Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems

Marco Aurélio Wehrmeister, Edison Pignaton de Freitas, Alécio Pedro Delazari Binotto, Carlos Eduardo Pereira

Abstract

Recent advances in technology enable the creation of complex industrial systems comprising mechanical, electrical, and logical – software – components. It is clear that new project techniques are demanded to support the design of such systems. At design phase, it is extremely important to raise abstraction level in earlier stages of product development in order to deal with such a complexity in an efficient way. This paper discusses Model Driven Engineering (MDE) applied to design industrial mechatronics systems. An aspect-oriented MDE approach is presented by means of a real-world case study, comprising requirements engineering up to code generation. An assessment of two well-known high-level paradigms, namely Aspect- and Object-Oriented paradigms, is deeply presented. Their concepts are applied at every design step of an embedded and real-time mechatronics system, specifically for controlling a product assembly industrial cell. The handling of functional and non-functional requirements (at modeling level) using aspects and objects is further emphasized. Both designs are compared using a set of software engineering metrics, which were adapted to be applied at modeling level. Particularly, the achieved results show the suitability of each paradigm for the system specification in terms of reusability quality of model elements. Focused on the generated code for each case study, statistics depicted an improvement in number of lines using aspects.

1. Introduction

Modern automation and control systems include electromechanical devices controlled by complex embedded and real-time systems, comprising hardware and software components. These systems enable the creation of “smart” or “intelligent” automation devices which are able to execute autonomously and support fully decentralized decision making. This dramatically changes the architectures (usually centered on Programmable Logic Controllers – PLC) adopted in industrial automation.

The increasing number of functionalities incorporated into these modern embedded and real-time systems may require not only specialized hardware and software components, but also their deployment over different processing units, being possibly physically separated. Real-time constraints affect both processing and communication. Handling such requirements cannot violate other system constraints and/or requirements. Therefore, engineers must deal not only with the design of software and hardware in the same project, but also with their interaction which is generally implemented via industrial communication protocols [37,14]. In this sense, it is important to minimize (or even avoid) inconsistencies in system specification, i.e., software and hardware teams must follow the same consistent system specification basis.

Furthermore, the non-functional nature of some important requirements of embedded systems can lead to several problems, such as scattered and tangled handling. If they are not properly handled, these problems increase the overall design complexity, affecting effort and project timeline. In this case, reuse of previously developed artifacts (e.g., software and/or hardware blocks) becomes harder. Additionally, software and hardware components are usually designed concurrently and using distinct languages, tools, and concepts, considerably increasing design complexity.

Several works propose the raising of abstraction level and separation of concerns in order to manage the growing complexity. Some works propose the use of high-level concepts from Object-Oriented (OO) paradigm [50,6]. However, to specify the handling
of non-functional requirements using only the concepts available in OO paradigm is not adequate. OO paradigm lacks convenient abstractions to represent and encapsulate non-functional requirements handling. More precisely, non-functional requirements handling is scattered and intermixed within many objects responsible for handling functional requirements. To overcome such problems at implementation level, subject-oriented programming [35] and Aspect-Oriented programming [28] have been proposed. Both works promote a partitioning of crosscutting concerns in units of modularization called subjects and aspects, respectively.

Model-Based Engineering [38,21] and/or Model-Driven Engineering (MDE) [43,1] are approaches intended to raise abstraction level by using models as the main artifacts created during design. The main idea is to create a Platform Independent Model (PIM) which is refined via model transformations into a Platform Specific Model (PSM). MDE can be seen as a trend in designing embedded systems for automation applications and several approaches have been proposed in the last years, like [48,23,21,1,2].

It is important to highlight that the use of models during design is not a completely new proposal. Other engineering disciplines have been using models for decades. However, particularly for systems comprising hardware and software (which is the case for embedded systems), models can play a more active and important role than only project documentation. Models shall be used to (automatically) generate system implementation through model transformations, keeping specification and implementation synchronized. In addition, model transformations enable correct-by-construction implementations, provided that models and transformations are formally and semantically proved. Meanwhile, intermixing the handling of requirements from different natures at modeling level is still a challenge task.

Within this context, this paper discusses the use of MDE and separation of concerns for handling functional and non-functional requirements. This article presents in details an innovative approach called AMoDE-RT – Aspect-oriented Model Driven Engineering for Real-Time systems, which combines Unified Modeling Language (UML)1 with concepts of Aspect-Oriented Software Development (AOSD) [22]. AMoDE-RT conceptualizes the separation of concerns with the handling of functional and non-functional requirements from earlier design stages (e.g., requirements engineering and modeling phases) to final implementation.

Practically, AMoDE-RT is presented throughout this text by means of a case study representing a real-world industrial mechatronics system, namely the control system for a product assembler industrial cell. Therefore, this paper contributes to the following goals: (i) apply AOSD concepts together with UML at modeling level; (ii) demonstrate the use of UML to model a concrete embedded and real-time system for controlling an industrial mechatronics system; (iii) assess both OO and AOSD in terms of UML models through software engineering metrics, comparing their strengths and weaknesses; and (iv) promote a discussion on the use of AOSD within design of embedded and real-time systems applied to mechatronics systems.

AMoDE-RT increases the reuse of artifacts produced during design, as it allows for a better separation of concerns in requirements handling. For that, the Distributed Embedded Real-time Aspects Framework (DERAF) has been developed to be used during the whole design process, from initial phases until system implementation. This framework provides a predefined set of aspects which deals with non-functional requirements commonly found in automation systems (see Section 5). Although there are crosscutting concerns related to functional requirements, this work concentrates on those associated with non-functional requirements.

The presented results demonstrate indicators on the effectiveness of AMoDE-RT to design embedded and real-time systems for industrial automation applications.

