An Infrastructure for UML-Based Code Generation Tools

Marco A. Wehrmeister[1], Edison P. Freitas[3], and Carlos E. Pereira[2]

[1] Instituto de Informática, [2] Dep. de Engenharia Elétrica
Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil
mawehrmeister@inf.ufrgs.br, cpereira@ece.ufrgs.br
[3] School of Information Science, Computer and Electrical Engineering,
Halmstad University, Halmstad, Sweden
edison.pignaton@hh.se

Abstract. The use of Model-Driven Engineering (MDE) techniques in the domain of distributed embedded real-time systems are gain importance in order to cope with the increasing design complexity of such systems. This paper discusses an infrastructure created to build GenERTiCA, a flexible tool that supports a MDE approach, which uses aspect-oriented concepts to handle non-functional requirements from embedded and real-time systems domain. GenERTiCA generates source code from UML models, and also performs weaving of aspects, which have been specified within the UML model. Additionally, this paper discusses the *Distributed Embedded Real-Time Compact Specification* (DERCS), a PIM created to support UML-based code generation tools. Some heuristics to transform UML models into DERCS, which have been implemented in GenERTiCA, are also discussed.

Keywords: UML, Aspect-Oriented Design (AOD), code generation, aspects weaving, distributed embedded real-time systems.

1 Introduction

The design of embedded systems is not a trivial task. The domain of embedded real-time systems presents many specific requirements (e.g. deadlines for tasks accomplishment, energy consumption, reduced footprint, etc.) that do not specify system's functionalities but are tightly related to them. Such requirements are called non-functional requirements. Traditional approaches, such as object-orientation or the structured analysis, do not have specific abstractions to deal with these requirements, whose treatment is usually found intermixed with the handling of functional requirements. This situation leads to problems such as tangled and scattered handling, which hinder the reuse of previously developed artifacts (e.g. models or code). To solve the above-mentioned problems, some proposals can be found in the literature, such as subject-oriented programming [2] and Aspect-Oriented (AO) programming [3], which provide special constructions to specify and encapsulate non-functional requirements handling into single elements.

However, solving these problems only at implementation level is not sufficient to deal with the mentioned complexity. Abstraction level increase is an old but widely accepted idea to help with such quest. Thus, embedded systems community is looking for new techniques/approaches, such as Model-Driven Engineering (MDE) [1], aiming at the management of complexity in the design of distributed embedded real-time systems. An important issue to allow the use of MDE in embedded systems design is tool support [1]. Automatic transformation from a Platform Independent Model (PIM) to a Platform Specific Model (PSM) is a key issue to make models the main artifact during the whole development cycle instead of source code. Additionally, it avoids errors coming from manual transformations, and also helps to keep specification and implementation synchronized. Code generation from high-level models can be seen as a transformation of PIM into PSM, but instead of using meta-model to meta-model transformations (i.e. transforming meta-model elements from a PIM into PSM meta-model elements), it applies the translation of meta-model elements into text representing source code in a target language.

The aim of this paper is to extend the discussion presented in [18] by providing details on key elements of a tool called *Generation of Embedded Real-Time Code based on Aspects* (GenERTiCA). GenERTiCA has been developed to support the *Aspect-oriented Model Driven Engineering for Real-Time systems* (AMoDE-RT) approach [4], which uses concepts from the AO paradigm to deal with non-functional requirements since early design phases. Besides code generation, GenERTiCA also performs aspects weaving into both the generated code and the input model to provide non-functional requirements handling as specified in aspects' adaptations. Therefore, in addition to the discussion on code generation and aspects weaving processes, this text also presents the *Distributed Embedded Real-Time Compact Specification* (DERCS), a PIM created to represent system's structure, behavior, and non-functional requirements handling (in terms of AO elements). Moreover, heuristics to transform UML into DERCS models are also discussed.

This text is organized as follows: section 2 provides details on DERCS; section 3 presents the transformation from UML to DERCS models; section 4 discusses GenERTiCA's code generation and aspects weaving processes; section 5 discusses some related works. Concluding remarks and future work are discussed in section 6.

