Tools Integration through a Central Model and Automatic Generation of Multi-Platform Control Code

Marco Colla
SUPSI - University of Applied Sciences of Southern Switzerland
CH-6928 Manno, Switzerland
marco.colla@supsi.ch

Tiziano Leidi
SUPSI - University of Applied Sciences of Southern Switzerland
CH-6928 Manno, Switzerland
tiziano.leidi@supsi.ch

Abstract

The increasing complexity of automation systems and the great variety of platforms and languages favor an abstract approach to the design of control applications, which should also integrate mechanical, electrical and process information. A European project tackled this problem, and for achieving the goal of integrating engineering activities and tools proposed a link between the design and implementation phases by using a model-based repository, supported by automatic generation of control code for various environments. This paper focuses on some project results particularly regarding the generation of multi-platform control code.

Even if the project outcome has confirmed the feasibility of the proposed approach and generated code has been successfully used by project partners, some research questions are still worth to be investigated.

1. Introduction

In the industrial automation field, control applications can be run on various hardware platforms, from classical Programmable Logic Controllers (PLC) with a closed architecture to open Programmable Automation Controllers (PAC), from ad-hoc developed embedded systems to standard PC hardware architectures down to Field Programmable Gate Arrays (FPGA). The number of programming languages is accordingly wide; the authors grouped them into three main categories:

- industrial automation specific languages, like the ones standardized in the IEC 61131-3 norm [1] or the Function Block based IEC 61499 [2]
- generic computer programming languages, either procedural (C) or object-oriented (Java)
- hardware description languages, e.g. Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) [3].

In consequence, the implementation of the control solution for a given problem, and the required skills, greatly varies depending on the chosen hardware, run-time environment and programming language.

On the other side, the growing number of sensing and acting points of a mechanical device or plant, and the increasing complexity of the related functionality aggravate the task of the control engineer.

In order to reduce the hurdles and risks of implementing a control solution by manually writing code, various design approaches and tools can be used to tackle the control problem at a more abstract level.

On the previous themes, a survey and analysis of current practices and needs has been performed by the authors [4] on a limited number of industrial enterprises of different size and from various sectors. It showed that the step from the design specification to platform specific implementations of control applications is often done by hand, and rules for doing such conversion are seldom codified, often relying on the programmer's skill.

To overcome this situation, the European project MEDEIA [5], Model driven Embedded systems Design Environment for the Industrial Automation sector, was conceived and led by the Austrian company Profactor GmbH with support of eight partners. The automatic linking of design and implementation phases through a central model-based repository supporting, by contrast to comparable approaches, various tools, methods and languages was the main goal.

The automatic generation of the control code out of the information contained in the repository is necessary to achieve this goal. Several hardware platforms and programming languages have been considered, and a number of transformations have been implemented according to the industrial partners' needs.

This paper reports some information and results from such experimental development, particularly regarding control code generation out of a central model-based repository. Section 2 introduces more details about some project objectives, while section 3 analyzes other approaches from the industry and research domains. Section 4 details the code generation process adopted by the project, section 5 introduce considered languages and platforms, and section 6 ends up the paper resuming conclusions and further research issues.

2. MEDEIA integration approach

Engineering of industrial automation systems requires knowledge from and information transfer among different scientific disciplines. Moreover different application fields adopt various design methods and approaches, while solutions based on heterogeneous hardware and software platforms aggravate the implementation of control applications. To overcome these hurdles, specification of systems and of their functionality should be done at a high level of abstraction, different design techniques should be applicable, and the integration of these data and the implementation on different platforms using various languages should be automatic [6].

The MEDEIA project set various goals, covering also diagnostic, simulation and verification aspects. In this presentation we introduce the ones that support the bidirectional flows depicted in Fig. 1, which shows some considered design techniques on the left and some implementation languages on the right, namely:

– a formal framework for model-driven component-based development of control applications
– a modeling approach that considers design methods and tools currently used by experts from different domains
– automatic platform specific code generation.

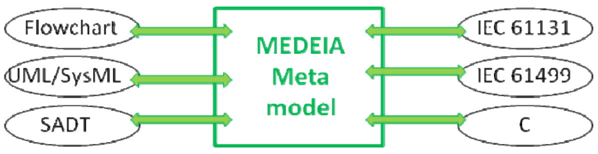

Figure 1. MEDEIA approach.