This article is organized as follows: Section 2 discusses related work, followed by Section 3 that provides an overview of AMoDE-RT design flow. Section 4 introduces the case study used throughout this text. The requirements engineering process proposed in AMoDE-RT is described in Section 5. The specification of functional requirements using UML is discussed in Section 6 while non-functional requirements specification is approached in Section 7. Section 8 presents the tools created to support the AMoDE-RT approach. An assessment of AMoDE-RT and results obtained for the case study are presented in Section 9, which includes a discussion on reusability of design artifacts and also on applying MDE in industry. Finally, conclusions and future work directions are discussed in Section 10.

2. Related work

This section briefly discusses some relevant related work on applying MDE in the domain of automation and embedded systems. An interesting survey on using UML in mechatronics systems design is presented in [50]. The author identifies an increasing number of researchers proposing the use of UML as a specification language to complement traditional approaches, such as those using MATLAB, Simulink, and Statecharts.

Traceability is an important feature in MDE to enhance automated analysis, consistency, and coherence of models used during software development. The work of [36] contributed with a management approach for the complexity of traceability information in MDE by means of identifying trace-links in a MDE process and of defining semantically rich trace-links between models. Identifying rich trace-links is crucial to maintain traceability during software development.

Targeting software product line engineering, [17] shows important issues to include an aspect-oriented MDE tool-chain in the software development production line process. The Aspect-oriented Model-driven Product Line Engineering (AMPLE) project2 proposed an aspect-oriented MDE methodology for Software Product Line (SPL), aiming to improve modularization of software variations and maintenance of their traceability during SPL evolution. The lifecycle proposed in AMPLE comprises early activities as requirements engineering, as well as architecture definition and implementation activities of a software based on SPL. In this sense, AMoDE-RT is similar in the way it uses AOSD and MDE techniques and covers a similar range of activities. However, AMoDE-RT focuses not only on software but also on hardware components of an embedded system. This is achieved by using a Platform-based Design approach, which comprises platforms that provide hardware and software components, and also model transformations and code generation. Nonetheless, AMoDE-RT is more restricted when using AOSD, since it applies aspects to deal with non-functional crosscutting concerns, whereas AMPLE deals with both functional and non-functional ones.

The work presented in [20] applies Theme/UML, an aspect-oriented MDE approach to separate embedded system concerns from earlier design phases to system implementation, reducing design complexity. That work illustrated the Theme/UML approach using a design of a packmaker as case study, from modeling down to code. According to the authors, Theme/UML has some limitations: (i) it addresses only non-functional concerns that manifest themselves as code in the system; (ii) inheritance is not supported in UML-to-C transformation; and (iii) the behavior specified within aspects can be specified only with sequence diagram, leading to
composition problems when state diagrams are used to specify functional requirements. However, despite these limitations, this approach seems to be useful in managing design complexity, since it uses a higher abstraction level and transformations supported by CASE tools to produce system implementation.

At last, there is a pragmatic lack of consensus regarding the acceptance of MDE in industry. The work of [32] investigated the state of practice for its adoption using several subjective metrics among users from different companies. Based on users feedback, the authors concluded that MDE is useful when applied to the development of complex systems, but it is still missing effective and easy-to-use tools to perform such development. In addition, another practical example of using MDE combined with an aspect-oriented approach in industrial applications was carried in previous work [8]. It demonstrates the practical use of developed tools during requirement and modeling phases of a complex Computational Fluid Dynamics application, from design to code generation for specific processing units. The work provided a tool to automatically generate code for CPU or GPU (Graphics Processing Unit). Once a task is scheduled to CPU or GPU, the correspondent code is generated. However, the authors concentrated on scheduling issues and not on discussing the benefits of using MDE with an aspect-oriented approach in lieu of default object-oriented approach or direct coding.

On the other hand, aspect-oriented proposals in the area of MDE have focused mainly in providing design notations to support representation of aspect-oriented elements in system models [16]. A remarkable proposal following this trend is presented in [18]. It describes a standard design language for aspect-oriented software development. A study about traceability between Theme/UML and AspectJ is presented, but it still lacks a full roadmap from specification of non-functional requirements to system code generation. On the other hand, AMoDE-RT provides support for the whole process, starting at requirements specification (identifying the crosscutting concerns for handling functional and non-functional requirements), passing over system model design (addressing these concerns), and finalizing with the code generation based on the system model.

In [3], a C++ template-based code generation approach is presented. That work focuses on a specific class of non-functional requirements, i.e., Quality of Service (QoS). A similar template-based approach, but with a broader scope is presented in [11]. These approaches provide a method to specify non-functional requirements that have similarities to our requirement identification and specification phase using templates. But, it is closer to the code that will be generated – in the case of [3], C++ code. Despite the similarity (their approaches are closer to the code), they do not cover design phase as in AMoDE-RT. It is possible to define that our approach provides better portability to different target programming languages compared to theirs.

Following, an approach to assess aspect-oriented generated code is presented in [47]. Feature models are used to capture variations in design strategies, architectural specifics, and technology platforms, of an aspect-oriented developed system. This approach provides means to size aspect-oriented code generated by MDE tools. Despite the different focus, this work is related to ours due to the tracking of variations in the development of aspect-oriented code to provide the assessment that they aim to. In our case, it is provided tracking of non-functional concerns to system design; and generated code by different tools that is proposed to use, such as the mapping table. Using these tools, it is possible to track decisions made about the non-functional requirements in early phases of system development to the generated code. In their work, the feature models perform a similar role, but focusing on code sizing assessment due to system variations.

