Des Autom Embed Syst (2016) 20:127–153
DOI 10.1007/s10617-016-9172-6

System-level design based on UML/MARTE for FPGA-based embedded real-time systems

Marcela Leite[1] · **Marco Aurélio Wehrmeister**[2]

Received: 15 March 2015 / Accepted: 25 February 2016 / Published online: 9 March 2016

Abstract This paper discusses an approach to generate VHDL descriptions from high-level specifications, namely UML/MARTE models that include aspect-oriented semantics. Standard UML diagrams describe the handling of functional requirements, whereas crosscutting concerns associated with the non-functional requirements are handled by aspects. UML-to-VHDL transformation is performed automatically by a script-based code generation tool named GenERTiCA. For that, mapping rules scripts define how to generate VHDL constructs from model elements, including the implementation of aspects adaptations. The generated VHDL description does not require any manual modification, in order to be fully synthesized onto a FPGA device. Some case studies have been performed to evaluate the proposed approach, including examples of real systems implemented as a FPGA-based embedded system. Obtained results show an improvement in system design in terms of an increase in system performance as well as a better utilization of FPGA reconfigurable resources. Such positive results are related to a better modularization of components achieved by using the proposed high-level approach. These case studies demonstrate the practicability of full translation of platform-independent specifications into VHDL descriptions.

Keywords Model-driven engineering (MDE) · UML · VHDL · Code-generation · Aspect-oriented design · Non-functional requirements

1 Introduction

Nowadays, an increasing number of embedded systems are being delivered to the final customers with *Field-Programmable Gate Array* (FPGA) components interconnected with

✉ Marco Aurélio Wehrmeister
wehrmeister@utfpr.edu.br

Marcela Leite
marcela.leite@ifc-araquari.edu.br

1 Instituto Federal Catarinense (IFC Araquari), Araquari, Brazil

2 Federal University of Technology - Paraná (UTFPR), Curitiba, Brazil

traditional IC components (e.g., processors and memory) as the system hardware platform. Programmability, flexibility and better performance are some of the reasons to justify the use of a FPGA device to perform some tasks of embedded systems [19,25].

In such modern FPGA-based embedded systems, the hardware/software co-design and partitioning of system tasks, as well as the increasing number of services demanded from them, lead to an increase in design complexity. New approaches are required in order to cope with such a complexity. An old but still valid idea is the increase of the abstraction level used during design [11,18]. Both academia and industry are looking for new approaches that use high-level specifications such as, for instance, *Unified Modeling Language* (UML) models annotated with stereotypes of the profile for *Modeling and Analysis of Real-Time and Embedded Systems* (MARTE). *Model-Driven Engineering* (MDE) [12,18,32] makes intensive use of models, in order to manage complexity and also other design constraints. Such an approach advocates that engineers need lesser focus on technological details, and thus, system design (as a whole) may profit from using the information contained in the high-level models to generate the executable artifacts of embedded and real-time systems.

Since late 1990's, one can find approaches, e.g., [16], that are able to generate VHDL descriptions from UML models. In [6,27], an UML model (especially its class diagram) is used as source of information to generate a VHDL description that comprises only the system structure, i.e., its components and their connections, not its behavior. System behavior is commonly generated from state diagrams as in [34]. The main drawback of such approaches is that it is common to find VHDL statements within the UML model, decreasing its abstraction level, as well as its usability for generating code for other languages. Furthermore, state diagrams are not frequently used by software engineers, and hence, co-design of embedded system components may be hindered, increasing design time. Furthermore, most of the proposed approaches, e.g., [3,4,28], do not use the main principles of Object-Oriented (OO) paradigm, e.g., inheritance and composition/aggregation relationships in class diagram to define system structure. In addition, such approaches do not use other behavioral diagrams from UML, for instance, sequence diagrams to define system behavior. The mentioned high-level features (e.g. inheritance and multiple diagrams to specify system behavior) are some of the UML strengths [26] and should not be ignored within UML-based approaches.

Aspect-oriented Model-Driven Engineering for Real-Time systems (AMoDE-RT) [32] is a MDE approach that proposes the intensive use of UML/MARTE models, from which a system implementation can be automatically generated, including software/hardware implementation of embedded system components. In [20], AMoDE-RT has been extended to support the generation of VHDL descriptions, including both system structure and behavior. System behavior is specified in a platform-independent fashion via sequence diagrams rather than state diagrams, since the former is easily understood by both hardware and software engineering teams. However, the approach proposed in [20] has some limitations: (i) it supports only few classes/objects per system; (ii) only 1-to-1 associations are supported; (iii) the generated behavior does not differentiate synchronous and asynchronous method[1] calls used within sequence diagrams; (iv) engineers need to include unnecessary details in the UML model in order to generate the VHDL description as complete as possible; (v) depending on the amount of such details, generated VHDL descriptions need some minor manual modifications before they can be synthesized by a synthesis tool. This work improves [20] and proposes a new set of mapping rules for the UML-to-VHDL transformation, including the

[1] In this text, a "method" encapsulates a behavior of the object that is executed in response to a message sent from other object.

support of some OO constructions which have not been initially supported, as well as the support for sequence diagrams full semantics.

Another issue that affects the design of embedded and real-time systems is the effective handling of non-functional requirements, e.g., deadlines, energy consumption, reduced footprint, communication latency, etc. In the embedded systems domain, non-functional requirements introduce crosscutting concerns in system design and implementation. Such concerns are not properly handled by traditional approaches (e.g., object-orientated, component-based approaches) due to the functional decomposition [7,13] enforced by such approaches. This decomposition schema leads to modularization problems of crosscutting concerns, since their handling cannot be encapsulated within single modular units, which, in turn, results in scattered and tangled handling.

Aspect-Oriented Programming [13] addresses crosscutting concerns modularization issues in software components by proposing special constructs called *aspects*. Aspects encapsulate the handling of crosscutting concerns by defining *adaptations* (also known as *advices* in AOP terminology). Engineers indicate which elements of the system base code are affected by aspects adaptations. For that, they define *pointcuts* describing elements selection expressions to indicate *join points* on which adaptations are applied. In this sense, VHDL descriptions are somehow similar to a software source code files: they describe components structure and behavior using a language based on functional decomposition. Therefore, the handling of crosscutting concerns is found in many distinct elements, and hence, VHDL descriptions are subject to the same modularization issues related to crosscutting non-functional requirements [8,17,22].

AMoDE-RT supports aspect-oriented concepts in UML/MARTE model by means of *DERAF aspects* [10,33]. Therefore, other contribution of this work is the VHDL implementation of some DERAF aspects. Aspects adaptations have been specified as mapping rules scripts that generate VHDL constructs to handle crosscutting concerns. By using the proposed approach, it is possible to generate a fully functional and synthesizable VHDL description from a more abstract UML model—there is no need for manual modifications on the generated descriptions. Engineers can generate both hardware and software source code from the same UML model [32], facilitating hardware/software co-design. This paper extends [31] by providing a detailed discussion on the UML-to-VHDL mapping rules for the high-level object- and aspect-oriented features as proposed in AMoDE-RT.

This work has been validated through some case studies that represent real systems, namely, the design of a digital watch, a line-following robot, and a valve control system. These systems have been modeled following AMoDE-RT approach and implemented in a FPGA development board. Results demonstrate the feasibility of the proposed approach. A better utilization of FPGA resources has been obtained, which, on the other hand, impacted on system performance. In comparison with the results presented in [20], a greater amount of source code lines was generated from the UML model. This may indicate a decrease in the effort for creating the embedded system implementation, especially when designing larger systems. Moreover, by using AMoDE-RT approach, it is possible not only to deal with non-functional requirements earlier in the design cycle, but also to obtain automatically a fully synthesizable VHDL description[2], as demonstrated in the presented case studies.

The rest of this text is organized as follows: Sect. 2 discusses related works; Sect. 3 describes how AMoDE-RT generates VHDL descriptions from aspect-oriented

[2] The generated VHDL description was synthesized, uploaded and executed on a FPGA development kit without any manual modification.

UML/MARTE models; Sect. 4 presents the case studies and analyses the obtained results. Finally, Sect. 5 draws some conclusions and discusses directions for future works.

2 Related works

Generating VHDL descriptions from high-level models has been proposed in various works [5,6,20,24,28]. For instance, *COmponent LAnguage* (COLA) tool allows the system high-level specification as well as the generation of a VHDL description [28]. The generated VHDL description comprises the hardware structure and behavior, which is extracted from state machine diagrams. However, COLA has its own formal modeling semantics, which is not a standard, hindering its adoption in comparison with UML-based approaches.

On the other hand, GASPARD is a tool that generates VHDL descriptions from UML/-MARTE models [6,24]. GASPARD uses concepts from logical view of MARTE *Hardware Resource Modeling* (HRM) package. Such a tool focuses on modularization and components mapping. It is necessary to define templates for the static elements of UML, in order to generate the system VHDL description. As a consequence, GASPARD generates entity and components mapping but it does not support the code generation for system behavior.

In [5], the proposed approach generates VHDL descriptions intended for system behavior validation. Sequence diagrams are used to specify system interactions, from which a VHDL description is generated. Design constraints are specified in sequence diagram as MARTE stereotypes, defining system non-functional properties that are subject to validation. However, that approach generates VHDL files only for system validation; it does not generate VHDL description for the system functional requirements.

Recently, an increasing number of research works propose the use of Aspect-Oriented Programming (AOP) in hardware design. The use of AOP in conjunct with VHDL language is analyzed in [8]. This work identifies some elements in VHDL which may be subject to aspects adaptations. As result, those authors indicate *process* and *variable/signal assignments* as possible *join points* in VHDL. These places are suitable for code injection, especially *before*, *after* and *around* each join point [8].

