit indicates a value that is incompatible with the dynamics of the system that is expressed by Qest.

Qest is the estimation of water level produced by another function block. Qvalid is either Qmeasured or Qestim. The FB_SU function block type captures the behavior of the system in the case where the water level is above N1 or below N2. The FP_PID function block type represents a PID controller for maintaining the water level between N1 and N2.

Using instances of the above function block types, we proceed to the construction of the steam boiler function block interaction diagram. Figure 2 represents part of a draft function block interaction diagram that is mainly used to highlight the distribution process of the function blocks and the assignments required to take place in the interworking units. The ANALOG_INPUT function block instance has an event connection with CLK. When the ANALOG_INPUT accepts the event REQ it reads the digital value q of the water level and transforms the value to an IEC 61499 type. This value is read by the WATER_LEVEL_MU through the data connection Q0-QMEASURED and is compared with the estimated water value QEST for it's correctness. If the measured value is valid, the SUPERVISION function block reads it, and checks if it is in the range N1-N2.

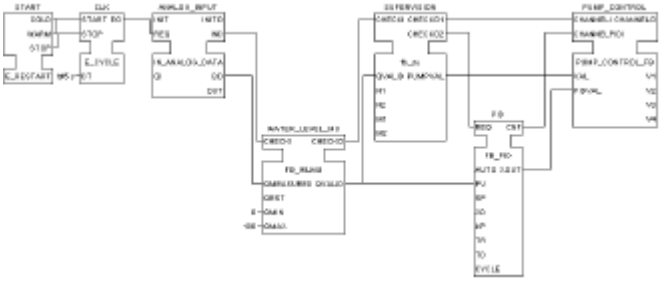

Figure 2. Part of a draft function block interaction diagram of the steam boiler control application.

The PUMP_CONTROL function block is connected to both the PID and the SUPERVISION function blocks. During normal operation the water level is between N1 and N2 and PUMP_CONTROL accepts the XOUT of PID. Otherwise, and when the water level is outside of the range N1-N2 the PUMP_CONTROL accepts the PUMPVAL from the SUPERVISION function block instance.

For the construction of the function block diagram we utilized the IEC 61499 Editor v20010203, provided by Rockwell Automation [6]. This demonstration software, which is in the form of a Function Block Development Kit, enables to build and test data types, function block types, resource types, device types and system configurations according to the IEC 61499 Publicly Available Specification (PAS).

V. DISTRIBUTING THE IPMCS APPLICATION

The next phase of the development process, after the construction of the function block diagram, deals with the mapping of the functionality captured by the IPMCS application into physical resources such as field devices. According to the IEC61499, *devices* may communicate with each other over one or more communication links, and may interface to controlled machines and processes. The IPMCS *applications* may be distributed among one or more devices. As far as, the field devices are interconnected with the same fieldbus, the distribution of the IPMCS application is a simple task. Data and event connections are mapped to the specific's fieldbus communication mechanisms and real time constraints are satisfied by the fieldbus nature. However, things are going to become more complicated, when the application's function blocks must be distributed

among devices assigned to different fieldbuses of the same or different type.

This is a common situation in the industry, where a lot of proprietary fieldbuses have been installed in different time periods under different circumstances. If, for example, an enterprise wants to transfer data between two buildings, with each one having its own fieldbus, it has to interconnect the two heterogeneous fieldbuses. There is also a requirement, from the corporate level, to have integrated monitor and control of industrial process assigned to different fieldbus segments, through the enterprise's intranet.

The obvious solution to the above problem, is to use interworking units to interconnect each fieldbus segment with the enterprise intranet. Although this solution, address the requirements for the integrated monitoring in corporate level, it does not satisfy the requirement for real-time interconnection between fieldbus segments. To satisfy this requirement, we were guided to adopt the network topology shown in figure 3. Our approach to use interworking units between each fieldbus and the enterprise intranet, leaves unchanged the already defined process of each fieldbus and only requires the extra effort for the interconnection of function blocks assigned to different fieldbus segments. This would result, in the least effort that the enterprise should spend over the configuration of the new system.

