MODEL DRIVEN DEVELOPMENT OF FUNCTION BLOCK BASED DISTRIBUTED CONTROL APPLICATIONS

Kleanthis Thramboulidis, Dimitris Perdikis, Spiros Kantas

Electrical & Computer Engineering, University of Patras, Greece.
thrambo@ee.upatras.gr, {perdikis,kantas}@ceid.upatras.gr

Abstract: In this paper, an approach for the model driven development of distributed control systems is presented. The proposed approach adopts the IEC61499 Function Block construct for the design phase and supports many different implementation platforms. Specific model-to-model transformers have been developed to automate the transformation of FB design models to CORBA component model (CCM) executable code. GME, a meta modelling tool, and CIAO, a QoS-aware CCM implementation, were utilized to develop Archimedes, an IEC-compliant prototype Engineering Support System. Archimedes demonstrates the applicability of the proposed approach and greatly simplifies the development process of distributed control applications.

Keywords: Computer aided engineering, control system design, Function Block, Distributed computer control systems, Industry automation, component based development.

1. INTRODUCTION

Today's control applications are usually developed in the form of large monolithic software packages that are difficult to be maintained, modified, and extended (Heck *et al.*, 2003). The engineering tools that are used in the development and deployment processes address a little, or not at all, dimensions such as modularity, flexibility, extensibility, reusability, and interoperability. To address these problems the International Electro-technical Commission (IEC), has defined the basic concepts and a methodology for the design of modular, re-usable Distributed Control Systems (DCSs). The IEC61499 standard defines the Function Block (FB) as the main building block and the way that FBs can be used to define robust, re-usable software components that constitute complex DCSs. However, the IEC model does not define the way that FB design models will be implemented and does not exploits current trends in Software Engineering.

Advances in software engineering may facilitate the development and deployment of complex DCSs. Model Driven Development (MDD) and Component-based architectures promote code reuse and significantly decrease development and validation time. CORFU FBDK (Thramboulidis and Tranoris, 2004), an IEC-compliant Engineering Support System (ESS), integrates the Unified Modelling Language (UML) with the IEC FB model to provide a model driven development process for the analysis and early design phases of control applications. In this paper, this process is extended to address the design and implementation phases of DCSs. The adopted MDD paradigm considers as primary artefacts of software development not programs, but models created by modelling languages. The CORBA Component Model (CCM) that is adopted as implementation platform in the proposed approach is the OMG proposal for component based development. CCM defines the CORBA component as the basic building block and establishes standards for implementing, packaging, assembling, deploying and executing applications as aggregation of components.

Archimedes, a prototype system platform has been developed to demonstrate the applicability of the proposed approach. Archimedes enables the

developer to construct the FB-based design models of the control application and the subsequent automatic transformation of these models to executable ones in the form of deployed executable components. GME, a meta modelling configurable toolset, was adopted and tailored to the control and automation domain. GME supports the easy creation of domain-specific modelling and program synthesis environments, and provides the generic functionality of graphical development environments required to address the needs of an IEC-compliant ESS. CIAO, a QoS-aware CCM implementation (http://www.cs. wustl.edu /~schmidt/ CIAO.html), was selected to move from the FB-based design model, to a component based, modular, reconfigurable implementation model. Other CCM implementations such as OpenCCM, MicoCCM, and Qedo, can also be used. To our knowledge there is no other process or tool that provides re-configurable executable code for IEC61499 FB-based design models.

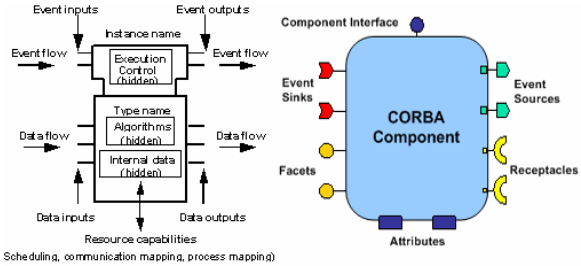
The remainder of this paper is structured as follows: In Section 2, a brief overview of the CORFU approach and the technological background used in this paper is given. In section 3, the proposed Model Driven Development process is presented. In section 4, Archimedes, a prototype IEC-compliant ESS that fully supports the proposed approach is presented. Finally, the paper is concluded.