In summary, this work goes beyond the MDE state-of-the-art by addressing a whole integrated process: from requirements engineering and early verification of system specification to its implementation using a given target platform comprising hardware and software. In other words, AMoDE-RT provides not only an integrated design flow, but also the necessary toolset support for each phase and a novel approach that separates the handling of functional requirements from non-functional ones from the initial phases throughout the whole design cycle.

However, there is still a need for studying system engineering, more specifically to demonstrate benefits of such an aspect-oriented approach in a wide range of applications and in comparison to an OO approach. In this paper, a framework that presents aspect-oriented modeling and implementation of an industrial packaging system with traceability support is deeply discussed. In addition, its generalization using an independent execution platform (and a comparison to common OO approach) is addressed by analyzing pros and cons of this modeling strategy in industry.

3. AMoDE-RT: MDE approach for mechatronics systems

AMoDE-RT – Aspect-oriented Model Driven Engineering for Real-Time systems aims to improve design of real-time and embedded systems that control electro-mechanical equipments of industrial automation systems. For that, AMoDE-RT raises the abstraction level carried out during design, and additionally enhances modularization of system requirements handling.

The main concepts of this method are based on: (i) a smooth design flow starting from requirements engineering to implementation, including early verification of specifications; (ii) MDE and UML to improve communication between software and hardware teams by allowing them to use the same high-level specification of system structure and behavior; (iii) concepts of AOSD are used to enhance the separation of concerns on handling functional and non-functional requirements; (iv) code generation techniques and tools to automate the transition from system specification to implementation; (v) platform-based design to allow the mapping of high-level elements (from UML model) into a target execution platform that will deploy the designed system; and (vi) automation techniques for execution of test cases in earlier stages of design.

Fig. 1 outlines AMoDE-RT design flow. A design flow covers several phases – from requirements engineering to the system implementation itself. The first step focuses on gathering requirements and constraints of the real-time system (see Fig. 1, steps #1 and #2). Following, the created requirements documents are used to specify system structure and behavior using UML diagrams annotated with stereotypes (see Fig. 1, steps #3 and #5). These diagrams specify the handling of functional requirements. The resulted UML model hides detailed information about the deployment technology, focusing just on concepts that are closer to the target application domain. In this sense, higher abstraction levels are easier to understand and allow designers to focus on applications foundations instead of concerning about implementation issues. On the other hand, the handling of non-functional requirements is then specified using a high-level aspects framework named Distributed Embedded Real-time Aspects Framework (DERAF) [53] (see Fig. 1, step #4).

In order to enable automated manipulation of UML models, it is necessary to transform the UML model into other PIM, which provides a more concise meta-model and also support concepts of aspect-oriented paradigm. For that, AMoDE-RT created the Distributed Embedded Real-Time Compact Specification (DERCS) (see Fig. 1, steps #5–#7). Therefore, the DERCS model is used as input for both behavior simulation and code generation.

The UML model is verified during specification phase by means of simulation and test cases execution. The system behavior can be simulated in parts, while the UML model is being created (see the
activities iteration loop represented in steps #3–#8 of Fig. 1). Thus, specification errors may be found earlier through design cycle. Once the specification details are considered sufficient, system implementation activities take place. Additionally, the created UML model is used as input to a code generation tool, named GenERTICA, which is able to generate source code for different target platforms (see Fig. 1, steps #9–#11). Thereafter, the generated code is compiled/synthesized using third-party tools that support the chosen target platform, and the executable implementation of the embedded real-time system is created (see Fig. 1, steps #12 and #13).

It is important highlight that, as usual in platform-based approaches, AMoDE-RT relies on existing technologies and platforms for system implementation. Therefore, existing hardware and software components, which may be developed in-house or obtained from third-party suppliers, may be reused. Evidently, the components integration is possible after an analysis of the quality-of-services characteristics of each component. Such an analysis is performed during the specification phase, and may demand changes on system requirements, including on the non-functional ones. Thereafter, the engineers need to specify the mapping rules from UML to the services requested/provided by these components.

The remainder of this text discusses the details of AMoDE-RT phases. A case study representing an industrial mechatronics system is used to illustrate each step of AMoDE-RT.

4. Brief overview of the industrial packing system case study

To illustrate AMoDE-RT, an real-world example of an industrial mechatronics system, namely an Industrial Packing System (IPS), will be used throughout the text. This case study is based on the industrial mechatronics system presented in [24,10].

In summary, IPS is a system composed of a robotic arm with a gripper, two conveyor belts, a storage unit, and several sensors. The input conveyor belt brings individual parts, which are combined to form products. The conveyor belt stops when the sensor detects the presence of a part. Then, the robotic arm will either put it in the storage unit or use it to assemble a product. The second conveyor belt brings empty boxes into which parts are inserted. This conveyor belt remains operating until its sensor detects an empty box at the expected position. When the product is completely assembled, the controller sends a command to the conveyor belt and it starts to move forward again. The controller is a periodic active object that verifies whether there are products to assemble and/or parts to be placed into the storage unit. When the new product requires a part, which is physically located in the parts conveyor belt, this part is taken from this conveyor and used to assemble the product; otherwise, the part is taken from the storage unit. This system is intended to be distributed, i.e., there are four different processing nodes: one responsible for controlling the products assembly process and the robotic arm; two nodes to control, respectively, the input parts conveyor belt and the assembled products output conveyor belt; and one to control the amount of parts in the storage unit.

The IPS functionalities are specified with a use case diagram, as depicted in Fig. 2. Note that several use cases are annotated with stereotypes related to non-functional requirements, i.e., decorated with “NFR_” stereotypes. It is important to emphasize that a discussion on the complete specification of the industrial packing system is out-of-the-scope of this paper due to space constraints. However, based on this overview, it is possible to understand how AMoDE-RT works, mainly on the identification of non-functional requirements that crosscut the system functionalities, such as the timing control of sensor samples acquisition and processing distribution. In the examples, the focus is concentrated in one specific non-functional requirement: periodicity.