Aspect Described Hardware-description-language (ADH) is described in [23]. ADH is an aspect-oriented language for hardware descriptions based on AspectJ language [13]. A compiler has been developed to translate ADH descriptions into VHDL descriptions. However, according to [23], the compiler generates "synthesizable" VHDL code but it cannot be directly used on FPGA devices due to missing library classes, as well as other open issues and bugs of the compiler.

AspectVHDL language is an extension to VHDL syntax that includes AOP concepts and constructs [17]. Such a language allows the definition of *join points* for procedures, functions, data type definitions, entity architecture, and processes sensitivity list. However, in order to allow the definition of *join points* and hence to use AspectVHDL, the system description must be structured in terms of functions and procedures.

In order to design embedded systems using the synthesizable subset of SystemC, Muck et al. [22] propose an approach that combined AOP and OO programming. That work focuses on increasing components reuse by means of decreasing components coupling. For that, their approach uses meta-programming, OO concepts such as inheritance and interfaces, as well as design patterns. According to [22], the obtained results indicate that, although using higher abstraction levels for system design, system performance is not impacted negatively in spite of a small increase in FPGA area usage.

In comparison with the mentioned related work, this work presents the following differences: (i) AMoDE-RT is based on MDE techniques and high-level UML/MARTE models allowing the engineers specify the system functional and non-functional requirements in a platform independent way using a standard modeling language; (ii) AMoDE-RT is supported by a flexible code generation tool such as GenERTiCA, opening room for exploring the use of distinct target platforms for implementing the embedded real-time system, since engineers can easily change the mapping rules script. Moreover, this work enhances the VHDL generation approach proposed in [20] by means of providing a proper support for high-level constructs available in UML. System behavior is completely specified using sequence diagrams, including MARTE stereotypes for specifying of non-functional requirements. These diagrams are used to generate the VHDL components behavior. It is worth mentioning that an important contribution of this work is to address the handling of crosscutting non-functional requirements in a high-level and independent of platform way throughout the use of a platform-independent aspects framework. This work proposes and demonstrates one possible VHDL implementation of these high-level aspects. Thus, it is possible to generate a complete VHDL description from UML/MARTE high-level specification. Such a description is synthesizable and fully functional, according to system requirements specified in the UML/MARTE model.

3 From aspect-oriented UML models to VHDL descriptions

3.1 Overview of AMoDE-RT

Aspect-oriented Model-Driven Engineering for Real-Time systems (AMoDE-RT) [32,33] is an UML-based MDE approach that promotes the use of *Platform Independent Models* (PIM) as the main artifacts to specify embedded real-time system requirements, including system structure, behavior, constraints and non-functional properties. One of its main contributions is the separation of concerns for handling functional and non-functional requirements. For that, AMoDE-RT supports concepts of the *Aspect-Oriented Software Development* (AOSD) within UML/MARTE models by means of the *Distributed Embedded Real-time Aspect Framework* (DERAF) [10,33].

Furthermore, in AMoDE-RT, system implementation is obtained automatically from UML/MARTE models by means of model transformations [32]. For that, AMoDE-RT proposes the use of an intermediate platform independent model called *Distributed Embedded Real-Time Compact Specification* (DERCS) [30,33]. DERCS model of an embedded real-time system represents functional requirements in terms of concepts of object-oriented paradigm, whereas non-functional requirements are represented using concepts of aspect-oriented paradigm. *Generation of Embedded Real-Time Code based on Aspects* (GenERTiCA) tool [29] supports the UML-to-DERCS transformation as well as the automatic generation of source code from the DERCS model. GenERTiCA implements a script-based code generation approach that uses a set of scripts to map model elements into constructs, services, and/or APIs provided by a given target platform. Engineers may specify mapping rules scripts for distinct target platforms, including both software and hardware (i.e., using *Hardware Description Languages*). Therefore, this work proposes a new set of mapping rules for VHDL implemented as the code generation scripts, enhancing our previous approach [20]. Figure 1 shows the code generation approach implemented in GenERTiCA tool [29].

In addition, it is important to highlight that GenERTiCA does not only generate source code but also performs aspects weaving. In other words, GenERTiCA weaves aspects adap-

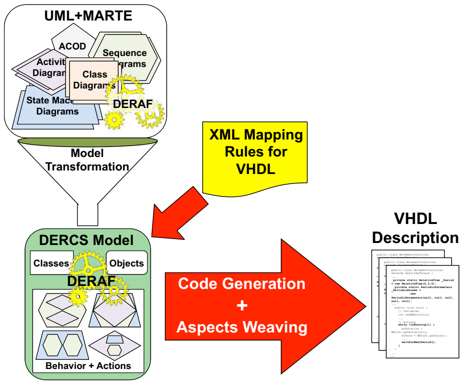

Fig. 1 GenERTiCA code generation process

tations into the generated code, allowing the use of aspect-oriented concepts, even though the target implementation language does not support such concepts. For that, GenERTiCA uses the information provided by DERAF aspects, which have been specified in a platform independent way within the UML/MARTE model. Aspects adaptations are implemented as mapping rules scripts, enabling their portability for different platforms or/and applications. An adaptation may occur in two ways: in model level or in code level. At model level, adaptations may include or change elements in DERCS model [29] generated from UML/MARTE specification. These modifications are accessible to mapping rules during code generation step. On the other hand, at code level, modifications affect directly the generated code without changing the input model. Thus, a relevant contribution of this work is the VHDL implementation of some DERAF aspects, via mapping rules scripts, that was not supported in [20]. Details on AMoDE-RT, DERAF and GenERTiCA can be found in [10,29,32,33].

3.2 Functional requirements

A set of mapping rules to generate VHDL code from UML/MARTE models has already been proposed in a previous work [20]. However, as mentioned in [15], our previous approach [20] has some limitations, especially when one considers the potential increase in the specification abstraction level by using object-oriented features, e.g., encapsulation and inheritance. Besides improving the management of design complexity, by using such abstractions, engineers may improve artifacts[3] modularity, and hence, facilitate their reuse.

In order to address the already mentioned issues, a new set of mapping rules have been created, as depicted in Table 1. First column shows the UML meta-model elements; second column depicts the VHDL elements mapping, as proposed in [20]; and, third column shows the modifications proposed in this work for UML-to-VHDL mapping. As one can note, classes are mapped into entities and their related architectures. In addition to the support for encapsulation and inheritance, an important contribution of these new mapping rules is the support for synchronous and asynchronous method calls.

Furthermore, it is worth mentioning that this work supports UML structural elements (e.g., classes and objects) and behavioral elements (e.g., interactions and actions). Some stereotypes from *Hardware Resource Modeling* (HRM) and *Software Resource Modeling* (SRM) of MARTE profile are also covered, e.g., Table 1 shows that «`TimedEvent`» stereotype indicates that the annotated message is mapped to a VHDL process that is activated period-

[3] E.g. model, source code, description of components, etc.

Table 1 Mapping concepts from UML/MARTE to VHDL

UML		P.Work [20]	New mapping
Class	Structure	Entity-arch. Pair	Entity-architecture pair
	Association	Entity ports (1-to-1)	Components (1-to-n)
	Generalization	–	Components for concrete classes; functions, attributes and/or methods from parent class
Attributes	Public	Entity ports	–
	Private	Signals	Ports, signals or constants
	Read-only	–	Constants
Data types	Enumeration	–	Enumerations within packages
	Composite	–	Vectors
Method/ operation	Behavior	Process	Process or *inline* code
	Message occurrence specification	Entity ports	Signals assignment or process activation
	«TimedEvent»	–	"Active" process

ically according to its timing constraint. It is worth mentioning that it is possible to extend the mapping rules to add different packages and elements. AMoDE-RT and GenERTiCA are flexible approaches, and hence, they can support other mapping rules describing different implementation strategies for the model elements or DERAF aspects.

In order to illustrate the translation process, the remainder of this section discusses some key UML-to-VHDL mappings representing the system functional requirements. For that the digital watch case study is presented. Figure 2 shows the class diagram of the digital watch. In summary, this system is composed of a timer (*WatchTimer* class), four seven-segment displays (*Display* and *Number* classes), and a controller component (*SystemControl* class). This digital watch shows time passing in terms of minutes and seconds. Each seven-segment display shows one digit of the given number of minutes or seconds and is controlled via an activation port (turn on/off each digit). In addition, *Segment* class represents the status (on/off) of each segment of the seven-segment display. The watch control system assigns the current number of minutes/seconds (generated by the timer) to the related digit display, which, in turn, enables the correct segments according to this number. Thereafter, the watch control system activates this digit turning off the other ones.[4] However, as this system has a high refresh rate for the seven-segment displays, turning on/off each display individually is imperceptible to human eyes.

As mentioned, this work provides the support for object-oriented features such as generalization/inheritance and association between classes. Generalization relationship is translated to VHDL in two ways. The first one considers concrete super classes. In this case, a component representing the super class is instantiated within all child classes, i.e., the super class is mapped to a separated entity which owns ports, signals and processes like a "regular" class. On the other hand, the second mapping is related to abstract super classes. In this case, all child classes incorporate the elements from the super class, i.e., all attributes and methods of the super class are included into the child class. Therefore, the UML-to-VHDL mapping

[4] Such a behavior is a limitation of the used FPGA development board.