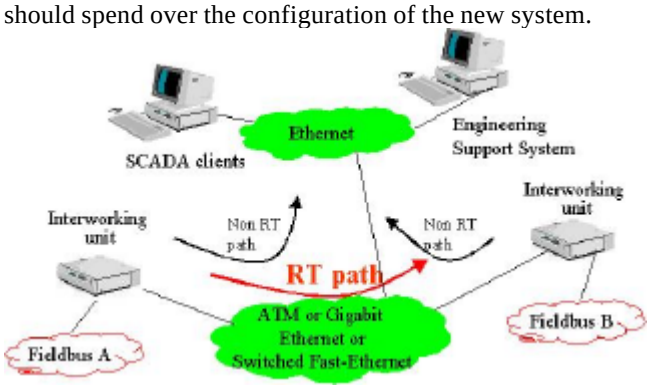

Figure 3. The proposed network topology for the real-time interconnection of fieldbus segments.

The communication subsystem used to interconnect the fieldbus segments must provide the quality of service required to meet the timing requirements. ATM is one of the successfully used technologies for the interconnection of fieldbuses [7][8]. However, to satisfy the key requirement of the simplicity of the communication system as well as the low cost of the equipment, switched Fast-Ethernet was selected.

We were guided to use the above network topology since the industry's solution to the problem of interconnecting heterogeneous fieldbuses, is not very clear. For example, Softing's Profigate is an Ethernet-Profibus Gateway which consists of a PROFIBUS connector, CPU board and an Ethernet TCP/IP software interface [9]. Another example is Siemens' SIMATIC OPC SERVER, which offers libraries for distant connectivity [10]. Two major reasons, though, put these solutions in question: first the real-time transmission between two fieldbuses cannot be guaranteed, due to the Ethernet backbone, and second these gateways offer only PROFIBUS connections. Another commercial product that promises connectivity between independent fieldbus segments, is SST-X-Link from SST Network gateways [11]. Although this product seems promising, a great effort from user's part for configuring is required, and additionally it doesn't offer the capability for remote configuration, diagnosis, monitoring, etc.

Working in the system layer of our architecture for the steam-boiler control system, we construct the system diagram using the constructs of the system layer editor. We next proceed to the distribution of the function block instances and their associated data and event connections from the application diagram to the system diagram resources. Figure 4 represents such a distribution. We assign the function block instance ANALOG_INPUT (FB1) to a Profibus device and the PUMP_CONTROL function block instance (FB5) to a Lonworks device. We also assign the PID, the FB2 and the FB3 function block instances to the Control Application module of the Profibus interworking unit.

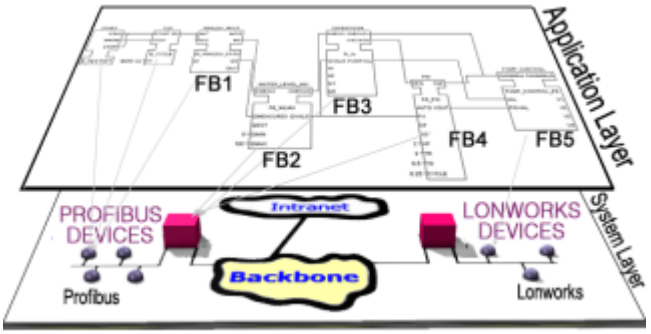

Figure 4. Mapping of application-diagram to system-diagram.

It is evident that an ESS must provide all the functionality required by the function block's distribution and assignment process to the system layer building blocks and mainly to field devices. Actions of the ESS that have no meaning in the application design phase, but refer to the configuration of the underlying communication subsystem must be properly hided by encryption and encapsulation mechanisms.

VI. CONFIGURING THE IPMCS SYSTEM

To implement the above mapping of the application's function block instances, to the system diagram constructs, a configuration phase is introduced. During this phase the interconnections between function blocks across device boundaries and filedbus boundaries must be considered. As it will be evident, the most of the actions may be automated by the appropriate ESS, disengaging the engineer from the particularities of the communication subsystem.

