Generating VHDL Source Code from UML Models of Embedded Systems

Tomás G. Moreira[1], Marco A. Wehrmeister[2], Carlos E. Pereira[3], Jean-François Pétin[4], Eric Levrat[4],

[1] Informatics Institute, [2] Dept. Electical Engineering,
Federal University of Rio Grande do Sul – Porto Alegre, Brazil
tgmoreira@inf.ufrgs.br, cpereira@ece.ufrgs.br
[3] Department. of Computer Science
Santa Catarina State University – Joinville, Brazil
marcow@joinville.udesc.br
[4] Centre de Recherche en Automatique de Nancy
University of Nancy – Vandoeuvre-Lès-Nancy, France
{jean-francois.petin, eric.levrat}@cran.uhp-nancy.fr

Abstract. Embedded systems' complexity and amount of distinct functionalities have increased over the last years. To cope with such issues, the projects' abstraction level is being continuously raised, and, in addition, new design techniques have also been used to shorten design time. In this context, Model-Driven Engineering approaches that use UML models are interesting options to design embedded systems, aiming at code generation of software and hardware components. Source code generation from UML is already supported by several commercial tools for software. However, there are only few tools addressing generation code using hardware description languages, such as VHDL. This work proposes an approach to generate automatically VHDL source code from UML specifications. This approach is supported by the GenERTiCA tool, which has been extended to support VHDL code generation. To validate this work, a use case focused in maintenance systems attended by embedded systems is presented.

Keywords: Embedded systems, system engineering, intelligent maintenance, UML specification, VHDL code generation.

1 Introduction

Embedded systems are dedicated system designed to perform a small number of functions. It contains predominantly digital components, consisting in a hardware platform upon which software application execute [1]. Embedded systems' functionalities can be distributed over different processing nodes. Distributed Embedded Systems (DES) rely on a communication infrastructure constrained by requirements/constraints of embedded systems domain, e.g. timing and energy consumption requirements.

In the industrial domain, DES may be composed by several intelligent components, which make decisions and perform their activities autonomously [2]. Industrial DES support conventional or innovative functions. The former concerns to simple control functions, whereas the later represents more elaborated functions, e.g. maintenance and prognostic systems' functions to perform Condition Monitoring (CM), Health Assessment (HA), Prognostics (PR), etc. Components' intelligence level is defined by the amount of different services required by the end-user that are implemented as component functions [3], [4]. As machines do not suddenly fail, they usually pass through a measurable process of degradation before failing. Intelligent maintenance or prognostics systems use information provided by sensors and computing components embedded into equipments. Then, algorithms for health estimation and failure prediction are applied to assess machines' degradation level. Hence, embedded sensors, intelligent actuators and processing elements play a fundamental role in the development of intelligent maintenance systems.

The complexity of this scenario demands new tools and techniques. Increasing the design's abstraction level by using, for instance Model-Driven Engineering (MDE) [6] techniques, is an interesting approach to deal with the mentioned issues [5]. Standard graphical languages, e.g. the *Unified Modeling Language* (UML)[1], must be used to facilitate the communication of design's intentions to different teams, i.e. software and hardware teams. UML is a high-level design language and is broad enough in scope to model DES. Usually, UML-based MDE approach focus only in the software part of embedded systems by defining a mapping between high-level specification's construction down to software construction using programming languages (e.g. C/C++, Java, etc.). There are many academic works and commercial tools (e.g. Rational Rose[2] and Artisan Studio[3]) that can generate software code from UML models. However, considering embedded systems' hardware part, only few works address the use of UML to produce *Hardware Description Language* (HDL) descriptions, as in [9], [10]. In this sense, the transformation of UML models into HDL code, e.g. using VHDL (*Very High Speed Integrated Circuit HDL*), is not yet well diffused, opening room for research on this subject.

This works presents an extension to our aspect-oriented MDE approach for DES named *Aspect-Oriented Model-Driven Engineering for Real-Time systems* (AMoDE-RT) [7]. This work's main contribution is to support automatic generation of VHDL descriptions from UML models. In other words, this work extends AMoDE-RT's supporting tool GenERTiCA (*Generation of Embedded Real-Time Code based on Aspects*) [12], aiming at automatic generation of VHDL descriptions from UML models. The generated VHDL code is intended to be used in FPGA (Field Programmable Gate Array) systems. Thus, the proposed approach allows the behavior of the required system to be described (modeled) and verified (simulated) before synthesis tools translate the design into real hardware (gates and wires).