2 Meta-model for Code Generation and Aspects Weaving

AMoDE-RT approach [4] uses UML as specification language for systems' PIM, representing its structure, behavior, and non-functional requirements handling in a platform independent fashion using aspects provided by the *Distributed Embedded Real-time Aspects Framework* (DERAF) [7]. Furthermore, designers use MARTE profile [8] to specify real-time features. UML has been chosen due to its acceptance as a standard, wide use in software community, and increasing interest by embedded systems community. Additionally, there are several academic and commercial CASE tools (supporting UML modeling) available to be used. However, for code generation purposes, UML has a weakness: its meta-model is huge and ambiguous, i.e. the same feature can be specified in distinct perspectives using distinct diagrams, which most often overlap each other making it a source of inconsistencies. Such inconsistencies could be partially verified using model consistency checking/testing techniques. However, despite its importance, this paper does not discuss model checking.

To address the mentioned specification problem, GenERTiCA transforms UML models into a new PIM called *Distributed Embedded Real-time Compact Specification* (DERCS). DERCS' meta-model is simpler than the UML one, but it provides the same information as UML models, i.e. system's structure, behavior and non-functional requirements handling, which is specified independently from functional requirements handling by using AO constructions. In addition, it is worth to mention that DERCS has been proposed to assist code generation tools design.

In DERCS, system structure is specified in the same way as in UML, i.e. objects are the fundamental entity. They represent system's elements, whose interaction and performed actions provide the expected functionality. Objects can be active or passive. The former kind represents objects that have their own execution flow (i.e. their own thread), such that they execute actions concurrently with other active objects. Usually, the active objects are compared to concurrent processes, and hence also have characteristics such as execution pattern (e.g. periodic, aperiodic, or sporadic), priority, deadlines, and others. On the other hand, passive objects execute actions sequentially as response to messages sent by other objects (passive or active). Additionally, objects can be deployed in different nodes, representing computing resources (i.e. a unit with processor, memory, network infrastructure, etc.) or a hardware unit upon which object execute their behavior. These concepts are similar to those available in UML and MARTE meta-models.

The behavior semantics of DERCS is to execute a sequence of actions in response to: (i) messages exchanged among objects, (ii) application events or (iii) state-related events. Objects, which can be deployed in the same (i.e. local objects) or different nodes (i.e. remote objects), interact by exchanging messages. After receiving a message, an object executes the behavior associated with this message. In DERCS, this behavior semantics is always the same, independently if the message has been sent from a remote or local object. The description on "how" to implement this communication is defined in the mapping rules according the target platform. GenERTiCA selects (based on DERCS information) the appropriate script for each sending message action. For events, the behavior semantics is similar, i.e. the behavior associated with an event is triggered after the recognition of this event's occurrence.

Furthermore, DERCS can represent not only send message action, but also other actions, such as assignments, evaluation of expression, state transitions, creation and destruction of objects. These actions represent an adaptation of UML 2.x meta-model's behavioral elements (e.g. actions).

The most important difference between DERCS and UML 2.x meta-model is the representation of aspects and their adaptations. Fig. 1 shows a fragment of DERCS meta-model that is related to AO concepts. Aspects provide structural and behavioral adaptations, which modify system's elements that have been selected through joint points. Inside the UML model, joint points are specified as *Join Point Designation Diagrams* (JPDD) [9]. JPDDs are "compiled" during the UML-to-DERCS transformation to create instances of the *JoinPoint*, which contain a list of elements selected by the JPDD. The following elements can be selected: (i) classes, (ii) attributes, (iii) methods, (iv) behaviors, (v) actions, (vi) objects, and (vii) nodes. All of these elements are descendents of the *BaseElement* meta-model class.