2. TECHNICAL BACKGROUND

2.1 The CORFU approach

The CORFU development process is an IEC61499-compliant process that defines a series of workflows to cover the phases of requirements, analysis, design, verification, implementation and deployment of control applications. It adopts best practices from component-based development, and utilizes specific UML diagrams and integrates them with the FB concept. The FB is an abstraction mechanism that allows industrial algorithms to be encapsulated in a form that can be easily understood and applied by industrial engineers. The FB consists of a head that is connected to the event flows and a body that is connected to the data flows, as shown in Figure 1. The functionality of a FB is provided by means of algorithms, which process inputs and internal data and generate output data. FB instances, interconnected by event and data connections, constitute the design model of the control application.

The well-accepted use-case concept is utilized for the requirement specifications. Component Interaction Diagrams, of two levels of abstraction, are utilized for the first realization of system's use cases and are used to show the system's internal components and the way they collaborate to provide the required behaviour. Statecharts and class diagrams are constructed in parallel to create the analysis model of the application. The so created UML-based analysis model is transformed to FB-based design specifications that are better understood by control engineers, and increase the reusability of legacy applications. The CORFU development process is already supported by a prototype ESS, namely the CORFU FBDK (http://seg.ee.upatras.gr/CORFU).



(a) Function Block *(b) CORBA component*

Fig. 1. The FB type and the Component Type.

2.2 The Generic Modelling Environment

GME is a configurable toolset with generic functionality for graphical development that supports the easy creation of domain-specific modelling and program synthesis environments (Ledeczi, *et al*., 2001). GME adopts the Model-Integrated Computing paradigm that systematically applies domain-specific modelling languages to engineer computing systems ranging from small-scale real-time embedded systems to large-scale enterprise applications (Gokhale, *et al*., 2002). GME has a modular component-based architecture that makes it easily extensible.

GME is tailored to the specific application domain through the construction of meta models that specify the modelling paradigm of the application domain. This modelling paradigm defines the family of models that can be created using the so constructed modelling environment. The tool's functionality can be further expanded with external software components of three categories: Interpreters, add-ons and plug-ins. Interpreters can be used to generate, by accessing the GME models, the input for static or dynamic analysis tools, configuration files for COTS tools or even source code. Add-ons, which are useful for extending the capabilities of the GME User Interface, can react to GME-events, i.e., events generated by its core components, to provide the required extra functionality. Plug-ins are paradigm-independent components that provide generic functionality, as for example a plug-in which searches for objects based on user defined criteria. The concept of aspect is utilized by GME to allow the developer to focus on selected views of a design document.

2.3 The CORBA component model

CCM is an example of component middleware that was specified by the OMG to address the limitations of the CORBA object model (Wang, *et al*., 2001). The CCM specification defines the construct of

CORBA component and establishes standards for implementing, packaging, assembling, deploying and executing applications as aggregations of components. A CCM component collaborates with its environment through ports, which are of four types as shown in figure 1. Facets define the provided by the component interfaces which can be invoked either synchronously or asynchronously. Receptacles provide a standard way for specifying interfaces required by the component to function properly. Event Sources/Sinks allow the component to loosely interact with its environment based on the publish-subscribe paradigm.

CCM defines a container mechanism, which serves as the runtime environment for CORBA component implementations that are called executors (Schmidt and Vinoski, 2004). The container allows the execution of components and provides a set of interfaces to be used by components to access various CORBA services, such as event notification, transaction and security. It also manages the life cycle of a component through a set of interfaces called callback interfaces that should be provided by the component.

CCM provides to component developers a framework called CCM Implementation Framework (CIF), which allows the rapid development of applications increasing the ratio of generated code to handwritten code. However, CCM, since it was designed primarily for the enterprise applications domain, is unsuitable for domains such as distributed control applications that need to satisfy more QoS aspects and mainly timeliness. Control applications have stringent QoS requirements and any failure to meet these requirements may lead to catastrophic consequences. A solution, to make CCM applicable to the real-time domain, is to combine component middleware with real-time CORBA to create a QoS-enabled component middleware. However, this cannot be obtained by only utilizing the real-time CORBA, since even though it standardizes the mechanisms and interfaces for managing the QoS policies, it does not separate the real-time policy configurations from the application logic. On the other hand, a QoS enabled component middleware, such as CIAO, offers performance and predictability, while improving the flexibility to compose and configure the key QoS aspects of distributed real-time systems.