Additionally, this work proposes an engineering process, which covers from requirements analysis and UML modeling phases to VHDL code generation. The focus is to generate VHDL source code for the logical functions of an embedded system, which so far has only been implemented in software. To validate this work, this paper presents a use case focused on a distributed embedded system (i.e. DES)

[1] UML 2.2 Specification. Object Management Group, http://www.omg.org/ spec/UML/2.2/

[2] http://www.ibm.com/software/rational/

[3] http://www.artisansoftwaretools.com

used for maintenance systems (intelligent components), which integrates both conventional and innovative functions.

This paper is organized as follows: Section 2 provides an overview of the proposed approach to map UML specifications into VHDL source code, whereas Section 3 presents the developed mapping rules. The case study of a valve component system is presented in Section 4. Section 5 provides a review of works related. Finally, conclusion and directions for future work are presented in Section 6

2 Overview of the Proposed Approach

The proposed approach follows the flow proposed by the Aspect-oriented Model-Driven Engineering for Real-Time systems (AMoDE-RT) [7], [15] (see Fig. 1), which uses MDE techniques combined with AO concepts to design DES. AMoDE-RT is supported by GenERTiCA [12] code generation, which uses mapping rules scripts to produce source code files for a given target platform from UML models annotated with the MARTE profile[4]. Therefore, GenERTiCA is capable of generating code for many distinct languages (Java, C/C++, etc.), since there are mapping rules for the target platforms. The process is the same to generate code for different languages and therefore it is considered generic. This work proposes an extension for the GenERTiCA tool in terms of a new set of mapping rules to map UML meta-model elements into VHDL constructs.

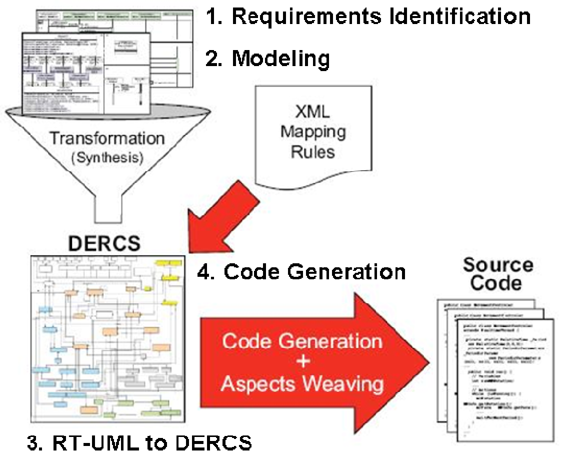


Fig. 1. General overview on the whole process.

Following, a brief description of each step of AMoDE-RT design flow is presented.

[4] UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE), http://www.omg.org/spec/MARTE/1.0

1. Requirements Analysis and Identification

In the first phase, requirements and constraints of the distributed embedded real-time system are gathered. To accomplish this, the RT-FRIDA [13] requirements analysis is performed, resulting in a set of documents describing system's requirements, functionalities and constraints. Afterwards, use case diagrams are created, depicting all expected functionalities of the distributed embedded real time system, and also the external elements that interact with the system.

2. Modeling

The next step is to specify the elements to handle the functional and non-functional requirements gathered in the previous phase. To model functional requirements, designers use class diagrams to describe the structure, and sequence diagrams to describe the methods behavior. Other behavioral and structural diagrams, such as activity or state diagrams, or composite structure or deployment diagrams, can also be used. However the class and sequence diagrams are mandatory to describe the structure and behavior of all system with correctness. These UML diagrams are annotated with the stereotype of the MARTE profile to specify real-time characteristics of (some) DES' elements. During this phase, the non-functional requirements handling are specified using aspects from the Distributed Embedded Real-Time Aspects Framework (DERAF) [12]. These aspects are modeled in the Aspects Crosscutting Overview Diagram (ACOD) [15], and the points (in the UML model) in which DERAF aspects perform adaptations are specified using Join Point Designation Diagrams (JPDD).