2.1. Formal framework

The main objective of MEDEIA was the development of a formal framework for integrating the various heterogeneous design and implementation models and tools existing in the industrial automation field.

For this reason the environment depicted in Fig. 2, named Design and Engineering Framework (MDEF), has been defined. It consists of a project manager and various model editors, transformers, simulators, verifiers and code generators that access a central data repository through a service manager. The whole framework integrates various software applications, some already existing on the market and others developed ad hoc.

MEDEIA supports the usage of already existing formal description methods through automatic model transformations to and from the central repository.

Moreover MEDEIA developed a set of code generators that navigate the information contained in the central repository and automatically generate the artifacts for the chosen platform and programming language. This way, systems from different vendors and various languages can be supported inside a single company or application, and existing applications can be reused after platform changes.

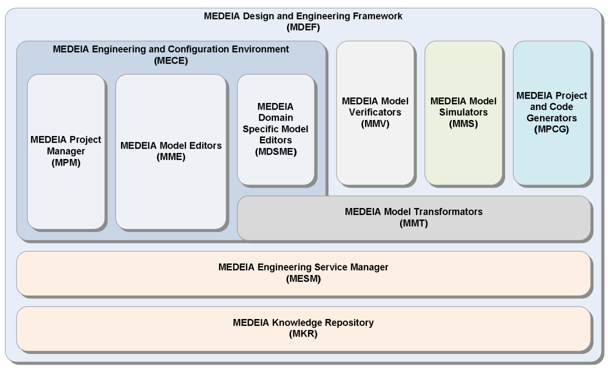
MEDEIA Design and Engineering Framework (MDEF)
MEDEIA Engineering and Configuration Environment (MECE)
MEDEIA Project Manager (MPM)
MEDEIA Model Editors (MME)
MEDEIA Domain Specific Model Editors (MDSME)
MEDEIA Model Verificators (MMV)
MEDEIA Model Simulators (MMS)
MEDEIA Project and Code Generators (MPCG)
MEDEIA Model Transformators (MMT)
MEDEIA Engineering Service Manager (MESM)
MEDEIA Knowledge Repository (MKR)

Figure 2. MEDEIA framework.

An Engineering and Configuration Environment (MECE) coordinates the previously listed applications.

The MECE is based on the open source Eclipse [7] project and on various plug-ins extensions. This choice has been motivated by the diffusion of this environment, by the growing number of projects and plug-ins that allow an easy integration of ready-to-use functionalities, and last but not least because it is open and free.

2.2. Central data repository

The information in the central repository is organized using various meta-models. At the basis of the architecture is the Automation Component (AC) meta-model, which represents the control functionality [6].

In the MEDEIA context, a component is similar to an integrated circuit (IC) in the electronic domain. The description of an AC is made up of its interface to other components or the physical environment through custom and plant ports, and of its internal, hidden contents and behavior. The behavior of basic components can be described e.g. using timed state charts, while bigger components and whole applications can be created by hierarchical composition, as shown in Fig. 3.

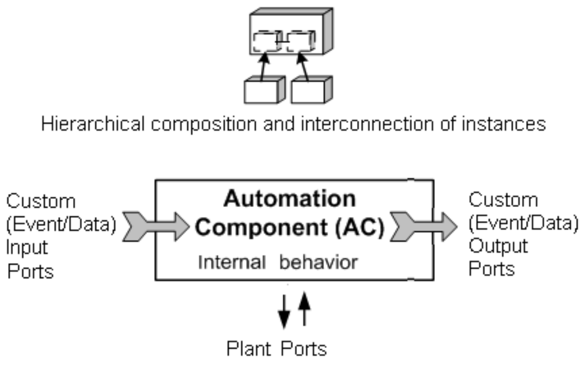
Hierarchical composition and interconnection of instances
Custom (Event/Data) Input Ports
Automation Component (AC)
Internal behavior
Custom (Event/Data) Output Ports
Plant Ports

Figure 3. AC concept.

Components can produce and consume events and data, contributing to build an event-driven architecture, and their instances can be connected together using port connections to form bigger ones. A flow-based paradigm is hence used: black boxes (ACs) build networks for exchanging information through connections.

This way the control designer can focus on the functional properties of the single components independently from the chosen technical architecture.