To proceed with the study of the configuration phase, we assume the use of an interworking unit having a modular and IEC compliant architecture as the one briefly described in the followings. For the design of this open interworking unit, we had to agree on some well-defined rules to govern the interactions among its subsystems. We use the term "architecture" to refer to the system's structure that consists of active modules, a mechanism to allow interactions among these modules and a set of rules that govern the interaction [12]. Figure 5, shows this part of the interworking unit's architecture, which covers the operational phase of the system. It is composed of the following active modules:

1. Local Fieldbus Proxy (LFP)
2. Control Application (CA)
3. Remote Fieldbus Proxies (RFP)
4. OPC server

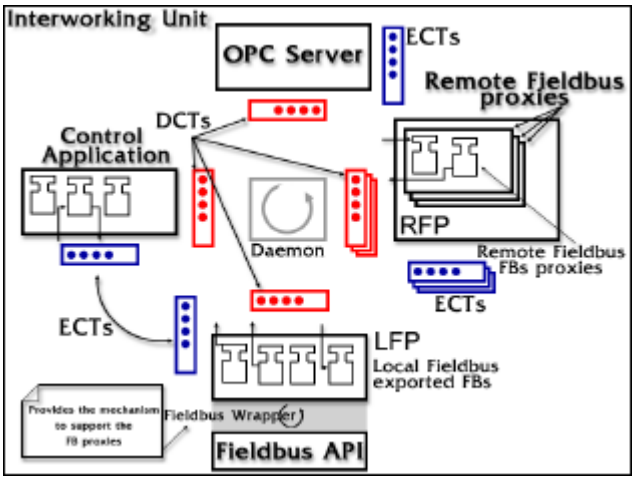

Figure 5. Architecture of the inteworking unit.

The LFP module is used to abstract the specific fieldbus, to the IEC 61499 level so as the interoperability in fieldbus level may be achieved. It mainly consists of the local fieldbus exported function block proxies. A function block is considered as exported, if it provides or accepts an event or data from a function block that is located, either in the CA module or in a remote fieldbus.

The CA module is an optional module and is only required when there is a need to assign function blocks to the interworking unit. We usually discourage this design choice, but there are cases that such a decision is enforced by the system. In these cases, the performance of the interworking unit is degraded and an upper limit to the processing requirements of this module must be defined.

The RFP module acts as a container for all the remote fieldbus proxies that must be present in the interworking unit.

Finally the OPC server module is also an optional module that is required when there is a need for commercially available SCADA clients to be connected to the controlled process. This module acts as a wrapper to our architecture, to provide an OPC-compliant interface. A detail description of the interworking unit's architecture is given in [13].

For the implementation of the mechanism to support the interactions among the above modules:

1. we use a daemon process and a set of data structures, as is shown in figure 5 and described in detail in [5] and
2. we have defined a set of rules to meet all functional and non-functional (performance, extensibility and fault-tolerance) requirements.

Due to the strict timing constraints imposed by the application domain, the interworking unit is characterized as hard real-time. Data processing is expected to recognize and to react to events as soon as possible or even in the ideal case instantaneously. The key issue is how efficiently the interworking unit can handle such requests and how the underlying communication subsystem can provide the required quality of service (QoS). To satisfy these requirements, RTLinux has been adopted for the implementation of the interworking unit and IP over ATM for the interconnection of filedbus segments [13].

The result of the distribution phase of the steam boiler function block interaction diagram is a set of configuration files. There is one configuration file for each interworking unit of the real world layer. The configuration file that is produced automatically by the ESS, contains in human readable representation the configuration of the interworking unit to meet the requirements of the specific application. We adopted XML for the format of the configuration file to allow the exchange of configuration information among different ESSs. We have expanded the XML specific tags introduced by the IEC61499 only for the description of the information that is related to the RFP module. In this part of the configuration file interconnections with remote fieldbuses and function block proxies should be described.

Figure 6, shows the layout of a configuration file, which captures the information for one of the interworking units of the steam boiler control system. We can discriminate the three resources named LFP, CA, and RFP. Each part begins with the declaration of the related resource as RTL_MODULE, which is a custom type. Its body contains references to the function blocks assigned to the resource as well as the definition of the appropriate data and event connections. References to algorithms that the function blocks encapsulate, and they will be downloaded later to the Interworking Unit are also contained.

The LFP and CA configuration parts, for example, refer to the function blocks that exist in the corresponding resource. These function blocks are real function blocks or proxies of function blocks assigned to field devices.