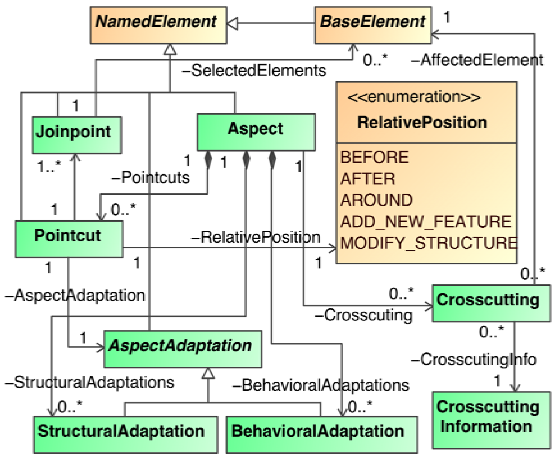

Fig. 1. DERCS elements related to non-functional requirements

Aspects' adaptations are linked with join points through *Pointcut* instances, which are obtained from *Aspects Crosscutting Overview Diagram* (ACOD) [4][10]. Adaptations are applied in the selected element at a *RelativePostion*, which can be before or after a certain point in the execution flow, or enclosing a behavior or action, or even modifying the structure or adding new features to an element. In this sense, DERAF aspects provide pre-defined adaptation semantics that shall be used to specify the non-functional requirements handling. *StructuralAdaptations* and *BehavioralAdaptations* are created according to aspects' adaptations that have been specified in ACOD.

Finally, it is important to highlight that DERCS tries to simplify the meta-model of UML 2.x by grouping structural and behavioral information, instead of having this information spread over different meta-model elements of different diagrams. More specifically, the modeled dynamic behavior (i.e. action sequences) described in different diagrams (i.e. activity, interaction and state diagrams) are grouped together into an instance of *Behavior* meta-model element. Moreover, AO-related meta-model elements separate the handling of function from non-functional requirements already in modeling level. Hence, this separation can be carried until the aspects weaving phase. This is an important contribution for the improvement of model elements reusability, as presented in [10]. Additionally, this separation is also helpful to mapping rules description since it allows designers to concentrate efforts in one concern at each time.

3 UML-to-DERCS Transformation

Once the distributed embedded real-time system is modeled using UML, MARTE and DERAF, it is possible to transform the UML model into a DERCS model. GenERTiCA performs this transformation automatically using pre-defined heuristics.

GenERTiCA has been implemented as a plug-in to the Magic Draw CASE tool [15], which provides an API to access the UML 2.x meta-model. Hence, the transformation from UML to DERCS (as well as DERCS meta-model itself) was written as "normal Java code" instead of using transformations frameworks, such as Acceleo [13] and OpenArchitectureWare [14]. The main reason for this choice was the possibility to access full information from the UML model and its diagrams using an MOF [17] API (provided by the modeling tool). Using the mentioned tools, the UML model must be exported as XMI to be imported as other representation, such as EMF [16], which could introduce XMI version incompatibility problems. However it is worth to mention that GenERTiCA implementation is not constrained to be a plug-in for a specific case tool. Its source code is modular enough to allow the replacement of the current transformation engine implementation to other one using different transformation frameworks as the ones mentioned.

The transformation of structural specification from the UML meta-model to DERCS meta-model is straightforward, i.e. is a one-to-one mapping, as depicted in Table 1. However a remark concerning relationships of classes must be written. For one to many relationships, an array attribute is created in the "one relationship-end" to represent the "many relationship-end". Further, for composition associations, methods to add or remove elements to/from this array are also created in the DERCS model.

Considering the behavior specified in the UML model, GenERTiCA analyzes sequence diagrams to find actions that are executed within the context of methods. It also identifies branches in the execution flow as well as loops. This is achieved using a call stack-based mechanism similar to one implemented in computer programs. Reserved words are used to indicate actions that are different from message sending, allowing the specification of assignments or expression evaluations actions within sequence diagrams. During the evaluation of sequence diagrams, objects are also "discovered". On the other hand, their deployment is performed according to information obtained from deployment diagram.

