Towards a Service-Oriented IEC 61499 compliant Engineering Support Environment

K. Thramboulidis, J. Koumoustos, G. Doukas
Electrical & Computer Engineering
University of Patras, Greece
{thrambo, koumouts, gduke}@ee.upatras.gr

Abstract- The IEC61499 standard introduces the Function Block (FB) model to support the development of open, interoperable, distributed control applications. Currently available Engineering Support Systems (ESSs) or those under development adopt the traditional architectural styles and do not cover the whole requirements of the development process of FB-based distributed control applications. However, the major drawback of traditional ESSs is that of extensibility. This problem can be addressed by providing a public domain framework that can be easily extended. In this paper an approach that promotes the idea of service-oriented computing in the control and automation domain, is described. A service-oriented architectural framework is proposed to provide the infrastructure for the exploitation of service-oriented computing in factory automation. Features required in the development process will be implemented as services and published in the public domain. These services can be used on demand by control engineers to construct their projects’ specific Engineering Support Environments.

I. INTRODUCTION

The IEC 61499 standard introduces the Function Block model for the development of complex interoperable distributed control applications (DCAs) [1]. It introduces the Function Block (FB) type with the execution control chart (ECC) and the FB network diagram as the main design-phase artifacts. To decrease the productivity gap, the IEC 61499 has also introduced the concept of the Engineering Support System (ESS), a toolset that should guide the control engineer through the development phase of DCAs. Prototype implementations of ESSs are currently available [2] [3] [4] [5] and a number of projects are under way for the development of such ESSs [6][7]. These ESSs are mainly based on a monolithic proprietary toolset that provides the functionality of FB type editor, ECC editor, FB network editor, device editors, system layer editors, etc. Their objective is to assist the control engineer to create FB type specification, FB network diagram specifications, to validate these design specs, and to deploy and execute complex DCAs.

From our experience it is more than evidence that these toolsets are not enough for an effective development process. Control engineers need improved techniques, methodologies and tools to better support the analysis, design, debugging, validation, deployment and verification of the system in order to improve productivity. Currently available ESSs do not fully cover these requirements. Even more control engineers will have to select the tool set that best fits their development requirements and in most of the cases the existing or under development tools do not address all of these needs. Extensibility in the form of extending these toolsets to suit project specific needs is currently slightly addressed by the available toolsets. Some of them [6][7][4], being open source, provide the capability for the control engineer to customize the toolset, an option that normally does not work due to the investment required to explore this capability. These toolsets also allow for model interchange with other toolsets using the XML based specification of 61499 in the representation of their models, but this leads to bigger investments in terms of cost and time required by these different tool sets to be used effectively.

What is needed is an Engineering Support Environment (ESE) where the requirements of the control engineer for the development process will have the principal role. Based on these requirements the control engineer should be able to set up and customize a project-specific ESE by easily integrating through plug-and-play the desirable features. The control engineer to effectively address the complex development process of the next generation agile DCAs wants to pay only for the resources actually used to solve the specific problem, and monolithic environments do not cover this requirement.

In this paper a framework, called SOA4DCS, is proposed, as an extension of Corfu and Archimedes system platform, to address this need. SOA4DCS is a service-oriented framework that intends to enable control engineers to set up and customize the ESE that best fits with the needs of their project. Middleware technology is exploited not only in the execution stage of DCSs but also in their development process. In particular web services [8] are exploited for the development phase while, real-time CORBA is utilized to meet the stringent real-time constraints impose by this kind of applications during run-time. It is shown that this framework can provide the infrastructure required for the easy set up and customization of a project oriented engineering support environment that best fits the needs of the specific project. The investment required for the set up of such an environment is minimum compared with the cost of buying specific tool sets, since it is based on the idea of using services. Using this infrastructure control engineers can implement their own desirable features and incorporate them into their custom engineering support environment. So, the control engineer instead of buying (or developing) software components and bind them together to form a custom ESE, will construct the project-specific ESE as an orchestration of web services that are only used and bound together at the time of use of the particular feature of the ESE.
This provides a powerful and flexible framework for customizing and yet extending the environment to address the control engineer’s particular requirements. It enables the control engineer to construct an ESE by using services by multiple suppliers to meet his needs for the specific project.