3. OUR MODEL-DRIVEN APPROACH

The proposed in this paper approach is in the context of the Model Integrated Mechatronic (MIM) paradigm that is described in (Thramboulidis, 2004). MIM is a new paradigm for the development of complex manufacturing systems that promotes model integration not only in implementation space artefacts but also during the early analysis and design phases of the development process. It supports the model driven development of complex MechaTronic Systems through the evolution of models that have as primary construct the MechaTronic Component. MIM exploits MDA and the FB approach to allow the MechaTronic System's builder to compose the design model of the system from already existing MechaTronic Component descriptions and proceed through an automated model transformation process to the implementation model of the system. In the context of this work the focus is on the application layer of the MIM Architecture. A fully automated process for the generation of the implementation model of the control application from the FB-based design models is presented.

To support the automated generation of the implementation model of the control application in the form of component based application, the following approaches are considered:

1. *The straightforward transformation approach.* According to this a straightforward transformation of FB-based models such as FB types and FB networks, to component and component assembly implementation artefacts such as .idl, .cidl, and .csd files is supported.
2. *The intermediate-model transformation approach.* This approach utilizes artefacts of higher layer of abstraction, such as the component type and the component assembly, to construct an intermediate representation that is next transformed to the final implementation model.

Both alternatives are based on the fact that the semantics of the IEC61499 FB type can be mapped to the CCM component. However, the intermediate-model approach results in a more powerful and flexible model transformation process since it:

- increases the portability of the design model,
- utilizes commercial CCM tools for the generation of the implementation model, and
- constitutes the first step towards the use of component based model analysis tools for performance and schedulability analysis.

Figure 2 presents the above two alternatives for the transformation process of FB type to the corresponding CCM implementation space artefacts.

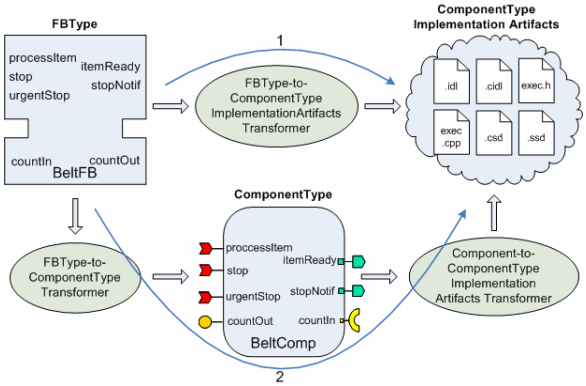


Fig. 2. FBtype-to-ComponentType transformation alternatives.

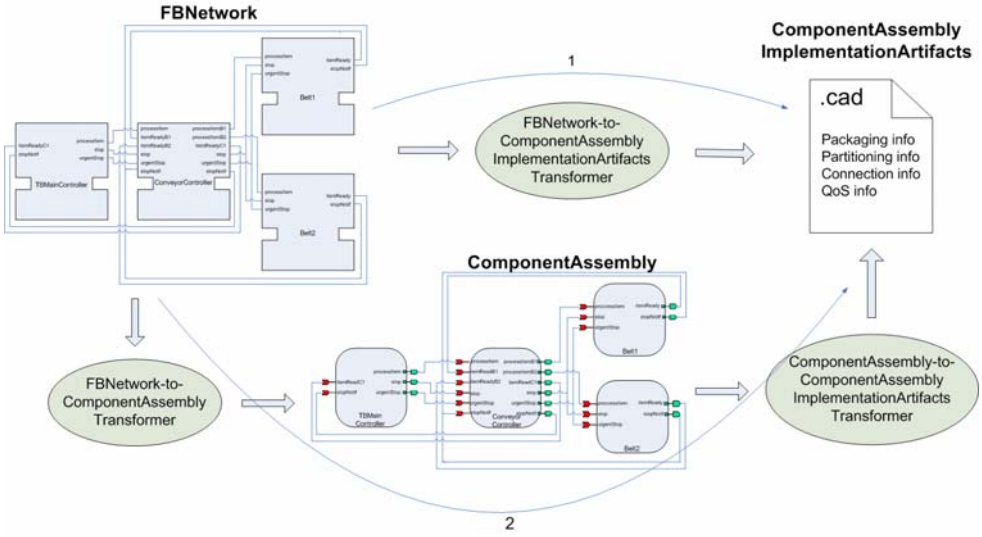


Fig. 3. FBnetwork-to-ComponentAssembly transformation alternatives.