Details regarding the proposed workflow is depicted in the following sections – step by step, from design to code generation targeting a technology – using the IPS case study as reference.

5. Requirements engineering

The requirements engineering is the initial step of AMoDE-RT (see Fig. 1, steps #1 and #2). Engineers perform the requirements engineering activities using a method that improves the specification of functional and non-functional requirements commonly found in the domain of real-time and embedded systems. This method is introduced here as RT-FRIDA (Real-Time FRIDA), which is an extension of FRIDA [7], a method oriented to the fault
tolerance domain with a vocabulary and a toolset tailored to support the analysis of fault-tolerant systems. RT-FRIDA provides a consistent way to deal separately with the functional requirements and the non-functional ones, including the early phases up to the system design. The main contributions of RT-FRIDA concentrate on both the system analysis and the mapping of requirements into design elements.

In order to adapt FRIDA to the domain of real-time and embedded systems, a first step is to identify concerns related to this domain. Some key requirements are shown in Fig. 3. Those requirements are based mainly on [12], IEEE glossary [27], and SEI glossary [42]. Based on this classification, some tools of FRIDA were adapted to consider those requirements. It is important to highlight that many real-time and embedded systems present fault tolerance requirements. Thus, each requirement considered in the original FRIDA method is also supported in RT-FRIDA.

According to the taxonomy presented in Fig. 3, real-time concerns are captured in the requirements stated in the Time classification, which is divided in Timing and Precision requirements. The first is concerned with specification of temporal constraints for activities execution, such as established deadlines and periodic activations. Requirements classified as Precision denote constraints that affect temporal behavior of the system, determining whether a system has hard or soft time constraints. Freshness and Jitter requirements can be quoted as examples. The former denotes the time interval within which the value of a sampled data is considered updated; while the latter affects directly system predictability, since a large variance degrades system determinism.

The Performance classification comprehends requirements that are tightly related to those presented in the Time classification, as well as those concentrated in the Distribution classification. Performance requirements are usually employed to express a global need of performance, e.g., the end-to-end response time for a certain activity or required throughput rate.

The goal of Distribution classification is to identify key requirements related to the distribution of system activities over distinct processing nodes. These activities usually execute concurrently. Besides, these concerns address problems such as task allocation over system nodes as well as communication needs and constraints.

Finally, as embedded systems are usually deployed in resource-constrained platform, the Embedded classification presents concerns related to memory usage, power and energy consumption, and also required hardware in terms of area (e.g. number of transistors or logic cells in an FPGA). Usually, these requirements are related to dealing with resources monitoring and control.

RT-FRIDA is divided in three phases: (i) Requirements Identification and Specification; (ii) Requirement Association with Project Elements; and (iii) Design Phase. The high level system analysis is performed in the first phase. Engineers identify and specify system requirements. In the second phase, the requirements are mapped into project elements. These elements are potential candidates for handling their associated concerns. Finally, in the third phase, these project elements are combined in a design solution for the system. As the original FRIDA has been extended to support new requirements, its vocabulary and tools needed to be adapted. The
following subsections discusses the initial phases using RT-FRIDA and their associated set of tools, followed by discussions regarding functional requirements in Section 6 and non-functional ones in Section 7.

5.1. Requirements identification and specification

RT-FRIDA starts once an initial description of the system is available. Therefore, RT-FRIDA first step is the non-functional requirements identification, in which engineers use predefined check-lists. In fact, these check-lists are predefined forms. Their several fields aim to capture information about non-functional requirements presented in system description.

Fig. 4 presents the check list created to identify the periodicity non-functional requirement. The first column lists non-functional requirements, inferring questions organized in the generic classification\(^3\) and its respective sub-classification. The second column explores requirements relevance, while the third column gives its priority (i.e., its attributed importance within system context). Additionally, the last column specifies information about restrictions, conditions, and/or provides requirement description.

After the use of check-lists, it may happen that a given non-functional requirement could not be satisfactorily specified or even it could not be identified at all. In such a situation, a lexicon can be used to refine the identification and specification of such non-functional requirements. This lexicon comprises a set of rules specified in the Backus Naur Form (BNF)\(^4\). An example of a lexicon for time requirements can be seen in Fig. 5. There are specific lexicons for each generic non-functional requirement.

The next step is to fill out a template for each identified non-functional requirement using information provided by inference questions of check-lists. This template summarizes the information necessary to specify a non-functional requirement. Fig. 6 presents information captured in the check-list and further analysis based on lexicon use. Specifically, this template gives support to the identification and specification of the periodicity non-functional requirement.

As it can be observed, the non-functional requirements specification indicates how functional requirements are affected (see Description and Affected Use Cases parts in Fig. 6). The specification consists in a more detailed concern description, indication of a list of affected use cases, and context in which the non-functional requirement shall be handled. The requirements presented in Fig. 6 specifies that the periodicity non-functional requirement affects six functions. For instance, the item detection function shall be executed every 150 ms. Such an information is latter used in the specification of system behavior with UML diagrams.

Additionally, functional requirements are also specified by means of a specific template that gives information about system use cases, being similar to the template depicted in Fig. 6. The main difference is based on fields “scope”, “context”, and “affected use case” that have been removed from the functional requirements template.

At the end of the requirement identification and specification, a use case diagram is then created. The use case diagram created in the IPS case study is presented in Fig. 2. It depicts use cases representing functional and non-functional requirements. Non-functional requirements are represented by means of stereotypes that decorate use cases. This indicates that functional requirements are affected by non-functional ones. Those stereotypes can be either the name of a particular non-functional requirement (e.g., period or deadline) or name of its generic classification according to the taxonomy depicted in Fig. 3.