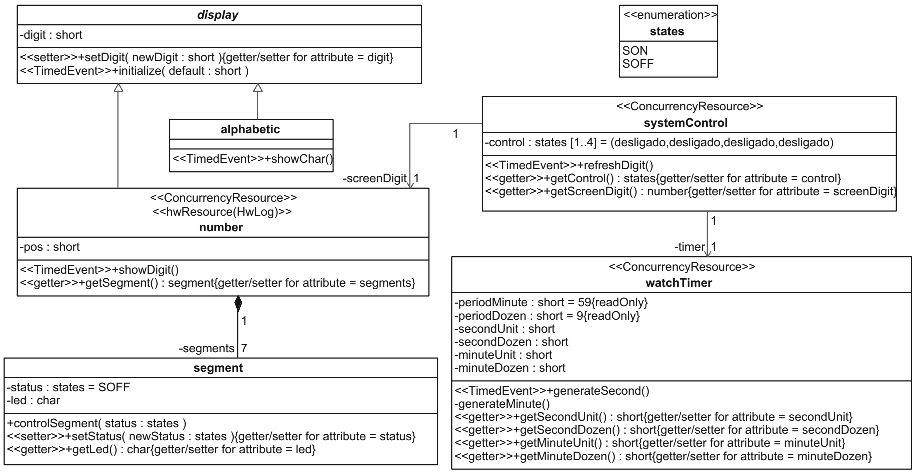
display
-digit : short
<<setter>>+setDigit(newDigit : short){getter/setter for attribute = digit}
<<TimedEvent>>+initialize(default : short)
<<enumeration>>
states
SON
SOFF
alphabetic
<<TimedEvent>>+showChar()
<<ConcurrencyResource>>
systemControl
-control : states [1..4] = (desligado,desligado,desligado,desligado)
<<TimedEvent>>+refreshDigit()
<<getter>>+getControl() : states{getter/setter for attribute = control}
<<getter>>+getScreenDigit() : number{getter/setter for attribute = screenDigit}
1
-screenDigit 1
<<ConcurrencyResource>>
<<hwResource(HwLog)>>
number
-pos : short
<<TimedEvent>>+showDigit()
<<getter>>+getSegment() : segment{getter/setter for attribute = segments}
1
-timer 1
<<ConcurrencyResource>>
watchTimer
-periodMinute : short = 59{readOnly}
-periodDozen : short = 9{readOnly}
-secondUnit : short
-secondDozen : short
-minuteUnit : short
-minuteDozen : short
<<TimedEvent>>+generateSecond()
-generateMinute()
<<getter>>+getSecondUnit() : short{getter/setter for attribute = secondUnit}
<<getter>>+getSecondDozen() : short{getter/setter for attribute = secondDozen}
<<getter>>+getMinuteUnit() : short{getter/setter for attribute = minuteUnit}
<<getter>>+getMinuteDozen() : short{getter/setter for attribute = minuteDozen}
1
-segments 7
segment
-status : states = SOFF
-led : char
+controlSegment(status : states)
<<setter>>+setStatus(newStatus : states){getter/setter for attribute = status}
<<getter>>+getLed() : char{getter/setter for attribute = led}

Fig. 2 Watch class diagram

generates an entity for each child class that incorporate ports, signals and processes from the super class. In a similar way, when a super class has abstract methods, these methods are implemented in all child classes and removed from the super class.

Listing 1 shows an excerpt of the mapping rule that checks whether a class has an abstract super class, i.e. it shows the second way to map inheritance relationships. Mapping rules run through all DERCS elements and can access and modify the information contained in the DERCS model. In the line 12, one can see that, after several information and consistency checks, the rules found an attribute in the abstract super class that, in turn, is added to the child class—the same happens with other features (e.g., methods) of the abstract super class. Listing 2 (lines 8, 30–36) shows the VHDL constructs generated from executing this script using the information of *Number* class. The *digit* attribute and *setDigit* and *initialize* methods are inherited from *Display* (see Fig. 2). Since *digit* attribute has a setter method (i.e., *setDigit*), the proposed UML-to-VHDL mapping determines that the inherited attribute *digit* is mapped to an INOUT port (see line 8). Details on attributes and methods mappings are discussed in [15].

Listing 1 Excerpt from Mapping rules for VHDL component generation from class structure

```
 1 ... // initial part of the script is omitted
 2 // checking whether there is a super class
 3 if ($Class.SuperClass && $Class.SuperClass.isAbstract)
 4   // select each method from super class
 5   foreach ($superMethod in $superClass.getMethods())
 6     // ignore methods which are class constructors
 7     if ($superMethod.Name != $superClass.Name)
 8       // check whether it is a getter/setter method
 9       if ($superMethod.isGetSetMethod())
10         ... // others validations are omitted
11       // thereafter add the new attribute to the child class
12       ${superAttribute.Name} : $Direction CodeGenerator.
             getDataTypeStr($superAttribute.getDataType())
13 ... // remainder of the script is omitted
```

Listing 2 Excerpt from *Number* entity description

```
 1 entity number is
 2  Port (
 3   clock : in STD_LOGIC;
 4   reset : in STD_LOGIC;
 5   segmentsled_0 : OUT BIT;
 6   segmentsled_1 : OUT BIT;
 7   ...
 8   digit : INOUT INTEGER RANGE -252 TO +252);
 9 end number;
10
11 architecture Behavioral of number is
12  signal segmentsstatus_0 : states:= SOFF;
13  signal segmentsstatus_1 : states:= SOFF;
14  ...
15  signal pos : INTEGER RANGE -252 TO +252:= 0;
16  ...
17 begin
18  segmentos_0: segmento port map(
19   clock => clock,
20   reset => reset,
21   status  =>  segmentosstatus_0,
22   led  =>  segmentosled_0
23  );
24  ... -- remainder of components instantiation is omitted
25  showDigit: process  (clock, reset)
26  begin
27   if (reset = '1') then
28    -- variables initialization
29   elsif (clock'EVENT and clock='1') then
30    if ( digit = 0 ) then
31     segmentsstatus_0 <= SON;
32     ... -- some lines are omitted
33     segmentsstatus_6 <= SOFF;
34    elsif ( digit = 1 ) then
35     segmentsstatus_0 <= SOFF;
36   ... -- remainder of behavior is omitted
37   end if;
38  end process showDigit;
39  ... -- remainder of behavior is omitted
40  initialize: process  (default)
41  begin
42   digit <= default;
43  end process initialize;
44 end Behavioral;
```

When the class *Display* is modified to represent a concrete class, the proposed UML-to-VHDL mapping generates an entity for this class as shown in the Listing 3. The *display* component is instantiated within the *number* entity in a way similar to a “regular” association between classes (see example in Listing 2, lines 18–23).

Listing 3 Excerpt from *Number* entity description

```
entity display is
 Port (
  clock : in STD_LOGIC;
  reset : in STD_LOGIC;
  digit : INOUT INTEGER RANGE -252 TO +252);
end display;
architecture Behavioral of display is
begin
 initialize: process  (default)
 begin
  digit <= default;
 end process initialize;
end Behavioral;
```

Other important feature supported by the proposed UML-to-VHDL mapping is the association relationship between classes. Table 1 shows that associations between classes are mapped to components instances and their related ports mapping. For instance, the composition relationship between *Number* and *Segment* classes (see Fig. 2) generates the instantiation of seven components, since this is a fixed 1-to-7 relationship. Listing 2 depicts this composition relationship mapping: lines 18–24 show the instantiation of the first of seven *segment* components within the *number* entity, as well as lines 5–7 show the ports associated to these components.

Regarding system behavior, this work provides the support for mapping the complete semantics of sequence diagrams elements into VHDL constructs. For that, the proposed UML-to-VHDL mapping supports the use of synchronous and asynchronous method calls[5], as well as MARTE «`TimedEvent`» stereotype to specify active object[6] concurrent behaviors.

Synchronous execution semantics is common in software components, while asynchronous execution semantics can be found in both software and hardware components. Synchronous execution semantics is not supported in VHDL synthesizable set, and hence, to support the use of synchronous method calls in sequence diagrams a heuristic has been proposed. The proposed UML-to-VHDL mapping defines that the whole behavior of a *called method* must be incorporated in the *caller method* behavior at the point in which the synchronous method call is specified. In other words, the VHDL fragment of the called method behavior is included into the VHDL fragment of the caller method. On the other hand, asynchronous execution semantics is supported in VHDL synthesizable set by means of processes. Therefore, besides creating a separated process for each method whose behavior is called asynchronously, the proposed UML-to-VHDL mapping defines a triggering control signal for this behavior. When an asynchronous method call is identified in the sequence diagram, a VHDL signal assignment is generated. Interested readers should refer to [15] for more details.

To illustrate the generation of VHDL code from a sequence diagram, Fig. 3 depicts the behavior of *refreshDigit* method of *SystemControl* class. VHDL description for this method is shown in Listing 4. As one can observe, the *alt* combined fragment is translated to an *if-then-else* statement (lines 8, 12, 16, etc.) and the messages within each compartment became

[5] In a *synchronous method call*, the execution of *caller method* stops when the method call occurs and continues only after the *called method* finishes its execution; in an *asynchronous method call*, the *caller method* continues executing its behavior after the method call, and the *called method* executes its behavior in parallel.

[6] An *active object* executes autonomously its behaviors in parallel with other active objects. A *passive object* only execute its behaviors in response to method calls from other objects.