The first alternative that has already been developed is based on the following mapping: Event inputs and outputs are mapped to event sinks and sources respectively; data inputs and outputs can be mapped either to facets and receptacles respectively invoked via asynchronous method invocation or to event sinks and sources. Since the CCM component does not have a construct to directly map the FB's Execution Control Chart (ECC), a template class was defined to capture the FB's behaviour. This template class is composed of: a) a vector which contains a set of entries that correspond to the ECC's transitions, and is initialized by the constructor of the Component's executor class, b) a pure virtual method called "condition", which is implemented in the Component's executor class and implements the conditions for each transition, and c) a "ProcessEvent" method which implements the statechart of the component. The executor class of every component type is defined to inherit this template class.

Figure 3 shows the adopted process for the transformation of FB networks to implementation artefacts. According to the first alternative, the one already developed, a straightforward transformation to CCM component assembly implementation artefacts is utilized. The second alternative is based on the transformation of FB networks to CCM component assemblies. Component assemblies can be subsequently transformed utilizing commercially available CCM tools to CCM component assembly implementation artefacts. These artefacts are next utilized by specific tools such as packaging, assembly and deployment tools, to execute the control application. Packaging tools are used to package implementations of components with their XML-based metadata. Assembly tools use XML-based metadata to describe component compositions, including their actual locations and the interconnections between them. The so created component assemblies and composition metadata are used from within appropriate deployment tools to deploy components into component servers, to interconnect and configure them and then run the control application.

4. ARCHIMEDES

4.1 Archimedes engineering support system

A prototype ESS, compliant with the IEC61499 standard, has been developed. Domain specific meta models have been created with the GME notation that is based on UML. The function block type meta-model and the function block network meta-model constitute the source meta-models that define the design space of the control application domain. These meta-models explicitly define all the syntactic, semantic and presentation information regarding the FB concept for the design of distributed control applications. UML class diagrams have been used to model the syntactic definitions, while the Object Constraint Language (OCL) was utilized to specify the static semantics as constraints. The above meta-models fully define the ESS editors, i.e., the FB-type editor, the ECC editor, and the FB-network editor.

The FB-type editor allows the developer to define the function block types that constitute the control application. Even though FB-types are not currently defined in an IEC-compliant graphical notation, such a notation can be easily obtained by utilizing GME's add-ons. The ECC editor allows for the specification of ECC types and their subsequent use in the definition of FB types. The modifications proposed by (Thramboulidis, *et al.,* 2004) to the ECC model can easily be implemented providing an alternative more flexible transformation process to the more widely used statechart notation.

Even though FB-types are not currently defined in an IEC-compliant graphical notation, the instances of the so created FB types appear in the FB network in

an IEC-compliant notation, as shown in figure 4. The so created graphical representation is translated to XML specifications that can be used for model interchange with other FB-based tools such as the FBDK of Rockwell Automation and the CORFU FBDK. Already defined FB-types can be imported in Archimedes ESS allowing:

a) the reuse of already defined FB-types from other vendors, and
b) the tool to be easily integrated with CORFU FBDK to exploit the integration of UML with the FB concept that is provided by the later.

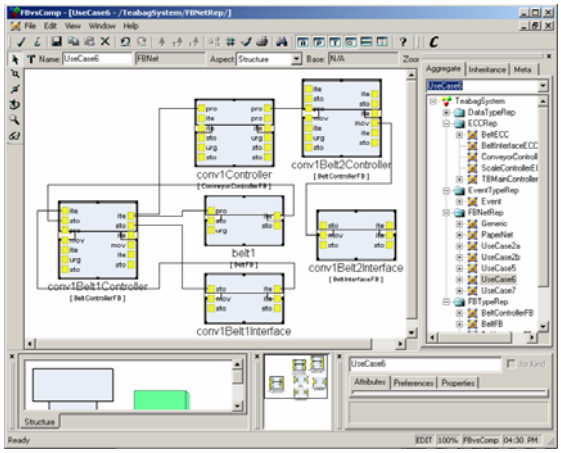


Fig. 4. The FB-network editor of Acrhimedes ESS.