3. UML-to-DERCS Transformation

At this point GenERTiCA transforms the system specification, i.e. the UML model, into another model called DERCS (Distributed Embedded Compact Specification) [12], which represents an embedded system PIM free of information overlapping[5]. A UML specification can contain several model elements, representing the same element that hinders the code generation process. Thus these ambiguous elements of UML model are mapped in a single DERCS element, eliminating such ambiguities that could result in code with errors. When an inconsistency is detected, the UML-to-DERCS transformation algorithm stops and GenERTiCA informs this occurrence to the designer, requesting his/her intervention to solve the issue. Interested readers are referred to [12] to have more details on this UML-to-DERCS model transformation.

4. Code Generation

In this phase the code generation process executes a set of scripts (i.e. mapping rules), which guide the GenERTiCA tool to perform the model-to-text transformation from DERCS elements to constructions in the target platform. Furthermore, the code generation process also performs the aspects weaving. If the element under evaluation

[5] Information overlapping in UML models means the same feature of the target system, which has been specified using distinct diagrams depicting different viewpoints of the same structural/behavioral characteristic.

is affected by an aspect, the aspects weaving process modifies the generated code fragments according to aspects adaptations described in the mapping rules.

3 Mapping UML to VHDL

As previously mentioned, to generate code from the UML model, GenERTiCA adopts a script-based approach, in which small scripts define how to map model elements into target platform constructions, generating source code fragments that are merged to produce source code files. The script-based code generation process improves separation of concerns in mapping rules specification, as each script is concerned with the transformation of a single model element (or few of them) into source code fragment.

Table 1. Concepts Mapping.

UML Element	VHDL Element
Class	Entity-Architecture pair
Public attribute	Entity Ports
Private attribute	Signals
Methods	Processes
Events and Message exchange	Entity Ports
Associations between classes	Entity Ports
Inheritance	VHDL key word “new”
Static polymorphism	Configuration structure
Objects instantiation	Component structure

In this sense, this work has proposed a set of mapping rules to allow VHDL code generation from UML models, following GenERTiCA’s approach. Table 1 shows the mapping from UML concepts into VHDL ones. Scripts to accomplish these transformations have been developed and inserted in an eXtensible Markup Language (XML)[6] file, which guide GenERTiCA in the code generation process. Details on the created mapping rules are provided in the following sub-sections.

Classes are mapped into VHDL entity-architecture pairs. The class parameters are mapped to VHDL generic statements, while public attributes to VHDL entity ports and private attributes to VHDL signals. The methods are mapped to VHDL processes. The composition relationship, which describes the composition of a system from components, is mapped to a VHDL port map statements. Objects are instantiated as component structures into other entities. Events and Messages exchanges are implemented as entities ports that allow the communication between different entities and their processes (methods). Associations between classes are similar to the approach used to messages exchange, however component structures representing each associated-class (i.e. entities) are instantiated into the pair-class to accomplish the association by the mapping of signals between these two classes. Inheritance is

[6] eXtensible Markup Language (XML) 1.0 (Fifth Edition). http://www.w3.org/TR/2008/REC-xml-20081126/

obtained making the entity or architecture declaration with the VHDL key word “tagged”, which means that the declaration is valid but still incomplete. Then, we declare a new entity/architecture using the VHDL key word “new”, with the desired modifications. Static polymorphism is obtained using the VHDL configuration structure to bind the same VHDL component to different entities or architectures.

3.1 Mapping Rules

Mapping rules are specified as small scripts that create source code fragments (representing target platform constructions) from DERCS model elements. Source code files are made up from these generated code fragments. Scripts are stored and organized in one mapping rules file specified using the XML format. This XML file has a portable format allowing the specification of self-described content organized in a tree structure. XML tree organization facilitates scripts storage in terms of platform mapping rules repositories. It allows scripts to be reused in further projects that use the same target platform. Hence, the design effort to derive system implementation from an UML model is decreased.

Scripts are located in the leaf nodes of the tree. The correct script is selected based on which element is being accessed by the code generation algorithm (i.e. the leaf node must match with the DERCS element). The code generation algorithm first identifies the type of the element. Afterwards, it tries to find the tree’s leaf that better represents the type of such element being evaluated. Then, it executes the script contained within the found node (i.e. leaf), which will create a VHDL code fragment to that element being evaluated.