The remaining of this paper is organized as follows. In the next section, a brief introduction to the basics of service oriented architectures is given, along with a discussion of its use in factory automation. In section 3, the proposed service oriented architecture that provides a framework for an engineering support environment in control and automation is presented, and specific services are discussed and described. In section 4, the FB type repository web service is described and a prototype implementation is presented. In section 6 the use of a real-time CORBA Object Request Broker that is exploited to provide a service-oriented run-time environment is presented. Finally the paper is concluded in the last section.

II. SERVICE-ORIENTED ARCHITECTURES IN FACTORY AUTOMATION

Software architectures have emerged as an important discipline for software engineers that were looking for better ways to understand their systems and new ways to build larger, more complex software systems [9]. The software architecture involves, according to Shaw and Garlan, “the description of elements from which systems are built, interactions among these elements, patterns that guide their composition, and constraints on these patterns.”

As the level of complexity of today’s systems is continually increasing, traditional architectures that have been defined over the last years seem to be reaching their limit in their ability to enable IT organizations to meet today’s complex set of challenges [10]. Brereton and Badgen in [11] argue that although component-based development, one of the recent architectural styles, offers many potential benefits, such as greater reuse and a commodity oriented perspective of software, it also raises several issues that developers need to consider. Service-oriented computing [12][13] and Service Oriented Architecture (SOA) are being promoted as the next evolutionary approach to address these problem. SOA, which is not only an architecture but also a programming model, defines a new way of thinking about building software systems. A service-oriented architecture is essentially a collection of services [14] along with an infrastructure that enables these services to communicate with each other. This communication can be simple as the case of simple data passing or complex as the case of two or more services coordinating in the context of some complex activity.

The concept of service-oriented architecture appeared from the time CORBA [15] provided the first infrastructure to integrate applications running on different heterogeneous platforms. Faster time-to-market, reduced cost, risk mitigation, continuous business process improvement and process-centric architecture are among the most important benefits of applying SOA [10]. However, the most important advantage of SOA for the factory automation domain is that it can evolve on existing system investments rather than requiring a full-scale system rewrite. Legacy systems can be encapsulated and accessed via service interfaces [16], preserving the huge amount of investment in this area.

A. The service construct

A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. A service can provide a single discrete function, such as converting an FB type XML specification to Java specification, as shown in figure 1, where the graphically presented by Eclipse interface of this service is shown, or it can perform a set of related business functions, such as handling the various operations in the deployment process, such as the one discussed in the next section.

Figure 1. An Eclipse graphical interface of a web service that converts an FB type Specification to Java specification.

A service has many characteristics that an architect must consider and specify as required. Performance, capacity, business organization, risks and issues, ownership, reliability, security, business impact, tolerance, service contract, and dependencies constitute a list of categories in which a service could require specification [17]. However, all services do not require the same level of definition. In any case, the following two questions “what does the service do?” and “what is the major required by the user functionality?” should be clearly answered by the specification of the service. The central role of the specification of user’s required functionality is the issue that differentiates SOA from object-orientation [17]. The primary construct of the object technology is the object that represents an entity (structure and behaviour) while the one of OSA is the service which represents how its consumers wish to use it.

B. Using Web services in control and automation

Web Services provide the infrastructure required to connect services together into a service-oriented architecture. Web services is a collection of technologies, including XML, SOAP, WSDL, and UDDI, that can be used to implement a service-oriented architecture. They let you build programming solutions for specific messaging and application integration problems. XML is used to create a robust connection between the service provider and the service consumer. SOAP, which becomes the de facto standard for application-to-application communication over internet, provides the envelope for sending the Web Services messages. HTTP is usually used for messaging, but other means of connection may also be utilized. WSDL is expected to become the de facto standard for describing services in the next few years. So defining existing
factory automation systems using WSDL will help enterprises add agility to their factory IT environments.