The FB-network editor is based on the FB network meta-model part of which is shown in figure 5. It is utilized by the control engineer to create networks of FB instances with the appropriate event and data connections that enable these instances to collaborate in order to provide the required by the FB-network behavior. FB-networks can be considered as realizations of use cases if the requirements of the control application has been captured using this technique. Alternatively, the CORFU FBDK can be used to support a model driven development process from use case based requirements specifications to FB-network design specifications. The so created FB-networks can be imported by the Archimedes ESS and utilized for the subsequent refinement of the design model and the automatic generation of the executable one. Figure 4 presents a FB-network for the Teabag Boxing System (TBS) case study that has been created using Archimedes ESS. TBS is a simplified version of a real world system that is used in the production chain of packed teabags. The prototype control system that has been developed with Archimedes ESS to demonstrate the effectiveness of the proposed approach can be downloaded from http://seg.ee.upatras.gr/MIM.

4.2 Implementation-model's generation process

The so created FB design model is implementation platform independent and can be utilized for the automatic generation of many different implementations either to meet specific QoS characteristics or to satisfy implementation platform constraints. A prototype process for the generation of a CCM implementation model has already been defined. A Real-Time Java and a real-time Linux one are under development.

For the automatic generation of component and component assembly artefacts of the CCM implementation model from the so created FB-based design models of the control application, Archimedes currently supports the straightforward transformation process. The model-to-model transformers access through the GME API the source domain models and automatically constructs, utilizing the source and target domain meta-models, the corresponding component implementation artefacts for the component types as well as the corresponding component assembly artefacts for the component networks. For example the results of this transformation process for the FB-network of the TBS are a set of .idl, .cidl, .csd, .ssd, .h and .cpp files for each FB type, as well as a .cad file that contains packaging, partitioning, interconnection and QoS information for this FB-network.

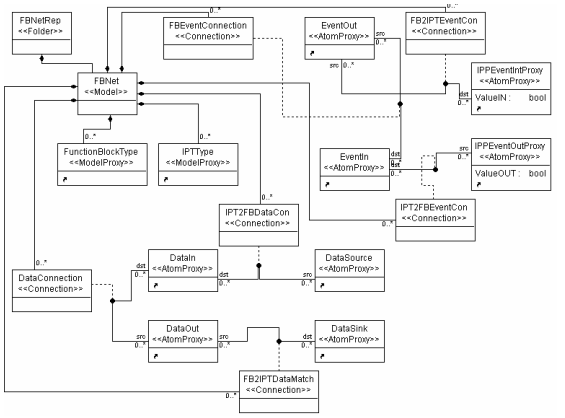


Fig. 5. FB network meta-model in GME notation.

Even though the straightforward approach resulted in a robust implementation process, it was proved hard and error prone to develop this transformer from scratch. It is estimated that the process of constructing interpreters can be automated by the GME. The idea is to construct an additional model that should capture the mapping of the constructs of the source meta-model to the constructs of the target meta-model, and to include in this model, the semantics that are required for the automatic generation of the model-to-model transformer from the source and target meta-models.

A more flexible transformation process is obtained following the approach of the intermediate component and component assembly models that utilizes already existing CORBA component based modelling tools such as the CADML (Lu, *et al.*, 2003). However, working to this direction it was proved that the CCM and the component assembly meta models that are provided by the CADML, have to be customized and extended so as to be applicable

to the FB concept and contain semantics for ECCs and algorithms. CADML can be utilized in a subsequent step to refine the component models and to automatically create the component assembly descriptor.

4.3 Archimedes execution environment

The currently supported execution environment of Archimedes is based on CIAO. CIAO is implemented on top of TAO (Douglas, *et al*., 1999), a Real-Time ORB that implements the Real-time CORBA 1.x specification and is widely used in many commercial and academic projects. Adding extra layers and operations on top of TAO results in a QoS enabled middleware but extra overhead is introduced. However, the overhead that CIAO introduces compared to TAO is minimal and slightly affects throughput, jitter, and latency (Wang *et al*., 2004), which are of great importance to many control systems. The fact that the overhead introduced by CIAO compared to TAO is minimal makes CIAO suitable for control applications with real-time constraints.