Table 1. Mappings between structural elements of UML and DERCS meta-model

UML meta-model	**DERCS meta-model**
Class	*Class*
Property	*Attribute*
Type ou *PrimitiveType*	*DataType* sub-class
VisibilityKind	*Visibility*
Operation	*Method*
decorated with <<getter>>	*Method, Behavior, ReturnAction*
decorated with <<setter>>	*Method, Parameter, Behavior, AssignmentAction*
Parameter	*Parameter*
ParameterDirectionKind	*ParameterKind*
Association	*Attribute, Method, Behavior, ReturnAction, AssignmentAction, Parameter*
if any association end defines *AggregationKind* as *composite*	Além dos acima, *CreateObjectAction, DestroyObjectAction*
Lifeline ou *InstanceSpecification*	
related to class decorated with <<SchedulableResource>>	*ActiveObject*
classes without stereotype or decorated with <<MutualExclusionResource>> or <<Resource>>	*PassiveObject*

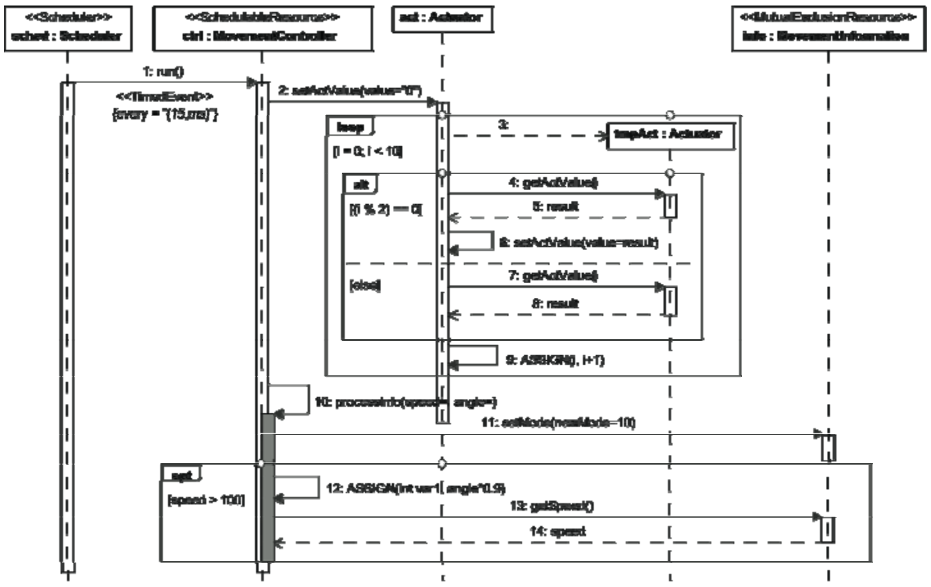

Fig. 2. Specification of system behavior using a sequence diagram

Fig. 2 depicts a sequence diagram that illustrates an example of system behavior modeling. From this diagram, transformation heuristics can infer the behavior of three methods: (i) *MovementController.run()*; (ii) *Actuator.setActValue()*; (iii) *Movement Information.processInfo()*. For (i), the behavior has two send message actions (related to messages 2 and 10). The behavior of (ii) is more complex having a object creation action (message 3), a branch, three send message actions (messages 4-8) and an assignment action (message 9). Considering (iii), its behavior has two send message actions (messages 11 and 13), an assignment action (message 12) and a branch.

An important part of UML-to-DERCS transformation heuristics is the creation of AO related elements in DERCS model. Aspects are specified in ACOD, which is a special kind of class diagram. Each class annotated with <<Aspect>> stereotype is transformed into an aspect. Structural and behavioral adaptations, and also pointcuts are represented as instances of the *Operation* meta-class in the UML meta-model. They are transformed into instances of, respectively, *StructuralAdaptation*, *BehavioralAdaptation* and *Pointcut* meta-classes of DERCS.

However the most critical part of the transformation is the evaluation of JPDDs to create *JoinPoint* instances, and also to select elements according the search criteria specified by JPDD. In spite of the expressiveness provided by JPPD to select join points, in current version of the UML-to-DERCS transformation heuristics, JPDD specification is constrained to simple selections. Hence, DERCS elements that can be selected are: (i) *Class*; (ii) *Attribute*; (iii) *Method*; (iv) *Node*; (v) *SendMessageAction*; (vi) *ReturnAction*; (vii) *CreateObjectAction*; (viii) *DestroyObjectAction*; (ix) *Behavior*. Fig. 3 depicts a JPDD that selects the active objects' behaviors that are associated to methods that are executed cyclically. Therefore, when this JPPD is evaluated, all instances of DERCS' *Behavior* meta-model element, which match with the specified selection criteria, are gathered into the join point created to represent this JPDD.