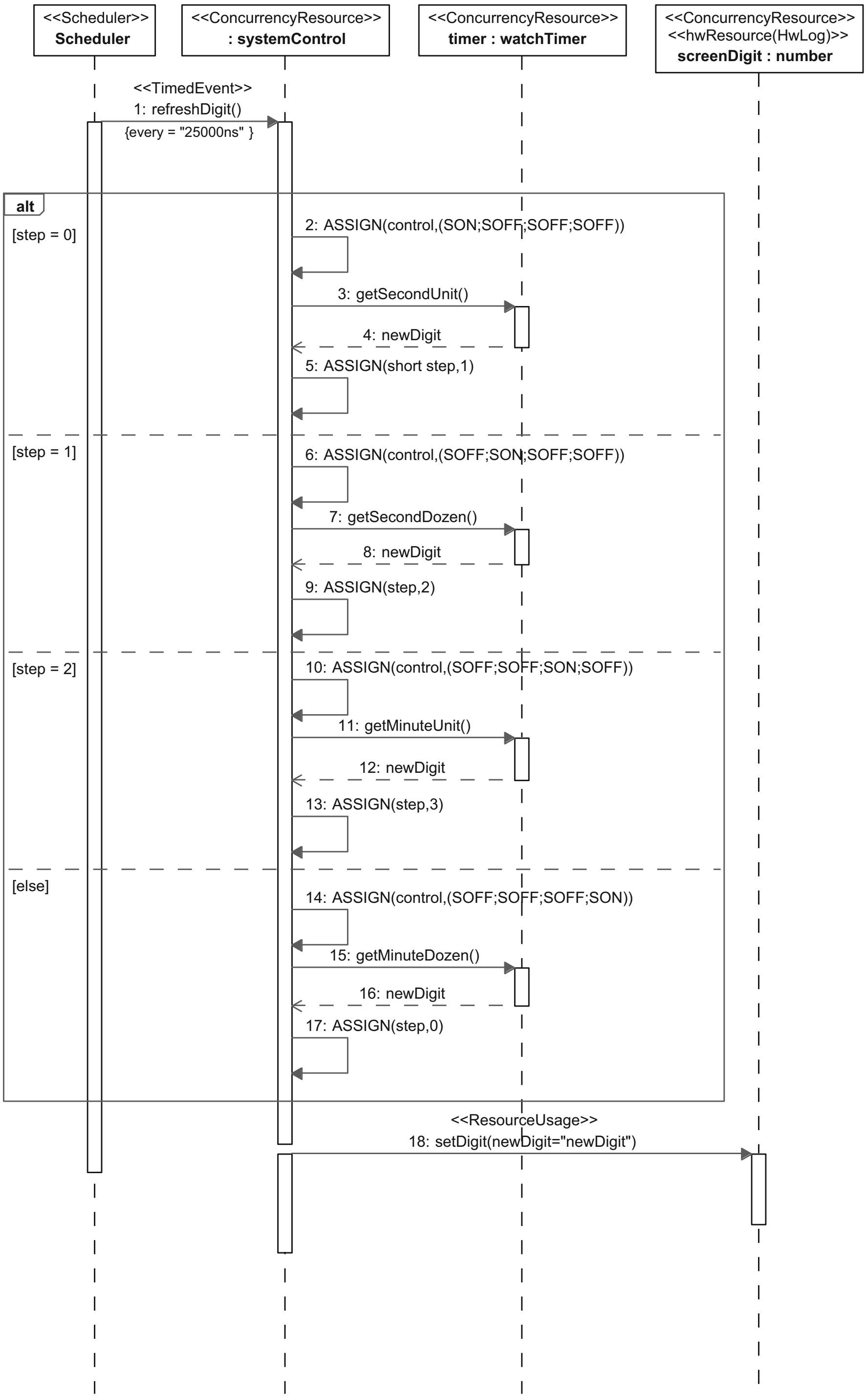

Fig. 3 Sequence diagram of *systemControl.refreshDigit*

action statements. According to AMoDE-RT modeling guidelines (see [33]), messages 2 and 5 represent assignment actions and are translated to the signal assignment statements shown in lines 9 and 11. Message 3, on the other hand, represents a synchronous method call of the *getSecondUnit* method, whose returned value is assigned to a variable. As discussed, the corresponding called method behavior is added at the point of the synchronous method call. In this case, as this message refers to a getter method, its behavior returns the current value of seconds counter. VHDL fragment that corresponds to this behavior is a variable assignment statement, which is included in the *refreshDigit* method behavior (i.e., the caller method) as shown in line 10. Similar VHDL fragments have been generated for the other messages contained within the *alt* combined fragment, as one can see in lines 12–15.

Listing 4 Excerpt from *systemControl* entity description

```
 1 refreshDigit: process  (refreshDigitClockdiv, reset)
 2   variable newDigit : INTEGER RANGE -252 TO +252 := 0;
 3   variable step : INTEGER RANGE -252 TO +252 := 0;
 4 begin
 5  if (reset = '1') then
 6   -- variables initialization
 7  elsif (refreshDigitClockdiv'EVENT and refreshDigitClockdiv='1') then
 8   if ( step = 0 ) then
 9    control <= (SON;SOFF;SOFF;SOFF);
10    newDigit := timersecondUnit ;
11    step := 1;
12   elsif ( step = 1 ) then
13    control <= (SOFF;SON;SOFF;SOFF);
14    newDigit := timersecondDozen ;
15    step := 2;
16   elsif ( step = 2 ) then
17    ... -- remainder of this behavior was omitted
18 end process refreshDigit;
```

It is important to highlight that sequence diagram elements provide all information needed to generate a VHDL process. As demonstrated in the Sect. 4, the proposed approach allows the generation of the complete and fully synthesizable VHDL description based on the information provided in the class diagram and various sequence diagrams. For that, 29 scripts of mapping rules have been created, totalizing 2365 code lines. By using these mapping rules scripts, engineers are allowed to use high-level object-oriented concepts supported in UML such as: encapsulation by means of get/set methods; 1-to-n relationships such as aggregations and compositions; inheritance/generalization relationships; full semantics of sequence diagrams, such as combined fragments, synchronous and asynchronous method calls. For additional details see [15].

3.3 Non-functional requirements

AMoDE-RT uses DERAF aspects to handle crosscutting concerns in a platform independent way within UML/MARTE models. The high-level semantics of DERAF allow its aspects to be implemented in distinct target platforms, including those that do not support AOSD concepts [32]. This section provides only an overview of the VHDL implementation of three DERAF aspects used in the case study. Details on how aspects adaptations are implemented in VHDL have been discussed in [14].

3.3.1 PeriodicTiming

PeriodicTiming aspect [33] deals with the periodic execution of one or more *active objects* behaviors. A similar behavior is achieved triggering periodically a *process* within VHDL entities. Therefore, the proposed VHDL implementation of *PeriodicTiming* considers VHDL entities as active objects, whereas each periodic behavior is mapped to a process. The execution frequency of periodic processes is controlled via a clock divider component associated to the entity. Therefore, in summary, *PeriodicTiming* aspect modifies the generated code by including a clock divider for each affected entity, as well as creates the interconnection logic between this component and the affected processes.

The clock divider is included as a FPGA platform component through mapping rules of platform configuration. Such a component is included in the generated VHDL files when *PeriodicTiming* aspect is used in the UML model. In fact, *PeriodicTiming* mapping rules scripts do not define a clock divider component; they only use its services. The target platform library should provide resources for aspect implementation related to the non-functional requirements handling [32]. Such an approach makes *PeriodicTiming* aspect portable to other platforms, which have their own mechanisms to control the execution frequency of active objects. According to AMoDE-RT approach, embedded system specifications (i.e., models) must be platform independent, in order to allow the reuse of model elements in distinct projects and with different target platforms. Therefore, when GenERTiCA identifies that aspects have been specified in the UML/MARTE model, it includes the necessary platform services used in their implementation, i.e., mapping rules scripts. In this case, the clock divider component is included as a FPGA platform component by means of platform configuration mapping rules, but only when *PeriodicTiming* aspect is used in the model.

3.3.2 DataFreshness

DataFreshness [1] can be understood as the temporal validity of real-time data. Such kind of non-functional requirement is important for some embedded system applications, such as control systems. In such systems, data can only be used whether it is temporally valid. For instance, before using data coming from other components (e.g., sensors), it is necessary to check whether these data are updated. To address such a non-functional requirement, *DataFreshness* aspect [33] associates timestamps with controlled data, checking them before using such data.

VHDL implementation of *DataFreshness* creates constants representing the temporal validity of affected object attributes. An additional process is created within the entity that represents the affected object. This process controls whether values are updated by using a one-bit signal, which, in turn, is checked before any reading access of the controlled attribute.

However, it is worth mentioning that this approach is one possible implementation of *Datafreshness* aspect. Other engineers may handle data freshness non-functional requirements in a different way. This VHDL implementation has been proposed as proof-of-concept for AMoDE-RT aspect-oriented MDE approach, and hence, more efficient implementations are indeed feasible.

3.3.3 COPMonitoring

Computer Operating Properly Monitoring (COPMonitoring) aspect deals with the identification of faults in system components (hardware and software). This aspect adds a mechanism

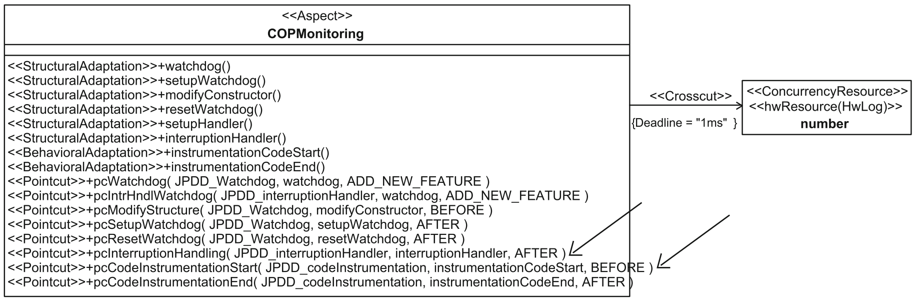
<<Aspect>>
COPMonitoring
<<StructuralAdaptation>>+watchdog()
<<StructuralAdaptation>>+setupWatchdog()
<<StructuralAdaptation>>+modifyConstructor()
<<StructuralAdaptation>>+resetWatchdog()
<<StructuralAdaptation>>+setupHandler()
<<StructuralAdaptation>>+interruptionHandler()
<<BehavioralAdaptation>>+instrumentationCodeStart()
<<BehavioralAdaptation>>+instrumentationCodeEnd()
<<Pointcut>>+pcWatchdog(JPDD_Watchdog, watchdog, ADD_NEW_FEATURE)
<<Pointcut>>+pcIntrHndlWatchdog(JPDD_interruptionHandler, watchdog, ADD_NEW_FEATURE)
<<Pointcut>>+pcModifyStructure(JPDD_Watchdog, modifyConstructor, BEFORE)
<<Pointcut>>+pcSetupWatchdog(JPDD_Watchdog, setupWatchdog, AFTER)
<<Pointcut>>+pcResetWatchdog(JPDD_Watchdog, resetWatchdog, AFTER)
<<Pointcut>>+pcInterruptionHandling(JPDD_interruptionHandler, interruptionHandler, AFTER)
<<Pointcut>>+pcCodeInstrumentationStart(JPDD_codeInstrumentation, instrumentationCodeStart, BEFORE)
<<Pointcut>>+pcCodeInstrumentationEnd(JPDD_codeInstrumentation, instrumentationCodeEnd, AFTER)
<<Crosscut>>
{Deadline = "1ms" }
<<ConcurrencyResource>>
<<hwResource(HwLog)>>
number

Fig. 4 ACOD diagram for *number* class of Digital Watch project

to monitor system components, checking whether they are executing correctly. For that, monitored components must periodically communicate with such a mechanism to inform they are executing without problems. When any component fails to accomplish such a communication, it is considered faulty and the system is warned on such an issue. Thus, the system may take any measure to overcome the problems caused by such a faulty component.