The language used to describe the scripts is the well-known open source scripting framework called Velocity[7], which defines the Velocity Template Language (VTL) that provides all functionalities required to assist the code generation approach implementation. VTL is a Java-like scripting language, which returns a string as result of script execution. Thus, the generated source code fragment is obtained by means of accessing model information through DERCS API.

To illustrate what is a script an example is given above. This script is responsible for the code generation to the classes’ methods. It generates one VHDL process for each method in the classes.

```
#if ($Message.Name != $Class.Name)
  \n${Message.Name}: process(
  #if ($Message.Name == "run")
      clock,
      reset,
  #end
  #if ($Message.ParametersCount > 0)
      #foreach( $param in $Message.Parameters )
          #if ($velocityCount > 1), #end
          $param.Name
      #end
  #end
  )
  \n$Options.BlockStart
  #if ($Message.Name == "run")
      \n
```

[7] http://velocity.apache.org

```
      \nif (reset='1') then
      \n-- variables initialization
      \n
      \nelsif (clock'EVENT and clock='1') then
  #end
  \n$CodeGenerator.getVariablesDeclaration(0)
  \n$CodeGenerator.getActionsCode(1)
  #if ($Message.Name == "run")
      \n
      \nend if;
  #end
  \n$Options.BlockEnd process;
  \n
#end
```

3.2 Concepts

The concepts used in this work to the transformation of UML structures into VHDL code are based in [9] and [14]. These works gave us some ideas on how to represent in VHDL the structure of classes, attributes, methods, association between classes, events and messages exchange, inheritance, and polymorphism.

UML Model is object-oriented, while the VHDL code is structured. This semantic gap between abstraction levels hinders the mapping between these two languages. It may be one of the main reasons why it is very rare, until the present days, to find works addressing to VHDL code generation from UML models and/or any commercial tool.

From these concepts some rules have been developed and tested to this version of the VHDL mapping rules. They represent the first version of the mapping rules to provide the VHDL code generation through GenERTiCA tool. Up to now, it has been developed and tested mapping rules to generate VHDL code from UML classes, attributes and behaviors. Their feasibility is shown in the case study of next section. The concepts of inheritance, polymorphism, associations between classes, etc, are being be implemented in the next version of this work.

4 Case Study

This section shows an example of automatic VHDL code generation from a UML model. The system under evaluation is composed by an automatic valve and the sensors that give information about the valve's states. This valve is used to regulate the water flow and is part of CISPI (*Conduite de grands systèmes industriels à risque*) experimental platform located at CRAN (*Centre de Recherche en Automatique de Nancy - France*). By applying the proposed approach, we intend to integrate new functionalities supported by a FPGA in this valve, leading to an implementation of an intelligent component. This intelligent component is part of a mechatronic system which also contains other mechanical, electronics and computational parts. The electronics and computational parts represent the control system, which is composed of logical functions executing in a hardware platform. Logical functions represent components' behavior and are usually implemented in software. This work aims at implementing these logical functions as hardware. Thus the system has been specified in UML and its implementation has been generated as VHDL code. The generated code represent the hardware description of the logical and the control system parts (logical functions), which is executed inside of a FPGA (hardware platform).

The development of this case study was compliant with the AMoDE-RT approach defined in the section III. Then, the first step was to gather the requirements and create a use case diagram to identify system's services and the actors that interact with the system. Only two actors were considered. The "User" actor request services to open and close the valve, while the "Maintenance Operator" actor requests services to know the number of times the valve performed opening and closing actions. Information about these numbers is used for assessing the component physical heath.

The second step was to create class and sequence diagrams. The class diagram has been built from the knowledge acquired in the use case. It represents the system's structure. All services have been modeled as classes. These classes work together to provide the system's services. They are enclosed by a main class responsible for all system. No DERAF aspects have been used to deal with non-functional requirements, and hence, no ACOD and JPDD diagrams have been created.

Services are modeled as sequences of actions in sequence diagrams. Sequence diagrams represent the exchange of messages between the objects that compose the system in order to represent the expected system behavior. Each service demanded by the actors results in the execution of one or many sequence diagrams. Fig. 2 depicts a part of the main sequence diagram of valve's system related to the service solicitations.