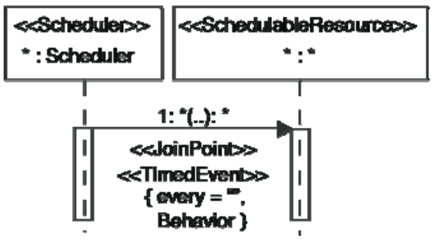

Fig. 3. JPDD: selection of active objects' behavior that is triggered periodically

4 Code Generation and Aspects Weaving

GenERTiCA is a tool created to support the AMoDE-RT [4], which separates concerns related to requirements handling by using aspects from DERAF [7] along with UML and MARTE profile [8]. At modeling phase the (informal) semantics of each aspect adaptation are platform independent, i.e. it is defined how the system is affected by adaptations but there are no definitions on how to implement such adaptations in a given platform. These adaptations implementations are done as mapping rules scripts that use services and constructions available in the target platform. A discussion on aspects implementation has been provided in [18] and [20].

GenERTiCA's code generation approach is based on mapping rules scripts that are described in XML files. Mapping rules for different platforms can be specified in a single XML file. GenERTiCA chooses the appropriate mapping rules according to objects deployment information available in DERCS model. Fig. 4 depicts organization of mapping rules in the XML file.

There are two kinds of mapping rules: *Application* and *Platform Configuration*. The former represents the mapping rules to generate the application source code, i.e. code that implements system's expected behavior according to application requirements; code describing how objects are connected and how they interact, as well as how the application handles system's non-functional requirements. On the other hand, the *Platform Configuration* branch contains scripts that are responsible to customize services provided by the platform, onto which the application runs. Such customization can be seen as an "on/off switch" for platform services. Depending on how the platform can be customized, GenERTiCA can create configuration files, and/or to tailor APIs' source code, using information from DERCS model, e.g. list of aspects specified in the ACOD.

The XML format was chosen due to its organization, as a tree structure, that facilitates the reuse of previously developed scripts, as well as the organization of the produced scripts. Leafs contain scripts, which are in fact the ones responsible to perform the translation from DERCS elements to source code, and also to weave aspects adaptations into the generated code or input model (i.e. DERCS model). By using XML, the designer can create repositories to store implementations of DERAF aspects, as well as mapping rules that have been previously created and validated. It is an easy task to describe a script to map, for example, actions that represent message exchanged between remote objects using a given API, and reuse it in a futher project that uses the same API and/or target platform.