5.2. Requirement association with project elements

An important task to link the requirement analysis and system design is the decision on how the system will handle specified requirements. Once system requirements are identified, engineers start reasoning about the element which is going to be responsible for each requirement.

As AMoDE-RT relies on concepts of both object-oriented and aspect-oriented paradigms (see Sections 6 and 7), design elements are expressed in terms of objects/classes and aspects, respectively. Therefore, in order to describe relationships between requirements and design elements, a mapping table is created based on information captured at requirements specification phase. Three main tasks must be performed in this phase:

- Using the information specified in the use cases diagram and functional requirements templates, concepts and their attributes are identified as the system functional part. This part consists of objects/classes that will be detailed in the design phase, in which a class diagram is created and populated.
- Identification of aspects using information specified in the use case diagram and non-functional requirement templates. This information defines the aspects (using the high-level framework of aspects presented in Section 7) that will handle each identified non-functional requirement.

\(^3\) As mentioned in Section 5, Time, Performance, Distribution, Embedded.

\(^4\) Please cite this article in press as: Wehrmeister MA et al. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. Mechatronics (2014), http://dx.doi.org/10.1016/j.mechatronics.2013.12.008
Composition of previously extracted information into a mapping table, in order to link identified requirements with project elements. This mapping table ensures the requirements traceability over system life cycle.

Fig. 7 presents the mapping table for IPC case study, in which it is possible to observe the list of the functional requirements on the left and the non-functional requirements on the top. Classes handling the functional requirements are on the rightmost column, in the respective rows for each requirements. On the other hand, the aspects responsible for handling the non-functional requirements are specified in the last row of the table; their names are placed in the column related to the addressed requirements. For the sake of space, just 5 non-functional requirements are presented in this table. Considering the classification of non-functional requirements presented in Fig. 3, the requirements NFR 1, NFR 2 and NFR 3 are related to Timing classification, while NFR 4 is an example of non-functional requirement under the Distribution classification.

---

Fig. 5. Lexicon for requirements of the Embedded classification.

<table>
<thead>
<tr>
<th>Item</th>
<th>Case Study</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifier</td>
<td>NFR-1</td>
</tr>
<tr>
<td>Name</td>
<td>Periodicity</td>
</tr>
<tr>
<td>Author</td>
<td>Stephan</td>
</tr>
<tr>
<td>Classification</td>
<td>Time/Timing/Period</td>
</tr>
<tr>
<td>Description</td>
<td>The system has activities that must be executed in regular periods of time. These activities are: (1) data acquisition from sensors in the conveyor and storage compartment shall be performed every 150ms; (2) robotic arm control shall be performed every 150ms; (3) Gripper control shall be performed every 200ms; (4) Conveyor control shall be performed every 300ms; (5) Overall system control shall be performed every 5s.</td>
</tr>
<tr>
<td>Affected Use Cases</td>
<td>(1) Items Detection; (2) Robotic Arm Control; (3) Gripper Control; (4) Arm Joints Control; (5) Conveyor Control; (6) Assembly Cell Control.</td>
</tr>
<tr>
<td>Context</td>
<td>Each time that a new cycle of data reading or a controlling task starts.</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Priority</td>
<td>8</td>
</tr>
<tr>
<td>Status</td>
<td>5</td>
</tr>
</tbody>
</table>

Fig. 6. Template for non-functional requirements.
classification, and NFR 11 regards both Distribution and Embedded (Memory Allocation) classifications.

Each cell indicates that a given functional requirement is affected by one non-functional requirement. In other words, an “X” sign is marked to indicate that the non-functional requirement (column) affects the functional requirement (row). It is interesting to notice the case of the functional requirement FR 7 – Items Detection, which is critical to the whole system performance. The delayed sampling of the current position of the items may result in an invalidation of this data, since the wrong information about the precise position of the items can compromise the whole packing process. To handle such an issue, the NFR 3 - Expiration was created. This requirement is handled within the DataFreshness aspect, which associates timestamps to data, verifying their validity before using them. When time-validated data are written, their timestamp must be updated. Analogously, before reading such data, the timestamps must be checked and, if the validity is expired, some corrective actions must take place (e.g. read the sensor again and update the value and timestamp). However, it is important to highlight that some functional requirements may be handled by one or more objects/classes, as well as some non-functional requirements may be handled by more than one aspect. Additionally, as one may infer, such a mapping table provides traceability from requirements to design elements and vice versa.

6. Specifying functional requirements

After the phase focused on requirements engineering, the system specification phase is the next step proposed in AMoDE-RT (see Fig. 1, steps #3 – #5). As AMoDE-RT follows MDE principles, engineers describe the intended services and characteristics of the system as high-level models using UML and its profile for Modeling and Analysis of Real-Time and Embedded Systems (MARTE). In other words, engineers specify the elements of the embedded and real-time system in terms of objects that cooperate to provide a result expressed as system expected services. The system is structured as objects (and classes) and their relationships, while its behavior corresponds to objects operations (or methods) and messages exchanged among objects.

Using the current version of UML (2.4.1), there are fourteen different diagrams to specify system structure and behavior, providing flexibility or different ways to model system characteristics. However, information captured in UML models is often redundant, since different diagrams may overlap their information [51]. It is not necessary at this point to use all of these diagrams to specify an embedded and real-time system, since some diagrams are more suitable than others to specify system characteristics in a given application domain.

AMoDE-RT proposes a set of modeling guidelines to assist engineers on specifying UML models for embedded and real-time systems. In addition to the use case diagram (described in the previous section), AMoDE-RT supports class, composite structure, and deployment diagrams for specifying system structure. Class diagrams depict static structure, whereas the other two diagrams depict how passive and active objects are interconnected and deployed over processing units.