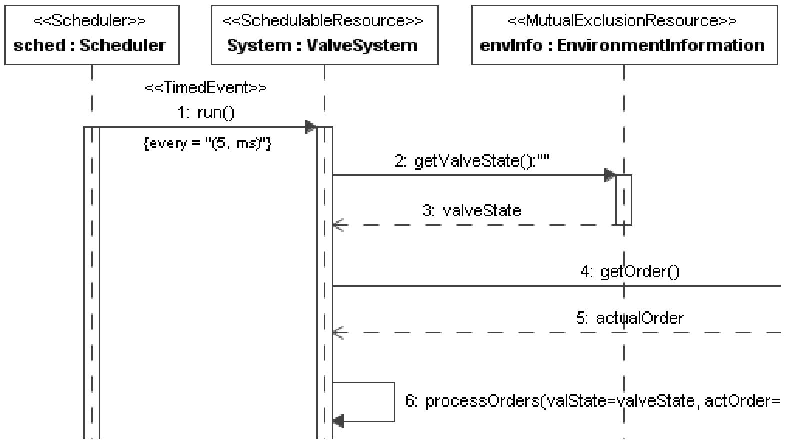
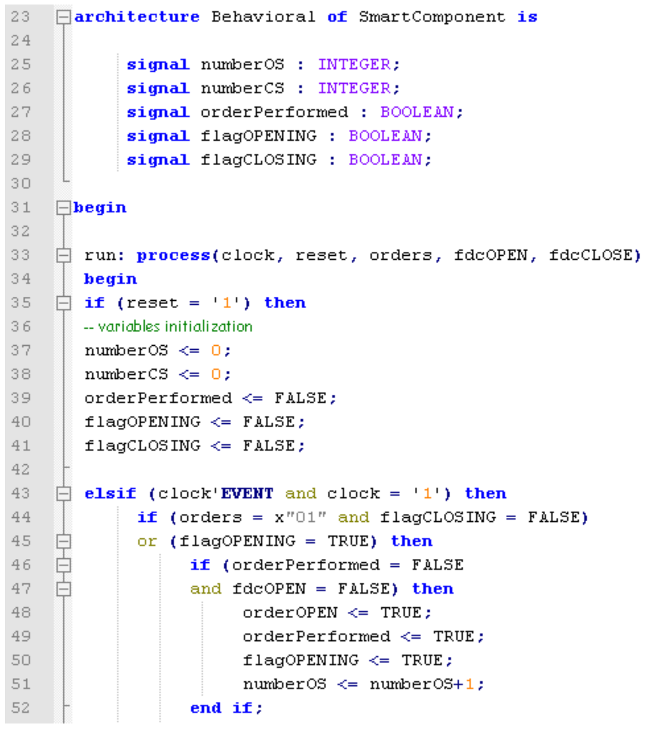


Fig. 2. Sequence diagram of the main function in the valve's system.

The rule implementing the association between classes was not developed in this first version of the mapping rules. Thus, all this case study was re-modeled as just one class and one method. From these new diagrams the feasibility of the system could be tested and the 3rd and 4th steps performed. The mapping has transformed the UML class and sequence diagrams into one single VHDL file, which contains an entity-architecture pair declaration. The UML class is represented by the entity-architecture pair, while the UML behavior (method) is represented by a VHDL process inside of the architecture.

The code that appears in fig. 3 is the result of the automatic code generation performed by the mapping rules proposed in this work. This first test generated code for only one embedded component. For this component have been automatically generated 80 lines of VHDL code, covering 100% of the needed code for the application. This component is simple and its model was developed on only one UML class, however the results are encouraging for future works.

The real feasibility of the VHDL code can be tested quickly, since this code can be synthesized in the resulting bitstream uploaded in a FPGA development platform, such as Virtex-II PRO (V2-Pro) development system by Digilent[8].



```
architecture Behavioral of SmartComponent is

     signal numberOS : INTEGER;
     signal numberCS : INTEGER;
     signal orderPerformed : BOOLEAN;
     signal flagOPENING : BOOLEAN;
     signal flagCLOSING : BOOLEAN;

begin

 run: process(clock, reset, orders, fdcOPEN, fdcCLOSE)
 begin
 if (reset = '1') then
 -- variables initialization
 numberOS <= 0;
 numberCS <= 0;
 orderPerformed <= FALSE;
 flagOPENING <= FALSE;
 flagCLOSING <= FALSE;

 elsif (clock'EVENT and clock = '1') then
      if (orders = x"01" and flagCLOSING = FALSE)
      or (flagOPENING = TRUE) then
           if (orderPerformed = FALSE
           and fdcOPEN = FALSE) then
                orderOPEN <= TRUE;
                orderPerformed <= TRUE;
                flagOPENING <= TRUE;
                numberOS <= numberOS+1;
           end if;
```

Fig. 3. VHDL code representing the architecture declaration and run method initial part.