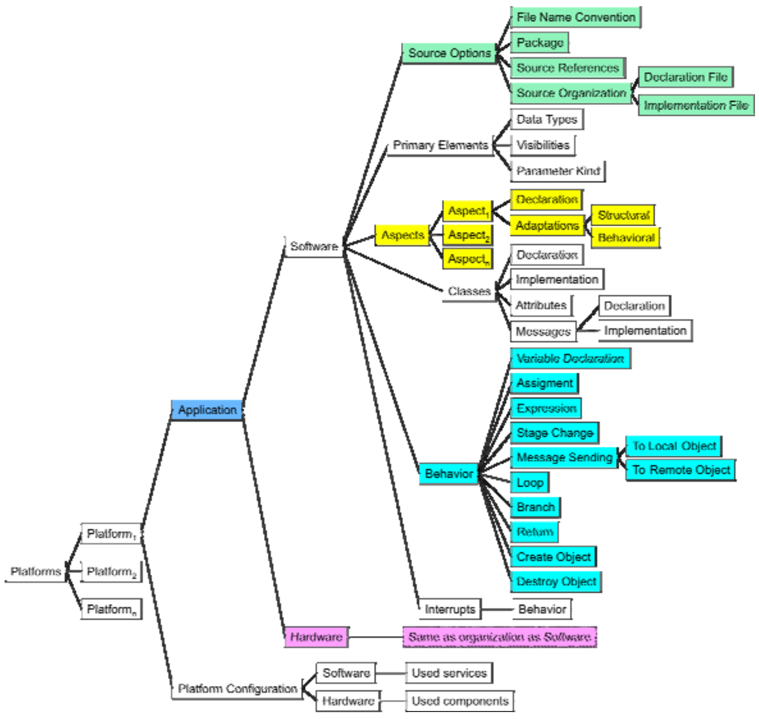

Fig. 4. Strcuture of mapping rules XML files

Simplicity was one of the aims for the description on how to map DERCS elements into source code. GenERTiCA adopts an approach that uses small scripts that are responsible to generate fragments of source code (for each single element of DERCS), which are combined into source code files. On other words, scripts in XML tree's leafs need only to access information from few elements (or just a single one), keeping the focus and aim of the mapping rule, instead of creating complex scripts that can generate the whole source code. The scripting language used in GenERTiCA is the *Velocity Template Language* (VTL) [11], an open source and wide used scripting language. VTL was chose because it allows writing any kind of text, into which one can specify simple commands to customize the text produced after script execution. Thus, VTL accomplishes GenERTiCA requirement for generating code for different languages. For examples of VTL scripts, interested readers are referred to [18] and [20].

To perform the code generation/aspects weaving, GenERTiCA implements the algorithm depicted as an activity diagram in Fig. 5. GenERTiCA traverses the list of DERCS' elements (e.g. classes, attributes, methods, behaviors, etc.), looking for a script that matches with the selected element, according to the tree hierarchy of the XML file. Once the script is found, it is executed, generating a fragment of text representing the source code for the selected element. In addition, GenERTiCA verifies if any aspect affects the selected element, i.e. it checks if this element is contained in the selection list of any join point. If this is the case, all pointcuts, which are related to that join point, are evaluated to find which adaptations must be applied in the selected element.

Adaptations' scripts are executed and modify the generated code fragment, i.e. the aspect weaving is performed. This algorithm repeats until all elements are evaluated.

It is important to highlight that, due to VTL, leaf nodes contain scripts that allow the generation of source code fragments for different languages such as C/C++, Java, VHDL, SystemC, and others. These generated code fragments are merged to create source code files, which are compiled or synthesized by external tools. The script language has full access to information about system objects using DERCS metamodel (see section 2). Therefore, it is possible to construct specialized and also complex mapping rules.

VTL allows scripts to access methods available by any object registered in the scripting engine. Therefore, aspect adaptation scripts can perform modifications in the DERCS model (i.e. aspects model weaving) by means of accessing methods available in DERCS API. Aspects model weaving allows an early evaluation of the impact caused by aspects adaptations in the model. Thus, tools such as design space exploration tools could evaluate different implementations of the same aspect in order to chose one that best fits with system requirements. For details on design space exploration at modeling level using UML, interested readers should refer [12].