Web services provide a number of advantages when using them over the global internet. However, when the majority of users are working within a corporate intranet current SOAP engines introduce a great overhead that results in an order-of-magnitude performance difference comparing with equivalent calls using other protocols such as IIOP of CORBA [18]. There are huge storage and processing requirements compared to the CORBA solution. Generating and parsing XML documents is a time consuming task that also needs a lot of memory compared to the actual data that is in these documents. With XML the actual information is only a small portion of the total data resulting to a high overhead. So, human readability provided by XML comes at an expensive price. The SOAP and XML-RPC messages are just over 14 times as large as the binary CORBA messages [19]. In the same paper it is also reported that for a test including the sending of 5,000 integers to the server, SOAP and XML-RPC took 882 and 66 times longer than CORBA on the same machine, respectively.

So, since no one distribution approach solves all problems, what many organizations have concluded is that they need to support multiple distributed object protocols and access mechanisms within their enterprise. A single application API may need to be available as an external Web Service using SOAP over HTTP, while RMI-IIOP can be used for internal remote clients [20]. This is the approach adopted in the context of the proposed architecture, where Web Services are utilized to provide a flexible effective, reliable, scalable, and extensible ESE and its interaction with the network of devices during the development stages, while real-time CORBA is utilized to get an efficient device-to-device interaction required during run-time to meet real-time constraints.

C. Related work

Other research groups are already exploiting Web services in factory automation [21][22][23]. In [21] authors present the SIRENA approach that intends to create a service-oriented framework for specifying and developing distributed applications in diverse real-time embedded computing environments. They examine the potentiality of the service-oriented paradigm and in particular the one that is based on Web Services in the industrial automation sector. They propose to extend the SOA paradigm “into the device space, that is, implementing a high-level communications infrastructure based on Web Services protocols at the device level, including in the lowest-level devices.” Authors even though state that “even this is not yet feasible today –for reasons of cost-effectiveness and of responsiveness in the presence of severe real-time constraints” they argue that this is definitely the ultimate perspective of the SIRENA approach. However, they do not provide any solution to this direction. Even though they state that the objective is to create a service-oriented framework for specifying and developing control applications in the industrial automation sector, they mainly focus on the infrastructure required during run-time and they do not consider the possibility of exploiting web services into the development process and deployment process.

Lastra and Delamer in [22] propose the use of semantic Web Services as a means to address the challenge of rapid reconfiguration of manufacturing systems required in order to evolve and adapt to mass customization. Authors propose the use of Web services down to the sensor device level to support orchestration of services as well as choreography during configuration and re-configuration. However, they do not describe the way that real-time constraints can be met in this kind of systems given the great overhead in performance and the increased storage and processing capabilities required by such end devices, as sensors, to provide semantic Web Services. Run-time reconfiguration, at least as far as the software level is considered, has strict deadlines that can not be met by the current status of web services. Authors also argue that a methodology “for developing self-configurable and autonomous production systems using elements that equally may or may not have previous knowledge on the type of other elements with which interactions will occur” is required. However, they do not propose and describe such a methodology as well as the toolset required to support such a methodology and the possibility to utilize SOA and especially web services to this direction.

In [23] XML and web services are applied to solve the integration problem in order “to reach a higher level of efficiency of design work during an automation project.” A prototype development based on two proprietary tools that are used in plant design is presented with basic objective to establish a bidirectional message transfer from one system to the other.

Brenan in [24] describes a prototype implementation of an automation object repository that should be able to contain metadata of all the artefacts used in the development process of distributed control and automation systems, i.e., machines, tools, methodologies, platforms, intelligent devices, etc. The described prototype implementation is developed on CAREO (Campus Alberta Repository of Educational Objects - http://www.careo.org/) that has two components the CAREO web application and the ALOHA metadata management server. The whole approach is based on a centralized management scheme that does not adopt neither utilizes the concepts introduced by service oriented computing and web services.