The proposed VHDL implementation of *COPMonitoring* aspect consists in a *watchdog timer*, an interrupt mechanism, and all glue logic that implements the mentioned behavior. One watchdog timer component instance is created for each affected entity, which, in turn, must handle the watchdog timeout interruption, i.e., it must handle the faults identified by the watchdog timer. In this implementation, when a watchdog timeout occurs, the system is reset. Moreover, the behavior of the affected entity is modified to include the logic that will reset the watchdog timer.

Watchdog components shall be provided by the component library of the chosen target FPGA platform. As it happened with the clock divider component in the *PeriodicTiming* aspect, when GenERTiCA identifies that *COPMonitoring* aspect is specified in the UML model, all required services are included into the generated VHDL description. Moreover, it is important to highlight that *COPMonitoring* is the first aspect that deals with faults provided within DERAF framework. This is an additional contribution of this work.

3.3.4 Example of VHDL implementation: COPMonitoring aspect

To illustrate the use of aspects in model level and its transformation to VHDL code, *COPMonitoring* aspect is taken into consideration in the context of the digital watch case study. Figure 4 shows the *COPMonitoring* aspect and its crosscutting relation with *Number* class. In this example, the seven-segment display represented by *Number* class may eventually fail during system runtime. Therefore, *COPMonitoring* is used to identify this fail and then warn the system so that corrective measures may be executed. For that, *COPMonitoring* adds a watchdog timer to the system, which, in turn, must be reset by *Number* component within 1 ms after the previous reset.

The next step is to define the system components that are affected by *COPMonitoring* aspect. In this example, a join point definition is created to select all hardware resource that must be monitored (see Fig. 5a) and other join point selects resources that depend on these resources (Fig. 5b). The *pointcut pcInterruptionHandling* (indicated in Fig. 4) uses *JPDD_interruptionHandler join point* (Fig. 5b) to specify the elements that are affected by

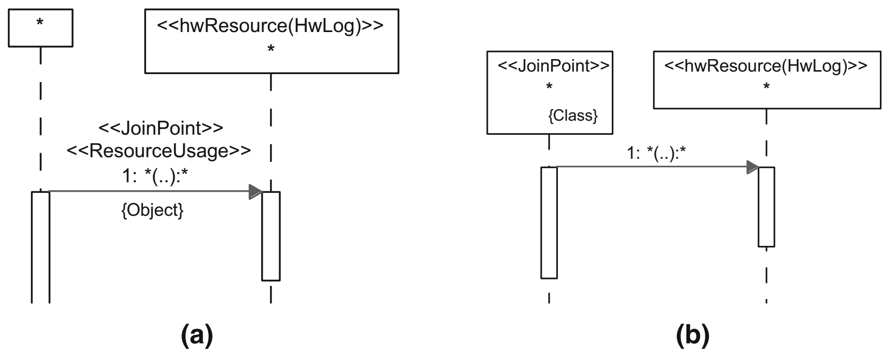

Fig. 5 Join points definitions. **a** Code instrumentation. **b** Interruption handler

the *InterruptionHandler* adaptation. The mapping rule script of this adaptation is shown in Listing 5. On the other hand, Listing 6 shows a fragment of the generated VHDL code for *Number* entity that includes the VHDL statements added by *InterruptionHandler* adaptation of *COPMonitoring* (lines 9–22). This adaptation code has been added automatically during the code generation process by the GenERTiCA aspects weaving engine.

Listing 5 Excerpt from COPMonitoring mapping rules

```
<Structural Name="interruptionHandler" Order="5" ModelLevel="no">
#foreach ($attrWt in $Class.getAttributes())
 #if ($attrWt.getDataType().getRepresent())
  #set ($attWCls = $attrWt.getDataType().getRepresent())
  #if ($attWCls.getTpResource() == 1)
    handlingInterruption${attrWt.Name}:process (${attrWt.Name}watchdogReset)
    begin
     if (${attrWt.Name}watchdogResetEVENT and ${attrWt.Name}watchdogReset='1') then
      -- reset the state of signals controlled by architecture
      ${attrWt.Name}reset &lt;= '1';
     end if;
    end process handlingInterruption${attrWt.Name};
  #end
 #end
#end
</Structural>
```

Listing 6 Excerpt from *Number* entity description

```
architecture Behavioral of number is
 ...
 showDigitDivider: clockDiv port map(
  clock => clock,
  threshold => showDigitThreshold,
  clockdiv => showDigitClockdiv);
 begin
  ...
  handlingInterruptionScreenDigit:process (watchdogResetScreenDigit)
  begin
   if (watchdogResetScreenDigit'EVENT
       and watchdogResetScreenDigit='1') then
     -- reset the state of signals controlled by architecture
     resetScreenDigit <= '1';
   end if;
  end process handlingInterruptionScreenDigit;

  showDigit: process  (showDigitClockdiv, reset)
     begin
  ... -- remainder VHDL description is omitted
```

3.4 On the VHDL implementation of the remainder DERAF aspects

DERAF provides a total of 22 aspects classified according the non-functional requirements classification proposed in [10]. Although this work has focused on implementing VHDL mapping rules for *PeriodicTiming*, *DataFreshness* and *COPMonitoring*[7], other DERAF aspects can be implemented as well. However, due to space constraints, this section discusses briefly a possible VHDL implementation for some of these aspects.

Timing package provides aspects that deal with time issues. Besides the *PeriodicTiming* aspect, this package provides the following aspects: *TimingAttributes*, *TimeBoundedActivity*, *SchedulingSupport*. For instance, let us consider the VHDL implementation of *SchedulingSupport* aspect. This aspect inserts a scheduler object in the affected computing nodes. The scheduler object is responsible to control active objects execution, indicating instants at which they must start performing their behavior. In VHDL, all process execute in parallel without the need for any element to explicitly start their execution. However, when one considers reconfigurable FPGA devices, *SchedulingSupport* can be used to control the partial reconfigurations during runtime, and hence, what tasks/components execute on the FPGA during a given time interval. For that, *SchedulingSupport* will add an entity/component[8] to control and handle the FPGA partial reconfiguration. In addition, this aspect must add the logic to: (i) manage the reconfigurable blocks, and (ii) implement the scheduling policy, e.g., FIFO or EDF.

Precision package provides aspects that deal with the precision in meeting time requirements. In addition to *DataFreshness* aspect, this package provides the following aspects: *Jitter*, *ToleratedDelay*, *ClockDrift*. For instance, *Jitter* aspect measures the variance on timing characteristics of the activities performed by the system, e.g., the periodic activation of *active objects*. A possible VHDL implementation for this aspect would be the following. Firstly, a variable or signal is included in the affected processes. This variable stores the pre-

[7] This new aspect has been proposed in this work.

[8] Such a component must be an IP available in the library of the chosen FPGA platform.

vious occurrence of the controlled event, e.g., the start of the execution of a periodic process. This variable would be used to calculate the jitter of the controlled event, i.e., the difference between current and previous occurrences of such an event. If the jitter is greater than the tolerated threshold, a signal indicated such an issue to the affected entity, and hence, it can execute some process to mitigate the jitter effects.

Synchronization package provides the following aspects to deal with non-functional requirements related to the synchronization and the concurrent access control to shared resources: *ConcurrentAccessControl*, *MessageSynchronization*. For instance, *ConcurrentAccessControl* aspect provides means to control the concurrent access to objects, which share their attributes information with other objects. In VHDL, such a non-functional issue might be understood as concurrent access to shared variables or signals by distinct parallel processes within an entity. A simple implementation of this aspect would be to include a busy signal to indicate when a shared variable/signal is being used by any process. Other process will have access to this shared variable/signal only when the busy signal indicates that no one else is accessing it. In addition, it is important to include a logic that implements an algorithm or protocol that assures that only one process is able to set the busy signal to "on".

Communication package provides the following aspects to deal with objects communication in terms of messages sending: *MessageAck*, *MessageIntegrity*, *MessageCompression*. In the case of a VHDL implementation of these aspects, a message sending is mapped to a signal connecting two entities. Therefore, for instance, *MessageAck* aspect may be implemented using a handshaking protocol. For each affected OUT or INOUT port in the sender entity, an additional IN port (i.e., to indicate an acknowledge from the other communication partner) is added to the entity port declaration. Likewise, on the receiver entity, for each IN or INOUT port, an additional OUT port is added. In addition, the logic to set the acknowledge signal is added into the receiver entity, as well as the logic for waiting the acknowledge is added into the sender entity.

TaskAllocation package provides the aspects to handle non-functional requirements related to objects distribution on different computing devices at runtime: *NodeStatusRetrieval*, *TaskMigration*. For instance, *NodeStatusRetrieval* aspect includes a mechanism to gather information on the system dynamic characteristics, such as processing load, message sending and reception rates, and if the computing device is running. A possible VHDL implementation for this aspect would be to include variables or signals to calculate these metrics while the system is running. For instance, all signal assignments that indicate a communication among entities may be subject of this statistics gathering.