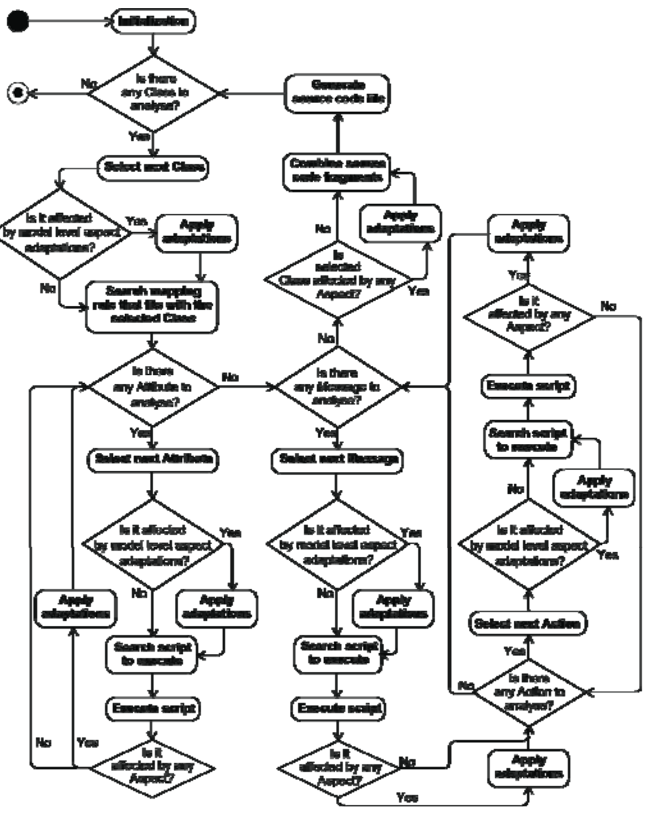
Initialization
Is there any Class to analyse?
No
Yes
Generate source code file
Select next Class
Combine source code fragments
Is it affected by model level aspect adaptations?
Yes
No
Apply adaptations
Search mapping rule that fits with the selected Class
Is selected Class affected by any Aspect?
Is there any Attribute to analyse?
Is there any Message to analyse?
Select next Attribute
Select next Message
Search script to execute
Execute script
Is it affected by any Aspect?
Is there any Action to analyse?
Select next Action

Fig. 5. Code-generation and aspects weaving algorithm

5 Related Work

Code generation is not a new topic. The idea of using computers to generate code from some higher abstract specification comes since the beginning of the use of computers. Several works aiming at distinct domains have been done in this subject. In general, it can be stated that techniques vary from the sort of input specification, passing through the way to perform code generation, until the amount of generated code and the target language. In order to keep the focus on the generation of code for distributed embedded real-time systems, this section discusses a small fraction of recent related works.

CoSMIC [5] provides generative tools that work with a set of Domain Specific Modeling Languages (DSML) in order to provide a component-based implementation for a distributed embedded real-time system. DSML are interesting options for distributed embedded real-time systems design, however user-made languages introduce some problems due to the lack of standardization for concepts and language constructions. In our approach we use a standard and well-accepted modeling language like UML that overcomes such problems. Besides, the CoSMIC's AO-related tools work with aspect like AO languages does, i.e. aspects, joinpoints, pointcuts are specified as text instead of using a graphical notation as our approach does. However, besides not using graphic representations for AO concepts, these tools can also modify the input model.

SysWeaver [6] is a toolset for analyzing non-functional requirements, automating design choices (design space exploration), and also to generate code for distributed embedded real-time systems, which are specified using Mathlab/Simulink. Basically, the difference from that work is that our work deals with distributed embedded real-time systems specification in terms of OO and AO concepts, instead of specifying functional and para-functional properties. In addition, our code generation approach seems to be more flexible to generate code for different target languages due to ability of VTL's scripting engine, which recognizes commands within any normal text, and also due to the approach of using small script dedicated to a specific element of the input model.

6 Conclusions and Future Work

The proposed work addresses the problem of automatic code generation for distributed embedded real-time systems. The presented infrastructure deals with functional and non-functional requirements using concepts of AO paradigm in order to improve the separation of concerns on their handling.

GenERTiCA is a code generation tool capable to deal with aspects during the code generation process. The adopted approach uses small scripts to produce code fragments that are merged to create source code files. Besides code generation, GenERTiCA can also perform aspects weaving at model and implementation levels. Hence, it is possible to perform aspects weaving in the input model, as well as in the generated source code. GenERTiCA's approach supports the use of AO concepts together with non-AO languages, e.g. C/C++, SystemsC, VHDL, since AO concepts are specified in the model, and aspects are implemented as mapping rules that uses constructions available in the target laguage.

GenERTiCA takes a UML model as input, transforming it into a DERCS model, which has a simplified meta-model (compared with the UML one) to provide information on system structure, behavior, and non-functional requirements handling (as DERAF aspects). Additionally, a set of mapping rules organized in a XML file is also used as input to GenERTiCA. With this approach, scripts previously developed can be easily reused in further projects [20], reducing the design time. Once a set of mapping rules is created and validated to a given platform, it can be easily reused into other projects that use the same target platform. As presented in [20], GenERTiCA has been successfully used in some case studies to generate RTSJ code from UML models, as well as C++ code for the ORCOS [21].