5 Related Works

This section discusses some projects and commercial tools that propose transformations from UML specifications to source code VHDL. Among these works different approaches to generate source code from UML models have been found. Some use only one diagram (e.g. class diagram) to generate code, while others use a combination of distinct diagrams (e.g. class and state diagrams, sequence and/or

[8] Digilent Inc. www.digilentinc.com

activities diagrams) [8]. Thus the presented related works generate code ranging from classes skeletons to code containing system elements behavior.

In [9], a framework has been developed to derive VHDL specifications from UML's class and state diagrams. They use homomorphic mappings between dissimilar structures while preserving metamodel class associations in a way that resembles MDA technique. However, their generated VHDL code focuses on simulation and verification of UML models rather than on hardware synthesis.

An interesting work has been developed in [10], in which the MODCO tool is presented. It uses MDA techniques to define high-level model-based system descriptions that can be implemented in either hardware or software. Thus it can transform UML state diagrams directly into synthesizable VHDL. State machines in UML are used to describe the behavior of a part of a system. However, the complete code is generated for the behavior. Functional requirements are mapped to UML component, class, use case and state diagrams. Non-functional requirements are specified as UML annotations that describe performance constraints using property-value pairs defined by UML profiles. However, the authors targeted flat state-transition diagrams without supporting hierarchy and concurrency, and also only covering a small subset of UML state diagram constructs.

In [11], they have developed a framework for deriving VHDL specifications from UML state diagrams, and also a set of rules, which enable automated generation of synthesizable VHDL code from UML. Their engineering process is based on meta-models. Concepts of the UML state diagram metamodel are mapped onto concepts of the VHDL metamodel. There are two transformations between models, happening in the following way: the first transformation converts the main UML model into state diagram models; and the second one maps the state diagram models onto concepts in the VHDL language. A model-to-text transformation is used to generate synthesizable VHDL code from the VHDL model.

Two commercial tools to generate VHDL code could also be found. StateCAD[9] by Xilinx is a graphical entry tool for digital design that has its own graphical notation to represent state diagrams as bubble diagrams. StateCAD automatically generates HDL (VHDL and Verilog) code, for simulation and also synthesis, directly from these state diagrams. The other tool is Simulink HDL Coder[10] by MathWorks, which generates synthesizable VHDL and Verilog code from simulink models, stateflow charts, and embedded Matlab code. Simulink HDL Coder also generates simulation and synthesis scripts, enabling to simulate and synthesize quickly the developed design. These commercial tools do not generate VHDL code from UML specifications.

The approach proposed in [11] realizes the mapping between models, similarly to other MDE techniques. In [10], there is a separation of the functional and non-functional requirements in the modeling stage using distinct UML diagrams. These mentioned works are limited, because they cover a specific subset of the UML structures, and also use only the UML state diagrams. By using GenERTiCA the designer can use distinct UML diagrams, combining them to specify the full functionality in terms of structure, behavior and non-functional requirements handling, since some guidelines are followed. These guidelines are simple and

[9] http://www.xilinx.com/

[10] http://www.mathworks.com/products/slhdlcoder/

intuitive, allowing designers to separate the functional and non-functional requirements. Moreover, by using the extension proposed in this work, it is possible to generate code for VHDL and also for the Java and C++ languages from the same model.

6 Conclusions

This work addresses the problem of generating HDL descriptions from UML models. It presented the initial set of mapping rules to generate VHDL code from class and sequence diagrams, using GenERTiCA tool. To achieve this goal, is has been proposed a mapping from object-oriented concepts supported in UML into concepts used by VHDL. Then, a set of mapping rules (used by GenERTiCA to generate VHDL code) has been developed. These rules extend the functionality of GenERTiCA tool, allowing it to generate HDL code in addition to software source code. Besides, this paper has described all steps of the proposed approach that must be followed to generate automatically VHDL descriptions.