Finally, the *Embedded package* provides aspects to monitor and control the usage of system physical resources, e.g., FPGA area, memory, energy. The following aspects are provided: *HwAreaMonitoring*, *HwAreaControl*, *MemoryMonitoring*, *MemoryControl*, *EnergyMonitoring*, *EnergyControl*. For instance, *EnergyMonitoring* aspect relies on the chosen target FPGA platform for providing means to monitor energy consumed by system activities. A VHDL implementation of this aspect would add the logic to measure the energy level at the beginning and at the end of the execution of processes within an entity. In addition, an additional logic is included in order to calculate the energy consumed by these processes, and hence, a measure of the remaining energy can be obtained.

4 Case studies and results

This work has been evaluated through some case studies, namely, a line-following robot, a valve control system and a digital watch. These systems have been designed using AMoDE-RT approach and implemented as an ASIP on a Xilinx Spartan-6 FPGA (model XC6LX16-CS324, Speed:-3) using Xilinx ISE WebPack.

System requirements have been specified in the UML/MARTE model following AMoDE-RT modeling guidelines [33]. Thereafter, GenERTiCA, using the created UML/MARTE model and the proposed set of VHDL mapping rules as input, has been generated a VHDL description. Once the VHDL description is generated, it was synthesized and uploaded into the FPGA by using Xilinx ISE WebPack.

By using the UML-to-VHDL transformation proposed in this work, we have been able to generate complete and synthesizable VHDL descriptions for all three case studies. No manual modification has been performed on the generated VHDL description, since it was synthesized without errors at the first attempt, indicating that the mapping rules scripts have been well specified.

4.1 Line-following robot

The robot case study is composed of two infrared sensors: one detects the left-hand side of the line, while the other one detects the right-hand side. An Arduino Uno board was used as an Analog-to-Digital Converter. It converts sensors analog signals into digital signals, which, in turn, are sent to FPGA board input ports. The robot has two servomotors to spin left/right wheels. The movement control system commands the robot to turn left/right based on sensors input.

This project presents three non-functional requirements: (i) active objects must be periodically activated according to the execution frequency specified in «`TimedEvent`» stereotype; (ii) servomotor faults must be handled within the robot control system; (iii) sensor and motor data have temporal validity. These requirements have been handled through *PeriodicTiming*, *COPMonitoring* and *DataFreshness* aspects, respectively.

In this evaluation, the line-following robot has been designed in two versions. The first one does not use DERAF aspects in UML/MARTE model. Non-functional requirements handling has been implemented manually, i.e., the generated VHDL description was manually modified in order to include these requirements handling. This implementation was done by the same engineer that made the implementation of DERAF aspects. On the other hand, in the second version, DERAF aspects have been used, and the complete VHDL description has been automatically generated. There was no manual modification in the generated VHDL description. The UML model of for both versions consists of one class diagram and five sequence diagrams. Table 6 provides details on these models. It is worth pointing out that both versions have implemented the same set of functional and non-functional requirements. In the first version of the line-following robot, the crosscutting concerns related to non-functional requirements have been implemented directly on the generated VHDL code. In addition, besides very high-level indications of non-functional requirements (by means of MARTE stereotypes), the UML model does not specify any handling of these crosscutting concerns.

Synthesis results are presented in Table 2. First column shows analyzed metrics. Column "FPGA resources" presents the total amount of resources available in FPGA. Third and fourth columns depict metrics for, respectively, object-oriented version (V1) and the aspect-oriented version (V2). GenERTiCA has generated the VHDL description of both versions. However, as mentioned, V1 was modified manually in order to include non-functional requirements

Table 2 Synthesized system metrics on line-following robot design

Metrics	FPGA resources Available	V1 manual Used	V2 DERAF Used	Variation (%)
Slices	2.278	123	262	113
Flip-flops (FF)	18.224	248	693	179.43
Look-up table (LUT)	9.112	389	949	143.95
Input/output blocks (IOBs)	232	20	80	300
Lines of code (LOC)		416	482	15.86
Minimum period (ns)		4.64	4.35	−6.22
Maximum frequency (Mhz)		215.50	229.85	6.65

handling; and V2 was fully generated automatically due to the use of DERAF aspects and the proposed VHDL mapping rules. Finally, last column presents the variation (percentage) in the analyzed metrics of the two versions.

As one can see, Lines of Code (LOC) increased around 16 % in V2. However, the amount of utilized FPGA hardware has doubled, since used FPGA resources increases ca. 184 on average.[9] Such a result indicates that, for VHDL descriptions, an increase in LOC cannot be related to a proportional increase in FPGA resources utilization. In software, on the other hand, this relation can be established, since the binary code size usually increases proportionally to LOC.

Furthermore, even though the generated system ASIP in V2 has a large increase in FPGA resources utilization, its performance is better: it presents a shorter critical path (6.22 % decrease in minimum period in comparison with V1), leading to a greater operation frequency (6.65 % increase). This improvement may have been achieved due to a better modularization of system elements achieved by using DERAF aspects (see Sect. 4.4), since concerns are better separated among components as indicated in *TR* metric presented in Table 5. Such a modularization improvement has also impacted positively on utilization ratio of FFs and LUTs per slice: 2.02 FF/slice and 3.16 LUT/slice in V1; and 2.65 FF/slice and 3.62 LUT/slice. In other words, this could indicate that the place and routing algorithms of the synthesis tool use more efficiently the available slices in V2, since the increase in FF/LUT is not proportional to the increase of slices. However, it is necessary to perform a more detailed analysis to confirm such hypotheses. In larger systems, this difference in utilization ratio may impact strongly on system performance, i.e., the higher the FF/LUT utilization ratio, the lesser the amount of used slices and better the system overall performance. Moreover, these results are different from those presented in [22], where aspects had a lesser impact on area usage (3 %) but did not affect system performance. Finally, it is worth pointing out that the trade-off between performance and FPGA area depends on system requirements and constraints. Hence, engineers shall choose a design approach that meets the projects requirements and constraints.

4.2 Valve control system

The valve control system case study was originally presented in [20] and extended in [21]. The aim of this case study is twofold: (i) demonstrate the practicability of the proposed UML-to-VHDL transformation; and (ii) compare the valve control system created in [21]

[9] 300 % increase in IOB metric in V2 was included in this average amount.

Table 3 Synthesized system metrics on valve control system

Metrics	FPGA resources Available	V1 manual [20,21] Used	V2 DERAF Used	Variation (%)
Slices	2.278	52	12	−76.92
Flip-flops (FF)	18.224	104	18	−82.69
Look-up table (LUT)	9.112	183	42	−49.39
Input/output blocks (IOBs)	232	46	17	−63.04
Lines of code (LOC)		67	206	207.46
Minimum period (ns)		3.250	3.301	1.56
Maximum frequency (Mhz)		307.659	302.897	1.54

with the one created in this work. For such an evaluation, some metrics have been calculated on both case studies. In summary, this system is composed by an automatic valve to regulate the water flow and sensors that provide information about the valve states. The UML model consists of one class diagram and four sequence diagrams, whose details are presented in Table 6.

This case study consists of generating the VHDL description from the mention UML model using both the previous approach [20,21] and the new approach proposed in this work. Thereafter, both implementations have been compared. Table 3 shows the obtained metrics. First column indicates each analyzed metrics, while the second one presents the available resources in Spartan-6 FPGA. Third column shows results presented in [21] whereas fourth column presents results obtained by applying the proposed approach. Finally, fifth column shows differences between this work results and [21].

The results show a considerable improvement in utilized FPGA area for the VHDL description generated by following the proposed approach. Such an improvement has been obtained due to a better modularization of components, which was achieved by applying mapping rules that follow the key principles of object-oriented design. In other words, system functions are better divided between components. A well-structured VHDL description leads to less resource usage, opening room for inclusion of additional functions into system design, and hence, improving system scalability.

On the other hand, when Lines of Code (LOC) is considered, the proposed approach resulted in a considerable increase in the number of generated VHDL lines. Such an increase is related to a greater number of components generated when using the proposed approach. As discussed, due to a better modularization, FPGA area was better utilized in spite of this increase in LOC, i.e. the code generated using the UML-to-VHDL mapping proposed in this work demands about 23 % lesser slices than our previous approach. In addition, there is the small decrease in system performance, i.e., operation frequency decreased from 307 to 302 Mhz (1.54 %). Such a result might had happened due to the increased number of interconnected components, which led to a longer circuit critical path. It is worth noting that system performance can be improved by making optimizations on the proposed mapping rules, aiming enhancements on the generated VHDL description.

4.3 Digital watch

The digital watch case study has already been introduced in Sect. 3. For this system, the following non-functional requirements have been identified: periodic refresh of display; periodic

Table 4 Synthesized system metrics on digital watch

Metrics	FPGA resources Available	V1 manual Used	V2 DERAF Used	Variation (%)
Slices	2.278	73	70	−4.28
Flip-flops (FF)	18.224	140	184	23.91
Look-up table (LUT)	9.112	231	244	5.32
Input/output blocks (IOBs)	232	48	50	4
Lines of code (LOC)		385	438	12.10
Minimum period (ns)		4.947	4.910	−0.75
Maximum frequency (MHz)		202.137	203.676	0.75

time counting; and identification of display faults. These requirements have been handled by *PeriodicTiming* and *COPMonitoring* aspects. As in the line-following robot, the digital watch system has been designed in two versions. The first one does not use DERAF, and hence, the non-functional requirements handling has been implemented manually on the generated VHDL description. The second one uses DERAF and the generated VHDL description has been synthesized without modifications.

For this case study, the created UML model consists of one class diagram and six sequence diagrams. Table 6 shows the details of this model It is worth mentioning that, although the higher number of messages within the sequence diagrams, the digital watch is a small system. Such an amount of messages is the result of describing the seven-segment display main behavior using a sequence diagram rather than a state machine diagram, which would present this behavior in a less verbose and more elegant way. These diagram covered system functional and non-functional requirements and provided the details needed to code generation.