As future work, other case studies are being developed, as well as mapping rules for other target platforms (e.g. EPOS [19], RTAI, and VHDL) are being created. The implementation of transformation heuristics needs to be improved to support other JPDD selections. Other possible future work is to modify GenERTiCA implementation to use one of the open source transformation engines mentioned in this text. Additionally, the presented methodology is intended to guide the development of adaptable and flexible middleware aiming at its use in wireless sensor networks. The use of aspect-oriented concepts will allow the aggregation of different features to the middleware as they are required, and this way, promoting the desired flexibility and adaptability.

References

1. Selic, B.: The Pragmatics of Model-Driven Development. IEEE Software 20(5), 19–25 (2003)
2. Ossler, H., Tarr, P.: Using Subject-Oriented Programming to Overcome Common Problems in Object-Oriented Software Development/Evolution. In: 21st International Conference of Software Engineering, pp. 687–688. IEEE Computer Society Press, Los Alamitos (1999)
3. Kiczales, G., et al.: Aspect-Oriented Programming. In: Aksit, M., Matsuoka, S. (eds.) ECOOP 1997. LNCS, vol. 1241, pp. 220–242. Springer, Heidelberg (1997)
4. Wehrmeister, M.A., et al.: An Aspect-Oriented Approach for Dealing with Non-Functional Requirements in a Model-Driven Development of Distributed Embedded Real-Time Systems. In: 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp. 428–432. IEEE Computer Society, Los Alamitos (2007)
5. Gokhale, A., et al.: Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications. Journal of Science of Computer Programming: Model-Driven Architecture (2004)
6. Niz, D., et al.: Model-based Development of Embedded Systems: The SysWeaver Approach. In: 12th IEEE Real-time and Embedded Technology and Applications Symposium, pp. 231–242. IEEE Computer Society, Los Alamitos (2006)
7. Freitas, E.P., et al.: DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. In: Moreira, A., Grundy, J. (eds.) Early Aspects Workshop 2007 and EACSL 2007. LNCS, vol. 4765, pp. 55–74. Springer, Heidelberg (2007)

8. OMG. UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE) (2005), http://www.omg.org/cgi-bin/doc?ptc/2007-08-04
9. Stein, D., et al.: Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In: 5th International Conference on Aspect-Oriented Software Development, pp. 15–26. ACM Press, New York (2006)
10. Wehrmeister, M.A., et al.: A Case Study to Evaluate Pros/Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems. In: 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, pp. 44–54. IEEE Computer Society, Los Alamitos (2008)
11. The Apache Velocity Project, http://velocity.apache.org/engine/releases/velocity-1.5/
12. Oliveira, M.F.S., et al.: Early Embedded Software Design Space Exploration Using UML-Based Estimation. In: 7th IEEE International Workshop on Rapid System Prototyping, pp. 24–32. IEEE Computer Society, Los Alamitos (2006)
13. Acceleo, http://www.acceleo.org
14. openArchitectureWare, http://www.openarchitectureware.org/
15. Magic Draw tool, http://www.magicdraw.com/
16. Eclipse Modeling Framework, http://www.eclipse.org/modeling/emf/
17. OMG, Meta Object Facility (MOF), http://www.omg.org/mof/
18. Wehrmeister, M.A., et al.: GenERTiCA: A Tool for Code Generation and Aspects Weaving. In: 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, pp. 234–238. IEEE Computer Society, Los Alamitos (2008)
19. Fröhlich, A.A., Wanner, L.F.: Operating System Support for Wireless Sensor Networks. Journal of Computer Science 4(4), 272–281 (2008)
20. Wehrmeister, M.A.: An Aspect-Oriented Model Driven Engineering Approach for Distributed Embedded Real-Time Systems, Ph.D. Thesis, Federal University of Rio Grande do Sul, Brazil (2009)
21. Organic Reconfigurable Operating System, https://orcos.cs.uni-paderborn.de/orcos/