To demonstrate the proposed approach, a case study has been presented. It showed a small part of a maintenance system, i.e. automatic control of a valve implemented as a Smart Component. As mentioned mapping rules have been implemented and tested to produce VHDL code from UML's classes, attributes and behavior. Results shown that, for developing simple systems, 100% of the necessary code could be generated. Hence, despite the case study's size, the results are considered satisfactory since we see great potential to scale the approach to more complex systems.

To continue this work the following direction will be pursued: to complete the rules needed for the code generation of VHDL structures. The concepts of inheritance, polymorphism, associations between classes are very important to be able the modeling of complex systems; to develop the rules for the non-functional requirements implemented by aspects; to test and to prove the new rules; to perform more tests with FPGA boards; to apply the proposed approach in a complex real system, such as an industrial maintenance and prognostic systems.

References

1. Micheli, D., Gupta, R. K.: Hardware/Software Co-design. In: Proceedings of the IEEE, vol. 85, no. 3, pp 349--365 (1997).
2. Pereira, C. E., Carro, L.: Distributed real-time embedded systems: Recent advances, future trends and their impact in manufacturing plant control. In: Annual Reviews in Control, vol. 31, pp 81--92 (2007).
3. Iung, B., Neunreuther, E., Morel, G.: Engineering process of integrated-distributed shop floor architecture based on interoperable field components. International Journal of Computer Integrated Manufacturing, vol. 14, no. 3, pp 246--262 (2001).
4. Pétin, J-F., Iung, B., Morel, G.: Distributed intelligent actuation and measurement (IAM) system within an integrated shop-floor organization. Computers in Industry, vol. 37, pp. 197--211 (1998).

5. Sangiovanni-Vincentelli, A.: The tides of EDA. IEEE Design & Test of Computers, vol. 20, no. 6, pp. 59--75 (2003).
6. Mellor, S. J., Clark, A.N. Futagami, T.: Guest Editors' Introduction: Model-Driven Development. IEEE Software, vol. 20, no. 5, pp. 14--18 (2003).
7. Wehrmeister, M. A.: An Aspect-Oriented Model-Driven Engineering Approach for Distributed Embedded Real-Time Systems. PhD Thesis, Federal University of Rio Grande do Sul, Brazil, Apr. 2009, http://lisha.ufsc.br/~marcow/publications/wehrmeister_thesis_final.pdf
8. Long, Q. et al.: Consistent Code Generation from UML Models. In: Australian Software Engineering Conference, Los Alamitos (2005).
9. McUmber, W. E., Cheng, B. H.: UML-Based Analysis of Embedded Systems Using a Mapping to VHDL. In: 4th IEEE International Symposium on High-Assurance Systems Engineering, Washington D.C. (1999).
10. Coyle, F., Thornton, M.: From UML to HDL: a Model-Driven Architectural approach to hardware-software co-design. In: Information Systems: New Generations Conference (ISNG), Las Vegas (2005).
11. Wood, S. K., et al.: A Model-Driven development Approach to Mapping UML State Diagrams to Synthesizable VHDL. IEEE Transactions on Computers, vol. 57, no. 10, pp. 1357--1371 (2008).
12. Wehrmeister, M. A., Freitas, E. P., Pereira, C. E., Ramming, F.: GenERTiCA: A Tool for Code Generation and Aspects Weaving. In: 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (ISORC), Orlando (2008).
13. Freitas, E.P., et al.: DERAF: A high-level Aspects Framework for Distributed Embedded Real-Time Systems Design. Early Aspects: Current Challenges and Future Directions (Lecture Notes in Computer Science), pp. 55--74, Springer (2007).
14. Ecker, W.: An object-oriented view of structural VHDL description. In: Proceedings of VHDL International Users Forum Spring '96 Conference, Santa Clara (1996).
15. Wehrmeister, M. A., Freitas, E. P., Pereira, C. E., Wagner, F.: An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in Model-Driven Development of Distributed Embedded Real-Time Systems. In: 10th IEEE Symposium on Object Oriented Real-Time Distributed Computing, pp. 428--432 (2007).