Other meta-models allow describing the real hardware of a specific implementation, and how the components instances inside an application map to the various physical devices and I/Os. Further meta-models cover diagnostic information for failure detection, identification and handling, and the description of the plant architecture for simulation purposes, while mechanical and electrical aspects can be added too.

3. Comparable initiatives

In the industrial automation field many research activities have been already undertaken or are currently running, e.g. the CESAR project [8], with goals similar to MEDEIA. Some commercial products on the market also present some similarities. Nevertheless, generally they focus either on the design or on the implementation side, and consequently adopt different solutions for the data repository and the code generation process.

At the University of Patras, Greece, the Software Engineering Group led by Prof. Thramboulidis [9] is active since more than ten years on various aspects regarding the model-driven development of component-based mechatronic systems, particularly focusing on UML [10] and SysML [11] for the design, and on the IEC 61499 standard as implementation platform [2].

A similar effort, oriented this time towards the integration of UML with the IEC 61131-3 standard, has been particularly pursued by Witch and Vogel-Heuser [12], leading to the new object oriented 61131-3 version.

The Department of Automatic Control and Systems Engineering at the University of the Basque Country, Bilbao, Spain, also fosters since many years a component-based approach for implementing industrial control systems [13]. They developed a markup language for industrial control systems called icsML whose software architecture closely follows the model of IEC 61131-3. This way the data transformation from icsML to the IEC 61131-3 platform is straightforward and can be accomplished through eXtensible Markup Language XML [14] based technologies like the eXtensible Stylesheet Language Transformation XSLT [15].

At the industrial level, the AutomationML organization [16], founded by four leading industries in the automation field, promotes a free, open and XML based intermediate data format. The goal is to permit the data exchange and synchronization among existing tools for all phases of plant engineering, covering plant topology, geometry, kinematics, motion planning and control behavior [17]. Regarding the logic description, AutomationML introduces an Intermediate Modeling Layer (IML) for enabling a two step data translation from various inputs, like e.g. Gantt charts or Impulse diagrams, to languages specific to PLC platforms. With this respect, AutomationML precisely focuses on Sequential Function Charts (SFC) and adopts the PLCopen XML data format [18], whose purpose is to allow the exchange of programs among tools from different vendors. As a direct consequence, IML is comparable to SFC, and the transformation between the two formats is straightforward.

To terminate this partial review of comparable initiatives it is worthwhile to cite Simulink [19] that, starting from Simulink models, Stateflow charts [20], or Embedded MATLAB functions [21], can generate code for different platforms and languages. A wide family of Coder products permits to create C, C++, VHDL and even IEC 61131-3 Structured Text supporting PLCopen XML as also many other vendor specific data formats.

Aside this commercial product, which supports different implementation platforms and languages starting from an own model based approach, the other activities previously presented mainly focus on one single specific language. Also for the modeling phase a single method or formalism is usually supported, except in case of AutomationML. As a consequence, the intermediate model between the modeling and the implementation levels usually can be similar to the one of the destination language. A transformation of models is then sufficient to accomplish the step towards the implementation, and a real code generation is not required. By contrast MEDEIA supports various methods, formalisms and languages for both the modeling and implementation phases. As a consequence, the difference between the central component-based model and a given implementation language can be large, due to the many-to-many relationship between the modeling and implementation environments.

4. MEDEIA code generation process

Even if the MEDEIA approach turns round the concept of the Automation Component, the AC meta-model is just one of the meta-models used in the MEDEIA repository for organizing information.

For automatically generating code such meta-models must be navigated and information must be collected, grouped, integrated with further constructs or modified accordingly to the selected platform and language. The navigation of different models by the code generators would be particularly difficult and heavy indeed. Moreover in the various meta-models the same concept, e.g. the internal behavior of a component, can be described with different methods. If code generators would directly access such different ways of information representation, then their implementation should cover a matrix of N description methods and M specific platforms. For this reason an additional internal model, called Platform Oriented Merged Automation Component Model (POMAC) has been introduced for concentrating the necessary information, reducing the complexity and increasing the efficiency of the code generation process.