Configuration files are downloaded to the target interworking units. The interworking unit translates off-line the human readable XML representation in machine readable, which is the one expressed by the proper update of ECT, DCT and required proxies.

Figure 7 shows a snapshot of the interworking units internals that highlight the way that function blocks communicate across device and fieldbus boundaries.

For the ANALOG_INPUT function block (FB1) to communicate with the WATER_LEVEL_MU (FB2) that is assigned to CA resource of the interworking unit a proxy is automatically created by the ESS in the interworking unit's LFP module. At the same time the related DCTs and ECTs entries are updated. The interaction mechanism uses this information to interconnect the FB1 proxy with the FB2 function block. To implement the communication links from FB3 and FB4 to FB5 that is allocated to a field device of a lonworks field bus, a proxy of the FB5 is automatically generated by the ESS in the interworking unit's LFP module. At the same time the related DCTs and ECTs entries in both interworking units are updated.

VII. CONCLUSIONS

Using the steam boiler system as a case study, this paper has discussed a general approach for the development of distributed IPMCS applications. We adopted the function block concept, to improve productivity in terms of re-use, reliability, flexibility and interoperability.

We exploited the architecture of the interworknig unit, which was defined to address the problem of interconnecting heterogeneous fieldbus segments, to allow a uniform development of applications from the ESS point of view. Our architecture promotes re-usability, obtains interoperability in the fieldbus level and favors automatic code generation. We constructed the whole architecture

```
<?xml version="1.0" encoding="UTF-8"?>
<--!DOCTYPE System SYSTEM "../LibraryElement.dtd" >
<System Name="Interworking_Unit" Comment="System Configuration" >
  <Identification Standard="61499-2" />
  <VersionInfo Organization="CLAB" Version="0.0" Author="CSI" Date="2000-05-09" />
  <Device Name="Interworking_Unit_WOODY_MACHINE" Type="FRAME_DEVICE" >
    <Resource Name="LFP" Type="RTL_MODULE" >
      <FBNetwork >
        <FB Name="FB_TIMER" Type="TIMER" />
        <FB Name="FB2" Type="FB" />
        <EventConnections>
          <Connection Source="FB_TIMER.TRIGGER" Destination="FB2.GETVALUE" />
          <Connection Source="FB2.EVENT_OUT" Destination="CA.FB3.DO_CALCULATE" />
        </EventConnections>
        <DataConnections>
          <Connection Source="t#500 ms" Destination="FB_TIMER.DT" />
          <Connection Source="0" Destination="FB2.WATER_LEVEL_MU_QMIN" />
          <Connection Source="100" Destination="FB2.WATER_LEVEL_MU_QMAX" />
          <Connection Source="WRAPPER_ITEM_0015" Destination="FB2.WATER_LEVEL_MU_Q" />
          <Connection Source="WRAPPER_ITEM_0016" Destination="FB2.STEAM_LEVEL_MU_INPUT"/>
              ....
          <Connection Source="FB2.SUPERVISION_CHECK02" Destination="CA.FB3.PID_REQ"/>
        </DataConnections>
      </FBNetwork>
    </Resource>

    <Resource Name="CA" Type="RTL_MODULE" >
      <FBNetwork >
        <FB Name="FB3" Type="FB_PID" />
        <EventConnections>
          <Connection Source="FB3.PID_CNF"
                      Destination="RFP.FIELDBUS_ISMOGOOD.FB4.PID_CONTROL_CHANNEL" />
        </EventConnections>
        <DataConnections>
          <Connection Source="2" Destination="FB3.PID_KP" />
              ........
          <Connection Source="FB3.PID_XOUT"
                      Destination="RFP.FIELDBUS_ISMOGOOD.FB4.PID_INPUT"  />
        </DataConnections>
      </FBNetwork>
    </Resource>

    <Resource Name="RFP" Type="RTL_MODULE" >
      <WideNetworkInfo>
        <InterUnit MachineName="FIELDBUS_ISMOGOOD" IP_Address="195.0.1.2"
                                                   FieldbusType="PROFIBUS" >
          <FB_PROXY Name="FB4" Type="FB" ACTUAL="LFP.FB4" >
          </FB_PROXY>
        </InterUnit>
        <InterUnit MachineName="FIELDBUS_DUCKY" IP_Address="195.0.13.12"
                                                FieldbusType="LONWORKS" >
          <FB_PROXY Name="FB13" Type="FB" ACTUAL="CA.FB13" >
          </FB_PROXY>
        </InterUnit>
      </WideNetworkInfo>
    </Resource>
  </Device>
</System>
```

Figure 6. Outline of the Interworking unit's configuration file.

having in mind the system properties of modularity, expandability, robustness, and flexibility.