In [25] real-time CORBA was exploited to provide a service oriented architecture for a device on which FB-based reconfigurable control applications can be executed. To our knowledge there is no other work at the moment towards the direction of utilizing SOA for the definition of an engineering environment in the form of an extended engineering support system that will exploit the advantages of SOA and Web Services.

III. A SERVICE-ORIENTED ARCHITECTURE FOR DCSs

Publicly available ESSs or those under development are mainly based on a monolithic proprietary toolset. These toolsets provide, as shown in figure 2, the functionality of FB type editor, ECC editor, FB network editor, device editors,
system layer editors, etc. Their objective is to assist the control engineer to create FB type specification, FB network diagram specifications, to validate these design specs, and to deploy and execute complex DCSs. These ESSs have been developed adopting traditional architectures that have been evolved during last years. A more flexible solution can be obtained adopting the service-based model of software. In such a model users can either assemble their services out of existing ones, or develop and evolve atomic services using traditional development techniques. The big advantage of this approach is that these services are sold and assembled on demand.

This model will allow control engineers to define their ESE in a way that the requirements for the development process will have the principal role. Based on these requirements the control engineer should be able to set up and customize a project-specific ESE by easily integrating through plug-and-play the desirable features. To effectively address the complex development process of the next generation agile DCAs control engineers should be able to pay only for the resources actually used to solve the specific problem, and monolithic environments do not cover this requirement.

Figure 2. Currently available ESSs are based on a monolithic proprietary toolset.

The above presented technologies seem to provide a promising infrastructure on which to construct the next generation ESEs. However, even though these technologies are well understood in isolation, the challenge is to place all these into a cohesive engineering framework so that it could be possible to apply them in the production process of the next generation factory automation systems. In this section we describe our proposal to this direction in the form of an architectural framework.

Figure 3 presents an overview of the proposed SOA4DCS service oriented architectural framework. The service is the basic construct in this architectural framework. All functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form the processes required for the development, deployment and execution of control and automation software.

Defined services include model definition, i.e., editing functions, implementation model generation functions, FB type repository functions for the discovery of the required FB types, deployment functions, as well as monitoring functions. An interesting question not answered yet has to do with the level of granularity that functions will be mapped to services.

Services, which should be completely independent of one another, should operate as black-boxes, with clients to neither know nor care how these services perform their function. Services are described by means of WSDL providing invokable interfaces, which define not the technology used to implement it but the nature of the service through the required parameters and the nature of the result. At the architectural level, it is irrelevant whether these services are local or remote provided by other vendors. It is also irrelevant what interconnect scheme or protocol is used to effect the invocation, or what infrastructure components are required to make the connection. The service may be within the same or in a different address space. Moreover, the so constructed development environment must include and enforce according to [10] a methodology that will clearly prescribe how services and components will be designed and built in order to facilitate reuse, eliminate redundancy, and simplify testing, deployment, and maintenance. Such a methodology is also required to guide the control engineer through the development process.

It is expected that a great number of services will appear to provide either generic functionality or specific required in specialized application domains. In any case the identification of services in such an environment is of great importance. A number of parameters such as performance, flexibility, maintainability and re-use, define the level of granularity in service definition. In the remaining of this section specific services are presented and discussed.

A. Model-editors as Web services

Part of the functionality of an IEC compliant ESS can be easily provided as Web Services, as for example the implementation model generation service and the FB network verification service. However, for part of the functionality, as for example the one related to graphical editors (FB type, FB network, etc.) there is no obvious way to implement and provide it through web services. The approach adopted in the context of this work is based on the existence of a generic model-editor that will provide the basic functionality required by a graphical model-editor. This editor, which will be executed in the address space of the user, will be specialized to the selected by the user editor type, i.e., FB type editor, FB network editor, ECC editor, system layer editors, etc., by utilizing specific Web Services provided by other vendors. An IEC-compliant 61499 model editor service will provide all
these constructs that are required for the specific model as well as their semantics and the XML schema that should be used for the model specification. So the control engineer will search using UDDI services for the required Web Services that will support the construction of the required models.