4.1. Intermediate meta-model

The POMAC can be seen as an intermediate layer that links the bigger and numerous MEDEIA central meta-models to the various code output formats. Various characteristics and sometimes diverging peculiarities of the selected platforms have been considered, and the information inside the POMAC model has been consequently structured for allowing the various code generators to adequately convert it. Also the naming of the single model elements was chosen in order to be neutral to the various platforms terms, which sometimes identify different concepts with the same name. An ad hoc model transformer selects, merges and unifies just the necessary information that is contained in the various models, particularly in the AC, Mapping and Execution System ones, and put it in the POMAC. The resulting information flow is shown in Fig. 4.

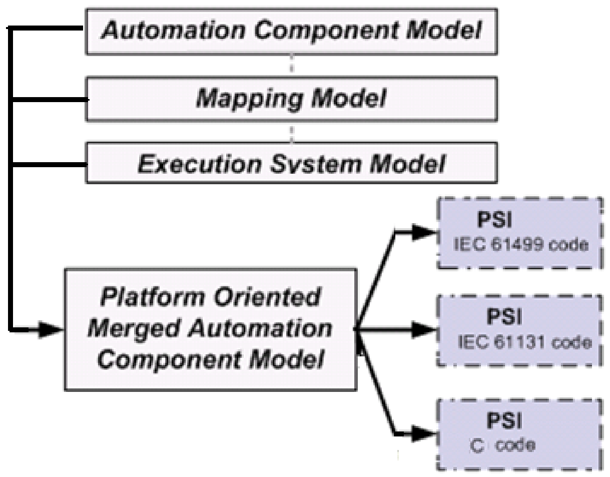

Figure 4. Code generation flow.

The POMAC consists of elements, their properties, relationships and rules used for automatic validation of the models. The meta-information inside can be grouped in the following main categories:

– Event, Variables and Complex elements as a combination of events and variables
– Components
– Component Aliases
– Component, Parameter and Plant Links for connecting the components interfaces
– Distribution
– Documentation.

The functional aspects of control applications are built around the component concept, whose internal behaviour is either described through state charts and actions triggered by events, or through event and data flows among contained components.

Aliases for a component and its interface elements are useful when on a platform a predefined implementation already exists, e.g. in library, in order to convert the neutral POMAC definition avoiding to generate code. Timer components are an example of application of aliases: one defines language independent timer functionality in the model, and then specifies aliases for e.g. IEC 61131-3 and IEC 61499 implementations.

The distribution category contains the configuration of an execution system made of applications, hardware devices types and instances, I/Os, tasks running on the devices, network connections, and the mapping of application elements on the physical and logical ones.

The last category, documentation, contains the information for specifying non functional properties like e.g. external behaviour of component interfaces (transaction sequence, timing, quality of service ...).

XMI (XML Metadata Interchange) format [22] has been used for storing the metadata information of the POMAC. Interchangeable code generators, as also additional graphical and textual editors for direct information editing, have been then developed and integrated inside the MDEF environment.

4.2. Code generation technique

For flexibly driving the transformations and for supporting the management of automatic operations related to generation (e.g. creation of projects and files) a generation engine approach has been implemented. This choice is due to the complex kind of the needed transformations, which include nonlinear generation phases where model objects are related to resulting artifacts in a many to many relationship. This approach was also well suited for handling the large amount of variability when generating source code for computer programming languages like C. The generation engine is supported by the JET (Java Emitter Template) language [23] contained in the EMF (Eclipse Modeling Framework) [24] plug-in of the Eclipse development environment. JET uses a subset of the JSP (Java Server Pages) syntax [25] for creating template implementation classes, in this case Java classes. Such classes have been used for generating small pieces of code made of static parts, written in the same way they have to be reproduced, and dynamic ones generated using Java code, embedded in the JET template, which queries the model. This way the generator engines are made of Java files, written by hand or built through templates without using complicated scripting languages, which query the model, load the elements and produce the code. This approach simplifies the navigability and maintenance of the code generator during refactoring phases too.

5. Languages and platforms

According to the interests of the project partners, the following four languages have been considered for automatic code generation, and the related transformations rules have been defined:

- IEC 61131-3
- IEC 61499
- C
- VHDL.

For C and VHDL no code generators have been written during the project due to time constraints.

5.1. IEC 61131-3

The concept of component, either basic or composite, can be mapped to the concept of Function Block (FB) in IEC 61131-3. By contrast events, not present in IEC 61131-3, have been converted both from the syntactic and semantic points of view using a BOOL type and generating extra code for preserving the transient character of an event. Complex ports have been converted to structures of data.