System behavior is specified using sequence, activity, and state diagrams. Each of these shows the behavior from a specific viewpoint. A combination of these diagrams can be used to specify a complete system behavior. Any of these diagrams may be annotated with MARTE profile stereotypes as it will be discussed later in this section. In an overview, UML diagrams are used as follows:

- **Use case diagram**, as mentioned in previous sections, depicts system services and external elements that interact with these services.
- **Class diagram** shows a system static structure in terms of classes, their attributes and operations, and also relationships among classes. Classes depicted in this diagram may be annotated with MARTE profile stereotypes as discussed later in this section.
- **Composite structure diagram**, on the other hand, presents system dynamic structure, i.e., active and passive objects and their relationships.
- **Deployment diagram** is used to distribute system objects over computing devices. Using this diagram, it is possible to partition and schedule functionalities between hardware and software. More specifically, engineers specify whether an object (or class) will be implemented as a software or hardware component, which is represented as a node in this diagram. Further, these nodes may also represent processing units that are physically

---

4 http://www.omg.org/spec/MARTE/1.0.

6 A processing unit can be either a processor (for example, a CPU or a GPU), upon which software objects execute, or a hardware device (possibly a configurable device, e.g. FPGA) that implement active objects. Processing units can also be physically separated.

---

Please cite this article in press as: Wehrmeister MA et al. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. Mechatronics (2014), http://dx.doi.org/10.1016/j.mechatronics.2013.12.008
separated. In this case, the relation between nodes represents a communication infrastructure. Sequence diagram shows execution scenarios, in which objects interact with each other through messages exchange. Engineers can also use some reserved words (defined in AMoDe-RT modeling guidelines) as message names to specify other sorts of actions, e.g., expression or assignments. Additionally, this diagram provides constructions to specify repetitions – loops – of action and branches. Activity diagram specifies a system behavior as a control flow model. This diagram may be used to specify system global behavior, which is usually divided into stages (e.g., initialization, execution, and shutdown) that are represented as (coarse-grained) actions. These actions are associated with sequence diagrams, which provide details on the (fine-grained) actions performed within such a stage. On the other hand, there are situations in which a object behavior does not represent interactions among objects. Hence, actions in activity diagrams are specified using the same granularity provided in sequence diagrams, as, for example, assignments, expressions, state changes, among others. State diagram describes system behavior in terms of states and transitions, which are triggered by internal or external events. Actions are performed in three distinct moments: (i) on entering in a state; (ii) while staying at a given state; (iii) on exiting the state. Therefore, engineers associate sequence diagrams to these states to graphically describe the expected behavior instead of using action/programming languages to write code for the specification of such a behavior.

Based on above mentioned diagrams, only use case, class, and sequence diagrams are mandatory. The produced UML diagrams represent a PIM, whose elements are mapped to elements of the target execution platform in later design phases. Hence, AMoDe-RT modeling guidelines must be followed to enable transformation and code generation tools in order to automatically extract information from the high-level specification of system structure and behavior. Moreover, AMoDe-RT is intended to deal with cooperation among objects that are deployed on different computing devices. Hence, it must deal with communication among them. At modeling level, engineers are concerned with objects interactions (through messages exchange), which are specified within sequence diagrams. Engineers do not need to be aware if objects are remote or local, or even if they are software or hardware components. The deployment diagram is used to specify this partitioning. During the code generation phase, GenERTICA (see 8.2) is able to identify such a deployment and select the appropriate script that generates code for a local or remote message sending.

Additionally, MARTE profile stereotypes are used to specify real-time characteristics, e.g., behavior periodic activation, active and passive objects, or shared resources. For instance, <<SchedulableResource>> is used to annotate classes that represent active objects, whereas <<Resource>> or <<MutualExclusionResource>> represents shared objects. One advantage of using standard profiles of OMG (rather than proprietary ones) to extend UML semantics is that the model meaning can be correctly interpreted by different engineers. Thus, the use of AMoDe-RT modeling guidelines in combination with UML and MARTE aids in communication issues of software and hardware teams. Interested readers are referred to [53] for a deeper discussion on AMoDe-RT modeling guidelines.

To pragmatically illustrate the specification of functional requirements, a small part of functional concerns of the IPS case study is taken in evidence. Fig. 8 illustrates a very simple sequence diagram that specifies the ItemReader active object behavior. This diagrams shows a scenario that includes three objects that represents part of the functional dimension of IPS: ItemDetector:ItemReader, Conveyor, Conveyor, DetectedItem:ProductPart. Although the Scheduler object belongs to the non-functional dimension of the IPS, it was specified to depict explicitly that the periodic execution of the ItemReader.run() behavior. It can be noticed the use of some stereotypes of MARTE profile. <<SchedulableResource>> indicates active objects and <<MutualExclusionResource>> this indicates shared passive objects whose services are accessed concurrently by other objects, but in an exclusive fashion, i.e. one object can access a service at a time. Moreover, <<TimedEvent >> indicates that the behavior triggered by ItemReader.run() executes periodically in every 150 ms. Such a timing constraint has been identified and specified in the requirements engineering phase (see Fig. 6).

Finally, as the specification phase is an iterative process, the UML model is refined/detailed in various iteration rounds up to achieve the sufficient information to allow the automatic code generation. However, it is important to highlight that, although detailed, the UML model is still a platform-independent representation of the system under development. During this phase, the engineers may also choose the target platform, which will provide services and hardware/software components to implement the industrial mechatronics system. These services and components may have been developed in-house, or bought from third-party suppliers. Rules are defined to map the UML model elements into the constructs and services provided by the selected components (see Sections 8.2, 9.2, and 9.4).