Table 4 shows the synthesis results. One can notice version V2 of the digital watch uses less slices, whereas there is an increase in the number of FF, LUT and IOBs. It can also be observed a small increase in system performance (0.75 %) and an increase of LOC, as it happened in the line-following robot case study. This corroborates with the claim that there is no relationship between the amount of LOC and the FPGA area usage. Moreover, the better modularization achieved in version V2 (see Sect. 4.4) might had helped the place/route algorithms of the used synthesis tool, leading to a better used of FPGA resources. However, it is important to highlight that such an indication is based on empirical knowledge rather than on solid analysis, and hence, in order to develop a definitive conclusion on this issue further analyses are needed.

4.4 Assessing the impact of DERAF on generated VHDL descriptions

This section discusses the influence of DERAF aspects on the generated VHDL descriptions for handling the crosscutting concerns related to non-functional requirements. Table 5 shows this analysis through some metrics that have been calculated on the generated VHDL descriptions of all case studies. The first three columns indicate the number of code lines for: (i) the architecture description of VHDL entities without any aspect (*LOC**), (ii) the architecture description of VHDL entities with aspects (*LOWC*), and (iii) the implementation of all used aspect adaptations (*LOAC*). The remaining columns indicate AOSD-specific metrics. *CDLOC* [9] indicates the number of context switches between functional and non-functional requirements handling code. One can see that LOAC varies among the case studies. Although

Table 5 Impact of DERAF on projects design

Aspect	Project	LOC*	LOWC	LOAC	CDLOC	TR %	AB
PeriodicTiming	Robot	206	277	34	44	15.88	02.08
	Valve	181	198	31	12	06.00	00.54
	Watch	346	385	31	36	09.00	01.25
	AVG	244.33	286.67	32.00	30.67	10.29	01.29
	SD	72.61	76.65	01.41	13.60	04.14	00.63
DataFreshness	Robot	206	319	84	28	08.77	01.34
	Valve	181	233	69	28	12.00	01.85
	AVG	193.50	276.00	76.50	28.00	10.39	01.60
	SD	12.50	43.00	07.50	00.00	01.62	00.26
COPMonitoring	Robot	206	299	98	50	16.72	00.94
	Watch	346	396	64	20	05.00	02.00
	AVG	276.00	347.50	81.00	35.00	10.86	01.47
	SD	70.00	48.50	17.00	15.00	05.86	00.53
General average		238.86	301.00	58.71	31.14	10.48	01.43
General Std dev		68.55	67.84	25.25	12.28	04.24	00.53

LOC*: Lines of Code; LOWC: Lines of Woven Code; LOAC: Lines of Adaptation Code; CDLOC: Concern Diffusion over LOC; TR: Tangling Ratio; AB: Aspectual Bloat

the DERAF aspects adaptation have been implemented once[10] and reused throughout all case studies, not all aspect adaptations have been used in all case studies. This happened due to the requirements of each systems that did not demand the use of all adaptations, e.g., all adaptations of *PeriodicTiming* are used in the line-following robot, whereas one of them was not necessary in the other case studies.

CDLOC and LOC* are used to calculate the *Tangling Ratio (TR)*. According to [2], *TR* indicates how much functional and non-functional concerns are intermixed. The obtained average value of 10.48 % is similar to results obtained in [2]. This indicates that the impact of DERAF aspects on the generated VHDL code is around 10 % for these case studies. It is important to highlight that higher TR indicates a more intermixed concern code within functional code, whereas lower TR indicates a more localized concern code. For instance, *PeriodicTiming* and *COPMonitoring* present higher impact on the line-following robot than on the other case studies. On the other hand *DataFreshness* has a higher impact on the valve control system. This impact is related to the amount of elements affected by these aspects, which, in turn, is associated to the non-functional requirements of each system.

Aspect Bloat (AB) [9] metric indicates the efficiency on using an aspect. AB is calculated considering the amount of VHDL code lines generated by DERAF aspects, as well as the amount of lines that have been written as DERAF adaptation scripts. In addition, AB indicates the impact of aspects on reusability throughout the case studies. According to [2], AB values lower than 1 indicate a low efficiency of the aspect because the number lines used to implement the aspect is higher than the lines woven in the actual system implementation. Therefore, when aspects adaptations are applied on many join points, aspect efficiency increases. AB average value is 1.43, indicating that, in average, for each aspect adaptation mapping rule script 1.43 lines of VHDL have been generated. One may observe that the most effective

[10] i.e. mapping rules scripts for these adaptations have been completely written once.

Table 6 Overview of projects features

Features	Robot	Valve	Watch
Class diagram			
Classes	3	5	6
Attributes	20	8	11
Methods	18	18	17
Sequence diagram			
Messages	48	16	118
Combined fragments	14	1	10
DERAF			
Aspects	3	2	2
Join points	7	2	5
Source code			
VHDL files	6	7	7
LOC	482	206	438
Metrics			
LOC/classes	160.67	41.20	73.00
LOC/(classes+aspects)	80.33	29.43	54.75

LOC refers to the total number of lines in the whole VHDL description excluding comments and blank lines

aspect is *DataFreshness* since its average AB is 1.60. On the other hand, it may be seen that AB value of *PeriodicTiming* is lesser than 1 in the valve control system, as well as AB value of *COPMonitoring* in the line-following robot case study. In the other case studies, AB metric of these aspects is higher than 1, indicating a higher efficiency. Additionally, it is important to highlight that, when the same implementation is reused without modifications in several case studies, AB may be calculated considering this historical data, and hence, the aspects efficiency might improve due to their reuse.

Considering the engineering cost, a comprehensive discussion on applying AMoDE-RT in the design of three other embedded systems[11] is presented in [32]. Although the present work does not evaluate the same set of high-level design metrics, this work demonstrates and evaluates the impact of AMoDE-RT on the VHDL implementation of three real-world embedded systems. For that, a set of lower-level metrics has been analyzed. These metrics consider the system implementation rather than the UML model, and hence, the results presented in this section are complementary to the ones discussed in [32]. The effort for obtained the VHDL implementation is decreased due to the reuse of DERAF and the VHDL mapping rules scripts. The main reasons of such a decrease are: (i) DERAF aspects have been used within the UML model of the three case studies without any modification on their adaptations—engineers need only to describe the affected elements by specifying the *pointcuts*, which may vary for different application; (ii) UML-to-VHDL mapping rules have been created for the line-following robot and reused without modifications in the other case studies; (iii) AB metric is greater than 1 (in average), indicating that aspect added more lines of VHDL code than the number of script lines actually written by the engineers. (iv) UML-to-VHDL mapping rules have been created in the line-following robot and reused without

[11] The case studies presented in [32] are: (i) the movement control of an unmanned helicopter; (ii) the control systems of an industrial packing system; and (iii) the control system of an electric wheelchair. Those systems are larger and more complex than the ones presented here.

modification in the other case studies. In addition, considering the statistics on the UML models and LOC of each case study (see Table 6), it is possible to see that GenERTiCA was able to generate 91.63 lines of VHDL code per class. These results corroborate with the results discussed in [32], indicating that, by using AMoDE-RT and its tools, it is possible to decrease the design effort in both system specification and implementation.

4.5 Pitfalls and limitations

Although the achieved results indicate that the expected benefits have been obtained by applying AMoDE-RT and the proposed UML-to-VHDL mapping, some factors may influence the conducted evaluation and the obtained results, as it is usual in any empirical study.

Despite the proposed work has been successfully applied to design of a FPGA-based embedded system, the proposed VHDL implementation of DERAF aspects presents some issues. Some aspects adaptation implementation have interfered in the project specification, e.g., *DataFreshness* reads signal value associated to an OUT port, but such a situation is forbidden in VHDL. To overcome this problem, this port type had to be changed to INOUT, so that this signal could be read internally within the entity. Other issue is related to the interference among aspect adaptations. For instance, a signal created in *COPMonitoring* had interfered the implementation created for *DataFreshness*, and hence, *COPMonitoring* implementation, i.e., its mapping rule scripts, had to be modified.

Other issue is regarding the proposed UML-to-VHDL mapping rules. Such mapping rules have been created with the focus on keeping the high-level semantics of the UML model in the generated VHDL description, in order to facilitate the hardware/software co-design and design space exploration. This may be achieved by means of using distinct hardware and software mapping rules on the same source UML model. At the moment, this research has not evaluated the effect of different kinds of strategies for the generated VHDL implementation, e.g., optimizations for area or speed, keeping/flattening hierarchy, or using behavioral or structural description style. Therefore, the obtained FPGA slices utilization rate, as well as system performance, might be directly affected due to a distinct description strategy.

Finally, it is important to mention that engineers skills may affect the results discussed in this paper. Engineers knowledge of UML or the application domain may impact the quality of the produced model, leading to inaccuracies in the discussed results. In addition, experience with AMoDE-RT, GenERTiCA as well as the chosen target platform may affect the way engineers describe the mapping rules. Thus, trained engineers might create better mapping rules for a new target language, e.g., SystemC, Verilog, etc.

5 Final remarks

This paper discusses an approach that allows the automatic generation of fully synthesizable VHDL descriptions from high-level UML/MARTE models. By applying MDE and AOSD, the proposed approach deals with functional and non-functional requirements in a platform independent way, opening room for hardware and software co-design of embedded systems.