A daemon process of CIAO is started on each target node to start the required component server. For the execution of the control application the following process is applied on the implementation artefacts that are produced by Archimedes ESS. The .idl and .cidl files are processed by idl and cidl compilers to generate stub, skeleton and component implementation skeleton files, which are subsequently used to generate the component's dlls. These dlls along with .csd and .ssd files constitute the component's packages. Component packages with the .cad file are utilized by specific deployment tools to initialize the component instances, create the connections and initialize the control application in a user friendly way.

5. CONCLUSIONS

An approach for the construction of IEC61499 FB-based design models and their subsequent transformation to CCM implementation artefacts was described, and a prototype ESS, namely Archimedes ESS, was presented. Archimedes demonstrates the applicability of the proposed approach and greatly simplifies the development process of distributed control applications. GME, the meta modelling toolset that was utilized to create Archimedes, was proved very helpful in the iterative development process of our modelling approach. It supports the rapid design of the modelling language and its immediate use, so hands-on experience was easily obtained for several alternatives. The key advantage of using GME, was that the effort needed for the development of Archimedes was orders of magnitude less than developing a custom-made toolset such as CORFU FBDK. However, Archimedes is very difficult to fully reach in some areas, the features of an environment, such as CORFU FBDK, that was specifically developed for the control and automation domain. Archimedes makes the development process of control systems: a) more reliable since many critical QoS aspects, such as concurrency, distribution, transactions, security and dependability are handled by already existing components, and b) easier, since it separates the handling of QoS aspects of the control application from its functional aspects.

REFERENCES

Gokhale, A., D. C Schmidt, B. Natajaran and N. Wang (2002). Applying Model-Integrated Computing to Component Middleware and Enterprise Applications. *Comm. of the ACM*, **45**, No. 10, 65-70.

Heck, B. L. Wills and G. Vachtevanos (2003). Software Technology for Implementing Reusable, Distributed Control Systems. *IEEE Control Systems Magazine*, **23**, 21-35.

Ledeczi, A., M. Maroti, A. Bakay , G. Karsai, J. Garrett, C. Thomason , G. Nordstrom, J. Sprinkle and P. Volgyesi (2001). The Generic Modeling Environment. *Proc. of WISP'2001,* Budapest.

Lu, T., E. Turkaye, A. Gokhale, D. C. Schmidt (2003). CoSMIC: An MDA Tool Suite for Application Deployment and Configuration. *ACM OOPSLA Conference*, Anaheim.

Schmidt, D.C., D. L. Levine, S. Mungee (1998). The Design of the TAO Real-Time Object Request Broker. *Computer Communications,* **21**, No. 4, 294-324.

Schmidt, D.C. and S. Vinoski (2004). The CORBA Component Model, Part3: The CCM Container Architecture and Component Implementation Framework. *C/C++ Users Journal,* April 2004.

Thramboulidis, K. (2004). Model Integrated Mechatronics: An Architecture for the Model Driven Development of Mechatronic Systems. *IEEE Int. Conf. on Mechatronic*s, Istanbul.

Thramboulidis, K., C. Tranoris (2004). Developing a CASE Tool for Distributed Control Applications. *The International Journal of Advanced Manufacturing Technology*, **24**, Number 1-2, 24-31, Springer-Verlag.

Thramboulidis, K., G. Doukas, A. Frantzis (2004). Towards an Implementation Model for FB-based Reconfigurable Distributed Control Applications. *7th IEEE Intern. Sym. on OO Real-time Distributed Computing*, Vienna.

Wang, N., D. C. Schmidt and C. O'Ryan (2001). Overview of the CORBA Component Model. In: *Component-based software engineering: putting the pieces together* (Heineman G. T., Bill Councill. W. T. (Ed)), pp. 557-571. Addison-Wesley, Massachusetts.

Wang, N., C. Gill, V. Subramonian and D. C. Schmidt (2004). Configuring Real-Time Aspects in Component Middleware. *Conference on Distributed Objects and Application*, Cyprus.