**B. System layer design models**

Device vendors should provide for their devices a description using a standard device description language. A device description will describe the device characteristics concerning storage, processing and communication capabilities of the device. Device descriptions will be stored in the vendor’s device meta-data repository. A web service that will be developed by the vendor and made available through specific UDDI will allow control engineers to search for the device that meets the required characteristics. Through this web service device descriptions can be downloaded and used for the design of the system layer diagram. Later on, and after the verification of the design models of the DCS the actual devices can be bought through the same web service. Device descriptions for the specific project will be stored in the project’s repository and will be exploited by design-model analysis tools to verify that the application’s design diagrams are implementable.

**C. The deployment web service**
Another important service is the one required to cover the requirements imposed by the deployment process. The choice to have every field device to provide a Web Service to allow the downloading and configuration of the part of the control application that will be assigned to the specific device was abandoned since it cannot satisfy the requirements imposed by run-time re-configuration scenarios. This approach proves to be inefficient as it imposes an unnecessary overhead to the industrial devices that may have limited storage and processing capabilities. Instead of it, a centralised Web service is defined to allow the deployment into a network segment of devices. The adopted approach seems more appropriate and flexible as it hides the complexity of the industrial network and leads to a more portable solution. The so defined service can be implemented either using the native execution environment and binary communication infrastructure to meet real-time constraints, as shown in figure 4, or as an orchestration of more primitive atomic services that can be provided assuming a lower granularity level in service definition. In this later case atomic services should be implemented using the native execution environment and binary communication infrastructure.

Extra operations may allow the discovery of devices and extraction of device capabilities so that the degree of automation of the deployment process can be considerably increased.

D. Implementation-model generation web service

Model-to-model transformers as the one used to create the implementation model of the FB type in a specific implementation environment, i.e., Java, C++, CCM, etc., will also be provided as Web Services by different vendors. The implementation model-generation service covers the need of transforming the FB type design specification to an executable specification (implementation model) for the specific platform. Implementation models of FB types should be in executable form in order to be executed on the specific application execution environment. As proof of concept a web service for the generation of the implementation models of FB types has been developed for the RTAI-AXE Archimedes execution environment. The produced implementation model is compliant with the proposed in [26] execution environment that utilizes the dynamic library loading mechanism of Linux OS to support a reconfigurable deployment scheme for FB types. The source code of the FB type dynamic libraries is produced automatically by parsing the FB type design model in XML form either using an Archimedes ESS interpreter or an independent generator written in Java using the Xerces Parser. This independent generator was utilized to construct a servlet based web service that accepts FB type specification as attachment in XML form and returns the corresponding generated library source code.

Implementation model generation services could be utilized automatically by the deployment service of the ESE to get the required by the target device implementation model for the FB types assigned to the specific device.

IV. FB TYPES REPOSITORY WEB SERVICE

This is a key service, as it enables control engineers to increase reusability by locating already available FB type specifications and using them in the development process of their control applications. This service also allows vendors to develop generic and specific FB types and advertise them for sale through the web infrastructure. If such an infrastructure
will be established, it is estimated that a lot of machine and tool vendor will provide specific web services that will allow control engineers to search and locate the FB types that better meet the requirements of their application. For the web service to be reachable by control engineers its developer should publish the service in a Universal Description Discovery and Integration (UDDI) repository. UDDI, which provides a service registry and API to publish, locate and search web services as shown in figure 5, can be thought of as a DNS for business applications. The UDDI can store or point to a Web Service Description Language (WSDL) document. Such a document describes the interface of the web service, i.e., the messages that can be exchanged, the operations, where to contact the service, what transport protocols can be used, etc. However, it does not describe what the service does, how it does it or how you should use the service, i.e., does not provide semantic information about the service. The repositories in which providers store their FB types will be considered as black box implementations as long as the messages and data formats described by the WSDL interface are maintained. It can be a simple database system with a specific search API, or it could be enriched with semantic information, taxonomies and ontology based classification rules. All the above can improve the ability to use the repository from both the costumer and the provider point of view.