One of the main contributions of this work is a new definition of UML-to-VHDL mapping rules, which have been implemented as code generation scripts for the GenERTiCA tool. These mapping rules include the support for key object-oriented features supported in UML, namely, encapsulation, inheritance, and 1-to-n associations, as well as the support for synchronous and asynchronous method calls from sequence diagrams. In addition, a VHDL implementation of some DERAF aspects have been created, in order to handle with crosscutting concerns related to system non-functional requirements. Such a handling code is woven into functional requirements code. Hence, it is important to highlight that, in comparison with a previous work [20,21], a greater amount of VHDL statements is generated, resulting in complete and fully synthesizable VHDL descriptions. Engineers do not need to modify manually the generated VHDL files, since our experiments have demonstrated that the generated VHDL descriptions have been synthesized and uploaded without any error into a FPGA device.

In order to assess the proposed work, this paper has presented the design of the control systems of a line-following robot, a valve control system and a simple digital watch. These systems have been implemented as an ASIP on a FPGA development kit. AMoDE-RT has been successfully applied, impacting positively in system implementation, for instance, in components modularization (*TR metric* is around 10 % in average) and system performance (circuit frequency increased 6.65 % in the line-following robot system). It is worth mentioning that occupied FPGA resources increased in the aspect-oriented version of this system. However, slices are used more efficiently. In larger systems, this may lead to a better use of FPGA resources without penalizing system performance. On the other hand, as DERAF aspects have led to a better modularization of crosscutting concerns, their usage opens room for a good reutilization rate. *AB and TR metrics* are indication that corroborate with this inference. As AMoDE-RT advocates for using platform independent specifications, e.g., UML models and DERAF aspects, design complexity management may be improved due to an enhanced opportunity for hardware/software co-design, as well as reuse of artifacts [32].

As future work, more case studies are already being performed. A similar analysis is going to be executed, in order to demonstrate the suitability and feasibility of using platform independent aspects to design FPGA-based embedded real-time systems. New rules for UML elements that are not yet covered will be defined as well, e.g., state machine diagrams and their relationship with sequence diagrams. In addition, other DERAF aspects need to be implemented in VHDL. However, for that, it is important to obtain or create a set of reusable soft IP components, in order to implement aspects adaptations using their services.

Acknowledgments This work is supported by National Council for Scientific and Technological Development (CNPq-Brazil) through the Grant 480321/2011-6.

References

1. Burns W (1994) HRT-HOOD: a structured design method for hard real-time systems. Real-Time Syst 6(1):73–114. doi:10.1007/BF01245300
2. Cardoso JAM et al (2012) LARA: an aspect-oriented programming language for embedded systems. In: Proceedings of the 11th annual international conference on aspect-oriented Software Development, ACM, New York, AOSD '12, pp 179–190. doi:10.1145/2162049.2162071
3. Ciccozzi F, Sjodin M (2012) Enhancing the generation of correct-by-construction code from design models for complex embedded systems. In: IEEE conference on emerging technologies factory automation (ETFA), pp 1–4. doi:10.1109/ETFA.2012.6489716
4. Ciccozzi F, Cicchetti A, Krekola M, Sjodin M (2011) Generation of correct-by-construction code from design models for embedded systems. In: Industrial embedded systems (SIES), 2011 6th IEEE international symposium on, pp 63–66. doi:10.1109/SIES.2011.5953681
5. Ebeid E, Fummi F, Quaglia D (2015) Hdl code generation from uml/marte sequence diagrams for verification and synthesis. Design automation for embedded systems pp 1–23. doi:10.1007/s10617-014-9158-1
6. Elhaji M et al (2012) System level modeling methodology of noc design from uml-marte to vhdl. Des Autom Embed Syst 16(4):161–187. doi:10.1007/s10617-012-9101-2
7. Elrad T (2001) Discussing aspects of AOP. Commun ACM 44(10):33–38. doi:10.1145/383845.383854

8. Engel M, Spinczyk O (2008) Aspects in hardware: what do they look like? In: Proceedings of the 2008 AOSD workshop on aspects, components, and patterns for infrastructure software, ACM, New York, ACP4IS '08, pp 5:1–5:6
9. Figueiredo E et al (2008) On the maintainability of aspect-oriented software: a concern-oriented measurement framework. In: 12th European conference on software maintenance and reengineering, pp 183–192. doi:10.1109/CSMR.2008.4493313
10. Freitas EP, Wehrmeister MA, Silva Jr ET, Carvalho FC, Pereira CE, Wagner FR (2007) DERAF: a high-level aspects framework for distributed embedded real-time systems design. In: Moreira A, Grundy J (eds) Early aspects: current challenges and future directions, LNCS, vol 4765, Springer Berlin Heidelberg, pp 55–74. doi:10.1007/978-3-540-76811-1_4
11. Habermann AN, Flon L, Cooprider L (1976) Modularization and hierarchy in a family of operating systems. Commun ACM 19(5):266–272. doi:10.1145/360051.360076
12. Hästbacka D et al (2011) Model-driven development of industrial process control applications. J Syst Softw 84(7):1100–1113. doi:10.1016/j.jss.2011.01.063
13. Kiczales G et al (1997) Aspect-oriented programming. In: Proceedings of European conference on object-oriented programming. Springer, Berlin, pp 220–242
14. Leite M, Wehrmeister MA (2014) Aspect-oriented model-driven engineering for FPGA/VHDL based embedded real-time systems. In: International symposium object-oriented real-time distributed computing (ISORC), IEEE Computer Society, pp 261–268. doi:10.1109/ISORC.2014.45
15. Leite M, Damiani CV, Wehrmeister MA (2014) Enhancing automatic generation of VHDL descriptions from UML/MARTE models. In: Proceedings of international conference on industrial informatics (INDIN'14), IEEE Eletronics Society, Piscataway, NY, pp 1–5
16. McUmber W, Cheng BHC (1999) Uml-based analysis of embedded systems using a mapping to vhdl. In: IEEE international symposium on high-assurance systems engineering, pp 56–63. doi:10.1109/HASE.1999.809475
17. Meier M, Hanenberg S, Spinczyk O (2012) AspectVHDL stage 1: the prototype of an aspect-oriented hardware description language. In: Proceedings of the 2012 workshop on modularity in systems software, ACM, New York, MISS '12, pp 3–8. doi:10.1145/2162024.2162028
18. Mohagheghi P et al (2013) An empirical study of the state of the practice and acceptance of model-driven engineering in four industrial cases. Empir Softw Eng 18(1):89–116. doi:10.1007/s10664-012-9196-x
19. Monmasson E, Cirstea M (2007) FPGA design methodology for industrial control systems—a review. Ind Electron, IEEE Trans 54(4):1824–1842. doi:10.1109/TIE.2007.898281
20. Moreira T et al (2010) Automatic code generation for embedded systems: from UML specifications to VHDL code. In: Proceedings of 8th IEEE international conference on industrial informatics (INDIN), pp 1085–1090. doi:10.1109/INDIN.2010.5549590
21. Moreira TG (2012) Geração Automática de Código VHDL a partir de Modelos UML para Sistemas Embarcados de Tempo-Real. Master thesis (in portuguese), UFRGS, Porto Alegre. http://hdl.handle.net/10183/55444
22. Muck T, Gernoth M, Schroder-Preikschat W, Frohlich A (2011) A case study of AOP and OOP applied to digital hardware design. In: Brazilian symposium on computing system engineering (SBESC), pp 66–71. doi:10.1109/SBESC.2011.23
23. Park SH (2006) ADH, aspect described hardware-description-language. Master of engineering, University of Canterbury, College of Engineering, Christchurch. http://hdl.handle.net/10092/1113
24. Quadri I et al (2008) MARTE based modeling approach for partial dynamic reconfigurable FPGAs. In: Proceedings of embedded systems for real-time multimedia, pp 47–52. doi:10.1109/ESTMED.2008.4696994
25. Salewski F, Taylor A (2008) Systematic considerations for the application of FPGAs in industrial applications. In: IEEE international symposium on industrial electronics, pp 2009–2015. doi:10.1109/ISIE.2008.4677068
26. Vanderperren Y, Mueller W, Dehaene W (2008) Uml for electronic systems design: a comprehensive overview. Des Autom Embed Syst 12(4):261–292. doi:10.1007/s10617-008-9028-9
27. Vidal J et al (2009) A co-design approach for embedded system modeling and code generation with uml and marte. In: Design, automation test in Europe conference exhibition (DATE), pp 226–231. doi:10.1109/DATE.2009.5090662
28. Wang Z et al (2008) A model driven development approach for implementing reactive systems in hardware. In: Proceedings of specification, verification and design languages (FDL), pp 197 –202. doi:10.1109/FDL.2008.4641445
29. Wehrmeister M, Freitas E, Pereira C, Rammig F (2008) GenERTiCA: a tool for code generation and aspects weaving. In: International symposium object-oriented real-time distributed computing (ISORC), IEEE Computer Society, pp 234–238. doi:10.1109/ISORC.2008.67

30. Wehrmeister M, Freitas E, Pereira C (2009) An infrastructure for uml-based code generation tools. In: Rettberg A, Zanella M, Amann M, Keckeisen M, Rammig F (eds) Analysis, architectures and modelling of embedded systems, IFIP advances in information and communication technology, vol 310. Springer, Berlin Heidelberg, pp 32–43. doi:10.1007/978-3-642-04284-3_4
31. Wehrmeister MA, Leite M (2014) On generating VHDL descriptions from aspect-oriented UML/MARTE models. In: Proceedings of the Brazilian symposium on computing system engineering (SBESC), IEEE Computer Society, pp 1–6
32. Wehrmeister MA, Pereira CE, Rammig F (2013) Aspect-oriented model-driven engineering for embedded systems applied to automation systems. IEEE Trans Ind Inf 9(4):2373–2386. doi:10.1109/TII.2013.2240308
33. Wehrmeister MA, de Freitas EP, Binotto APD, Pereira CE (2014) Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. Mechatronics 24(7):844–865. doi:10.1016/j.mechatronics.2013.12.008
34. Wood S et al (2008) A model-driven development approach to mapping uml state diagrams to synthesizable vhdl. IEEE Trans Comput 57(10):1357–1371. doi:10.1109/TC.2008.123