In addition, it is important to note that each of the selected components provides distinct quality-of-service characteristics for the delivered/requested services. Therefore, an analysis on how to integrate these components must be performed. Such an integration may demand changes on the requirements of the system under development, specially the non-functional ones. For instance, the behavior of a pump control or the conveyor belt length may impact the system timing constraints, and hence, the specified non-functional requirements, as well as the specified UML model, need to be changed to accommodate new constraints. However, as mentioned, the specification phase is an iterative processes, and thus, changes are easily integrated into the UML model and the requirements.

7. Specifying non-functional requirements

AMoDe-RT has extended UML in order to include the support for AOSD concepts to deal with non-functional requirements separately from the functional requirements. This section describes in details the developed aspect framework and how to use it within UML models during modeling phase (see Fig. 1, steps #3, #4, and #5).

7.1. Distributed embedded real-time aspects framework

In order to address the non-functional requirements presented in Section 5, an aspect framework has been created. The Distributed Embedded Real-time Aspects Framework (DERAF) is an extensible high-level aspects framework based on the conceptual model of an aspect-oriented paradigm proposed in [41]. It provides a set of aspects to facilitate handling of timing, performance, distribution, and embedded non-functional requirements to be used at two moments: (i) earlier design phases (e.g., requirements engineering and modeling), and (ii) implementation phase – more specifically, during code generation/aspects weaving step.

The embedded and real-time system specification is done by using a UML model, in which DERA Framework aspects are depicted affecting system elements (see Section 7.2). The main idea behind DERA Framework
to provide aspects which enhance the modeled system by adding specific behavior and structure to handle non-functional requirements. This is performed without binding the system UML model to any specific implementation technology.

To reach this implementation independence, DERAF aspects define high-level semantics for their adaptations, i.e., details about how to implement aspect adaptations are defined only during implementation phase. This means that engineers select aspects to be used during system design based on how these aspects affect the system elements. They define which elements will be affected by selected aspects within UML. Later during design cycle, AMoDERT tools take these specification and select the corresponding implementation for each aspect (see Section 8). The set of aspects available in DERAF is depicted in Fig. 9.

Each concern can be handled by one or more aspects. Thus, a brief description of the behavior/structural adaptations is necessary at this point:

**Timing package** contains aspects that handles time-related requirements.

- **TimingAttributes** adds timing attributes to active objects (for example, deadline, priority, worst-case execution time, start/end time, among others). It also provides the corresponding initialization of these attributes according to information provided in the UML model.
- **PeriodicTiming** adds a mechanism to periodically activate a given behavior of active objects. This improvement requires the addition of an attribute that represents the activation period, as well as a way to control execution frequency according to this period.
- **SchedulingSupport** inserts a scheduling mechanism to control the execution of active objects. Additionally, this aspect handles the inclusion of active objects into a scheduling list and also the feasibility test execution to verify if the list is feasible in terms of scheduling.
- **TimeBoundedActivity** limits the time execution of an activity, i.e., it adds the mechanism to restrict a maximum execution time for an activity, by, for example, limiting the time in which a shared resource can be locked by any active object). The time counting begins immediately before starting the activity. It must provide a way to interrupt execution whether time limit is reached.

**Precision package** deals with precision for meeting time requirements.

- **Jitter** measures start/end time of an activity and calculates the variation of this metric. If a tolerated variance was over-ran, corrective actions must be taken.
- **ToleratedDelay** controls tolerated delay for the beginning of an activity (e.g., it limits the time during which an active object can wait to acquire a lock on a shared resource). This aspect adds a time counter, which is started immediately before an activity execution triggering. If the maximum tolerated delay is reached, corrective actions must take place.
- **DataFreshness** associates timestamps to data and verifies their temporal validity before their use [13]. If a write operation is performed on any controlled data, its timestamp must be updated. Analogously, before reading any data, timestamps must be checked and, if such validity is expired, some corrective actions must take place (e.g., to read the sensor one more time, updating its corresponding value and timestamp).

---

7 In the context of this research, “corrective actions” means any application specific actions to mitigate, tolerate, and/or remove undesired effects that may appear as a consequence of the indicated issues.
ClockDrift measures time at which an activity starts and compares it with the expected beginning of this activity; if accumulated difference exceeds a maximum tolerated clock drift, some corrective actions are also taken.

Synchronization Package provides aspects to deal with non-functional requirements related to synchronization and concurrent access control to shared resources.

ConcurrentAccessControl adds a mechanism to control concurrent access of shared resources. Every time an active object needs to access a shared resource, it requests a lock to the control mechanism. After its use, the active object must notify the mechanism so that the shared resource lock can be released. MessageSynchronization adds a waiting mechanism which pauses the execution of a behavior until the arrival of an acknowledge message. This waiting mechanism can be implemented either as a busy wait or using the scheduling system, i.e., mark the active object as blocked and invoke the scheduler. This decision is made only at aspect implementation, allowing an implementation independence of this aspect.

Communication Package provides aspects to deal with objects communication in terms of messages sending. Communication can happen between objects located on physically separated computing devices, or on the same device but implemented as hardware and software components. Both situations are covered within this package.

MessageAck adds a message guaranteed delivery mechanism. This aspect has two facets: (1) on the sender object, the
mechanism must be notified (after a message sending) that a message was sent and an acknowledgment message must arrive; (ii) on the receiver object, an acknowledgment message must be sent after the reception of a message. 

MessageIntegrity verifies the integrity of a received message. This aspect has also two facets: (i) on the sender object, before sending the message, an algorithm (e.g., parity, CRC, etc.) must generate checking information which is going to be appended to the message; (ii) on the receiver object, after receiving the message, a checker algorithm must generate checking information from the received message, and then, compare it with information received within the message. 