Among the five IEC 61131-3 languages, for the behavior of basic components SFC initially seemed the ideal choice due to its graphical aspect that resembles a state chart. Nevertheless some problems arose, due to some limitations in the tools implementations and in the language itself. For these reasons SFC has been discarded in favor of Structured Text (ST).

The IEC 61131-3 language that naturally lend itself for expressing the behavior of composite ACs and applications was the Function Block Diagram (FBD). The conversion from the POMAC to FBD was straightforward; nevertheless some differences and limitations arose among the considered platforms.

Just a subset of the information in the POMAC about the execution system and the mapping can be used. In IEC 61131-3 the available concepts of configuration, resource, task, global variable, access paths or communication FBs are more restrictive, and some platform implementations set further limitations.

Code generators for three platforms using different persistence formats, all based on the eXtensible Markup Language (XML) [14], have been implemented and tested. No one of the platforms directly uses the XML format standardized by PLCopen [18]. Nevertheless, the first one, logi.CAD from logi.cals [26], can convert programs to and from PLCopen XML. The second one, Orchestra Logic Programmaing from Sintesi [27], even if it claims to offer this conversion functionality uses a slightly modified version of the PLCopen format. The third platform, UnityPro from Schneider Electric [28], doesn't offer conversion support at all, and code for its proprietary format is directly generated.

5.2. IEC 61499

The IEC 61499 model is different and more formal than the 61131-3 one, nevertheless the concept of component, either basic or composite, can be mapped to the concept of Function Block (FB) in IEC 61499.

As the IEC 61499 model is event based, if the definition of a component doesn't contain events default ones are automatically generated and connected.

The code generated for actions inside basic FBs is ST.

In 61499 the information about the management of a control application and its interfacing with the run-time environment, e.g. I/O mapping or timing of execution, has to be included into the application. Hence extra elements, called Service Interface Function Blocks (SIFB), are automatically generated and parameterized e.g. with the physical I/O addresses and the plant ports of the components are connected to them.

XML is used for persistence format as defined by the IEC standard [2], Part 2. The generated code has been imported in two different open source runtime environments, 4DIAC [29] and FBDK [30], and used for experimental validation tasks in the robotic domain.

6. Conclusions

Industrial partners have successfully employed the whole MEDEIA framework, e.g. in a manufacturing cell, for semi-carter production for motorbikes, composed of two complex machine tools, one common tool storage unit, two pallet changers, one robot and one conveyor belt. Code generation starting from a component-based event-driven model, most control engineers were scared about, demonstrated to be effective. Reusability of ACs and time efficiency rated the highest satisfaction scores (9/10). Process traceability and quality of generated code rated 7/10. An assessment of the methodology over ten years estimated a benefit-cost ratio of 2.2 and a reduction of work hours of about 20%.

The transformation for different IEC 61131 platforms showed that, in spite of standardization, considerable differences exist among the various implementations, particularly regarding graphical languages. Often the possibility of distributing applications, expressed in term of number of supported configurations and resources, is limited. At the end, just some tools support PLCopen XML, or derivatives of it, as exchange format.

6.1. Further research and developments

The authors are considering if a pure component-based model and the related composition approach is appropriate versus a class-based one, or a mix of the two. The concept of component is a highly debated one, from its formal definition of Szyperski [31] as unit of independent deployment and of third-party composition, to the distinction between source and binary level underlined by Thramboulidis [32]. Defining a control block as a class with methods and calling them inside another block, instead of connecting the interfaces of the two blocks, could offer greater design flexibility. Of course this approach applies if the two blocks instances cannot be distributed or deployed independently, otherwise components are more suitable.

From the industrial partners, two main suggestions emerged after the validation phase: first of all the possibility in AC of integrating functionality from a parent. The current version of the AC meta-model lacks of the inheritance concept indeed.

Secondly the possibility to properly instrument generated code in order to support e.g. application debug. Currently the transformation rules are documented but fixed in the generators Java source code.

The authors of the MEDEIA tool chain are now evaluating the best way for further developing and exploiting the proposed approach.

Acknowledgements

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° ICT 2007-211448.

References

[1] International Electro-technical Commission, (IEC), "*Programmable controllers - Part 3: Programming languages, International Standard IEC61131-3*", Second Edition 2003.