The control engineer through the UDDI API will search for appropriate web services that give access to FB types repositories and then using the appropriate WSDL interfaces will access the web services and search the vendors FB type repositories.

Semantic annotation, which is a very important part of today’s and mainly futures web, would resolve differences in terminology providing a common understanding of the domain, organize knowledge in conceptual spaces, and make it machine understandable. This way, providers will have the ability to easier build their FB types repository as they will have a reference point that they could extend and share. On the other hand, consumers would also benefit as search, location, combination and use of such services would be automated. Keyword-based search will be replaced by query answering based on well defined, organized taxonomies. Software agents will carry the burden forming the knowledge and interactions in a human-friendly way.

To demonstrate the applicability of the above described process and technologies in factory automation a prototype web service through which FB types can be located and downloaded for free, was developed. This web service was published in a local UDDI to allow for any user to try it. A WSDL interface for this service is also published at the same UDDI giving the ability for the user to construct a web service client and invoke the service. The provided UDDI will be reachable through the CORFU and Archimedes web sites.

V. THE RUN-TIME SERVICE-ORIENTED ENVIRONMENT

For the run-time environment to meet the stringent real-time constraints impose by control applications during run time, real-time CORBA was exploited to provide a service-oriented execution environment. Real-time CORBA is a middleware that provides QoS required by real-time applications. The publish-subscribe model, that was adopted, preserves compliance with the IEC model, satisfies the QoS required by DCAs and allows a flexible deployment, re-deployment and re-configuration of control applications, even during run-time.

For our prototype implementation the Zen Real-time ORB, were selected. ZEN [27] is a Real-time CORBA ORB implemented using the Real-time Specification of Java (RTSJ), thereby combining the benefits of these two standard technologies. ZEN’s ORB architecture is based on the concept of layered pluggability [28] where various components of the middleware may be “plugged” or “unplugged” on an as-needed basis allowing flexible middleware configuration.

Since, Zen does not support for the time, the Event Service nor the Notification Service of CORBA specifications extra functionality was required in order to provide the services required by the Application Execution (AE) layer of the execution environment. Our prototype Industrial Process Control Protocol (IPCP) implementation, which provides the services required for the run-time communication of FB instances, utilizes both ZEN’s Naming Service and its ability for Synchronous and Asynchronous transports for IIOP to establish any connection between the remote components that are needed by the DCA.

As shown in figure 6 the IPCP layer is composed of the following services: a) the IPCP-Configuration Management service (IPCP-CMS), b) the Publisher service, c) the Subscriber service, and, d) the Event Channel Management service. A detailed description of the proposed architecture for the execution environment and a prototype implementation are described in detail in [25]. However, it must be noted that the concept of demand led use of services introduced by service-oriented computing is not required to the device level.
VI. CONCLUSIONS

Currently available or under development IEC-compliant Engineering support systems do not provide the flexibility required from the development process of complex tomorrow’s agile factory automation systems. The most important limitations to this inability are introduced by the traditional architectural paradigms that were utilised to construct them.

A more flexible and more effective development process can be obtained exploiting the demand led paradigm introduced by the service-oriented computing model. This is the approach adopted in this paper to propose an engineering support environment in the form of an architectural framework. In this architecture the requirements of the control engineer for the development process will have the principal role during the process of setting up and customizing the project specific ESE. The whole process will allow the control engineer to easily integrate through plug-and-play the desirable features and effectively address the complex development process of the next generation agile DCAs paying only for the resources actually used to solve the specific problem. The adoption of such a service-oriented architecture in the development process of control and automation systems is expected to enforce the currently closed and dominated by a few vendors market to an open demand led market where small vendors will play significant role.

ACKNOWLEDGMENTS

This work has been funded in part by the Greek General Secretariat for Research and Technology in the context of PENED 2003 03ED723 project.

REFERENCES

[27] ZEN, freely available, open-source middleware, http://zen.ece.uci.edu/zen/