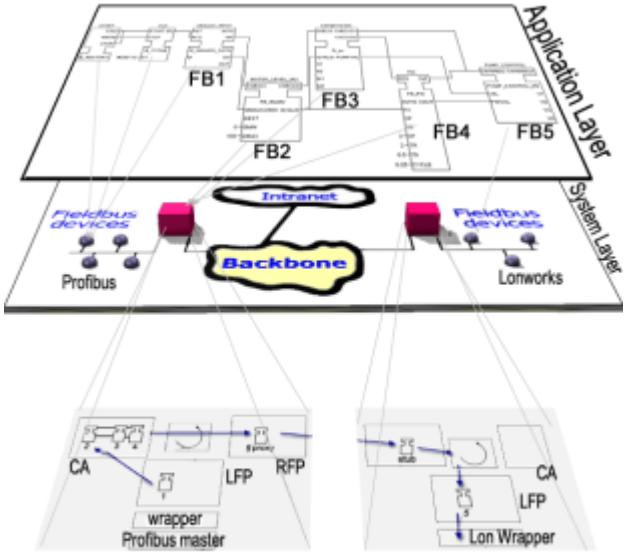

Figure 7. Interworking unit's configuration enables the distribution of the steam boiler control application

We consider the process of developing IPMCS applications in the three bottom layers of our 4-layer architecture of IPMCSs. The result is the construction of an application layer diagram that is independent of particular field devices or even fieldbuses. The subsequent mapping of the application-diagram to the system-diagram, fully supports the distribution phase of the control application. The adopted human readable representation enables the exchange of configuration information among ESSs and the refinement of the configuration file by the engineer to increase performance. The automatic production of the configuration file by the ESS disengages the engineer from the particularities of the communication subsystem. The engineer has only to concentrate to the actual application and not to the implementation details.

We hope this work will help in the direction of simplifying the development process of distributed IPMCS applications and in the direction of developing interoperable IEC 61499-compliant systems that will provide the ability to the industry to utilize heterogeneous software environments, with IEC 61499-compliant software tools and devices available from a wide selection of vendors.

Acknowledgements

Chris Tranoris was supported in part by the Greek General Secretariat for Research and Technology in the context of the PENED 99 ED 469 project. We gratefully thank Nick Kousoulas and Stamatis Manesis for their helpful discussions on the steam boiler control problem.

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] J.- R. Abrial, "Steam-boiler control specification problem", August 10, 1994.

[4] R.W. Lewis, "Modeling Distributed Control Systems Using IEC61499 Function Blocks", Technical Articles, URL: http://www.searcheng.co.uk/ selection/ control/tech.htm

[5] K. Thramboulidis, C. Tranoris, "An Architecture for the Development of Function Block Oriented Engineering Support Systems", IEEE International Symposium on Computational Intelligence in Robotics and Automation, Canada August 2001.

[6] "Function Block Development Kit", Rockwell Automation, http://www.holobloc.com.

[7] O. Kunert, " Interconnecting fieldbuses through ATM", IEEE international workshop on factory communication systems, 1997.

[8] C.Cseh, M.Jansen, J.Jasperneite, "ATM networks for factory communication", 7th IEEE International Conference on Emerging Technologies and Factory Automation, 1999.

[9] Softing's Profigate Datasheet, http://www.softing.com

[10] SIMATIC OPC Server, User guide http://www.siemens.com

[11] SST-X-Link gateway, http://www.mysst.com

[12] Maarten Boasson, "The Artistry of Software Architecture", IEEE Software, November 1995, vol. 12 No 6.

[13] C. Tranoris, S. Aslanis, K. Thramboulidis, "Using RT_Linux for the interconnection of industrial fielsbuses" ASME international, First National Conference on Recent Advances in Mechanical Engineering, September 17-20, 2001 Patras, Greece.