[2] International Electro-technical Commission, (IEC), "*Function Blocks, Part 1 – 4, International Standard IEC61499*", First Edition 2005.

[3] Electronic Design Automation (EDA) Industry Working Groups, http://www.vhdl.org/.

[4] M. Colla, T. Leidi, M. Semo, "Design and Implementation of Industrial Automation Control Systems: a Survey", *Proceedings of 7th IEEE International Conference on Industrial Informatics (INDIN '09),* pp. 570-575, 2009.

[5] MEDEIA: Model driven Embedded systems Design Environment for the Industrial Automation sector, ICT Project Nr. 211448, 7th EU FP, http://www.medeia.eu/.

[6] T. Strasser, M. Rooker, I. Hegny, M. Wenger, A. Zoitl, L. Ferrarini, A. Dedé, M. Colla, "A Research Roadmap for Model-Driven Design of Embedded Systems for Automation Components", *Proceedings of 7th IEEE International Conference on Industrial Informatics (INDIN '09),* pp. 564-569, 2009.

[7] The Eclipse open development platform, http://www.eclipse.org.

[8] CESAR: Cost-efficient methods and processes for safety relevant embedded systems, ARTEMIS Joint Undertaking, http://www.cesarproject.eu/ .

[9] Prof. K. Thramboulidis, University of Patras, Greece, http://seg.ee.upatras.gr/thrambo/dev/index.htm.

[10] UML: the Unified Modeling Language, http://www.uml.org/#UML2.0.

[11] SysML: the Systems Modeling Language, http://www.sysml.org/.

[12] D. Witsch, B. Vogel-Heuser, "Close integration between UML and IEC 61131-3: New possibilities through object oriented extensions", *Proceedings of IEEE International Conference on Emerging Technologies and Factory Automation (ETFA '09)*, pp. 1-6, 2009.

[13] E. Estévez, M. Marcos, D. Orive, "Automatic generation of PLC automation projects from component-based models", *The Int. Journal of Advanced Manufacturing Technology*, Springer London, Vol. 35, Nb. 5-6, pp. 527-540, December 2007.

[14] XML: eXtensible Markup Language, http://www.w3.org/XML/.

[15] XSLT: eXtensible Stylesheet Language Transformation, http://www.w3.org/TR/xslt.

[16] AutomationML, http://www.automationml.org/.

[17] A. Lüder, E. Estévez, L. Hundt, M. Marcos, "Automatic transformation of logic models within engineering of embedded mechatronical units", *The Int. Journal of Advanced Manufacturing Technology*, Springer London, Vol. 54, Nb. 9-12, pp. 1077-1089, June 2011.

[18] PLCopen Technical Committee 6, "XML Formats for IEC 61131-3", *Technical Paper*, Version 2.01, 2009.

[19] Simulink: Simulation and Model Based Design, http://www.mathworks.com/products/simulink/.

[20] Stateflow: State Machines and Flowcharts, http://www.mathworks.com/products/stateflow/.

[21] Using the MATLAB Function Block, http://www.mathworks.com/help/toolbox/simulink/ug/f6-6010.html.

[22] XMI: Xml Metadata Interchange, http://www.omg.org/spec/XMI/.

[23] JET: Java Emitter Template, http://www.eclipse.org/modeling/m2t/?project=jet#jet.

[24] EMF: Eclipse Modelling Framework project, http://www.eclipse.org/modeling/emf/.

[25] JSP: JavaServer Pages Technology, http://www.oracle.com/technetwork/java/javaee/jsp/index.html.

[26] Logi.CAD automation tool from logi.cals, http://www.logicals.com/products/logi.CAD/.

[27] Orchestra Control Engine by Sintesi SpA, http://www.orchestracontrol.com/Home/tabid/36/Default.aspx.

[28] Unity Pro from Schneider Electric, http://www.schneider-electric.com/site/home/index.cfm/ww/.

[29] 4DIAC, open source for Distributed Industrial Automation, http://www.fordiac.org/.

[30] FBDK, the Function Block Development Kit, http://www.holobloc.com/doc/fbdk/.

[31] C. Szyperski, *Component Software: Beyond Object Oriented Programming*, Addison-Wesley, 2002.

[32] K. Thramboulidis, "The Function Block Model in Embedded Control and Automation from IEC61131 to IEC61499", *WSEAS Transactions on Computers*, Issue 9, Volume 8, September 2009.