MessageCompression adds a compression mechanism to improve bandwidth usage. On sender object, the message is compressed before it is sent, while on receiver object the message is decompressed before delivering it.

**Embedded package** deals with the non-functional requirements related to the availability of physical resources, which are very common concerns in embedded systems design: 

*EnergyMonitoring* inserts an energy monitoring mechanism to measure energy consumption of an activity. Before the beginning of an activity, current energy level is measured. Thereafter, at the end of execution, energy level is measured once more and the difference is calculated and stored. 

*EnergyControl* adds a mechanism which implements an energy control policy. Control actions are performed depending on the remaining energy level, e.g., to eliminate unnecessary tasks, migrate active objects, loose temporal requirements, decrease system frequency, shutdown unnecessary hardware, among others.

*MemoryUsageMonitoring* inserts a mechanism to provide information on the amount of memory used by system objects. Before every memory allocation, the amount of requested memory must be accounted into the information on the total used memory. Similarly, after every memory release, the total of released memory must be subtracted from the total used memory. 

*MemoryUsageControl* performs memory control based on the selected policy, such as memory compression, migration of active objects, releasing of unused objects, among other control policies. 

*HwAreaMonitoring* provides a mechanism to monitor FPGA area usage. This aspect is useful when the system benefits from reconfiguration techniques. At each reconfiguration, the total area used in the FPGA must be accounted. If partial reconfiguration is allowed, the area utilization is updated in a similar way as memory. 

*HwAreaControl* verifies if requested hardware reconfiguration is possible and, if so, allows reconfiguration.

**TaskAllocation package** provides aspects to handle non-functional requirements related to objects distribution over different computing devices at runtime.

*TaskMigration* provides a mechanism to migrate active objects from one processing unit to another, or from software to hardware and vice versa. It is used by aspects that control embedded concerns (EnergyControl, MemoryUsageControl, and HwAreaControl), which are responsible for migration decision. 

*NodeStatusRetrieval* inserts a mechanism to retrieve information about processing load, message send/receive rate, and/or node aliveness (i.e., “I’m alive” message). Before every active object starts its execution, processing load is calculated; the same is done after its execution. Before every message is sent, the message rate is computed, as well as after the reception of each message. Additionally, the node availability message is sent periodically with an interval of “n” time units, or after “n” messages are sent/received. 

**Reconfiguration package** provides aspects to deal with the reconfiguration of active objects, including support for task running on CPU, GPU or FPGA. Additional details on this package are provided in [8].

*TaskReconfiguration* aspect provides the reconfiguration mechanism that allows tasks to be executed on CPU, GPU and/or FPGA. This aspect is similar to the TaskMigration aspect, but it deals with reconfiguration rather than migration of tasks.

*SystemProfiler* aspect deals with the analysis of several characteristics of system runtime, which cannot be known at design time, e.g., size and type of input data, data transfer between PUs, tasks’ performance, etc. It affects the system behavior by including a mechanism to measure and store the collected data on system runtime performance.

*Task AllocationSolver* is responsible for deciding whether a task needs to be migrated/reconfigured. It also selects to which processing unit this task should be moved. For that, *TaskAllocationSolver* checks the overload status of all destination processing units, in order to decide if it is worthwhile to perform such a migration or reconfiguration. Finally, every time the *TaskAllocationSolver* changes the tasks schedule, both TaskReconfiguration and TaskMigration are commanded to reconfigure or migrate a given task.

*TimingVerifier* aspect is responsible for checking if the processing units are using enough memory with timing requirements specified through TimingAttributes, PeriodicTiming, ToleratedDelay and TimeBoundedActivity aspects. Thus, it includes a mechanism check whether timing attributes are being respected. For that, the current time is measured in distinct moments, depending on the timing characteristic being controlled, e.g., the activation delay for a task is measured when the task becomes ready to execute and also right before its execution.

As previously stated, the goal of DERAF is to provide high-level aspects to be used at modeling phase of system design to specify handling of non-functional requirements. In the following phases, as implementation phase, these aspects must be realized through application code or platform services code. Therefore, to proceed with design process, the implementation of DERAF aspects must be provided, and thus binding each aspect with an implementation technology, like, for example, CUDA or OpenCL for GPU, C++, Java, or OpenCL for CPU, and VHDL for FPGA. The semantics of “how” and “where” each aspect affects system elements are defined in high-level and must be preserved. That is, every implementation must follow these semantics in order to allow a reuse of previously developed aspects implementation. The idea is to build a library of aspects implementations that could be easily reused in further projects and, hence, reducing design effort.

Sometimes, it is also interesting to evaluate the impact of aspects implementation into original embedded and real-time system at modeling phase. Therefore, it is necessary to provide models to describe how each aspect implementation affects the original specification. This way, it is necessary to provide an aspect model weaving, following concepts presented in [45]. In spite of its importance, a detailed discussion about both implementations (models and code), as well as model weaving, are out of the scope of this paper. Interested readers should refer to [53].

It is important to highlight that the current version of DERAF does not provide aspects to handle all non-functional requirements presented in the design of embedded and real-time systems. For
instance, fault-tolerance is an important non-functional requirement that still does not have support in its current version. However, as DERAF is intended to be an extensible framework, aspects to deal with new non-functional requirements (e.g. availability, safety, security, and others) may be incorporated. For that, the engineers must analyze the semantics behind the non-functional requirement handling. Once the required features are understood, one or more aspects should be created, including their adaptations, to handle the non-functional requirement. It is worth mentioning that, to enable the use of the new aspect(s) in UML models, each adaptation shall be defined in a platform independent fashion without constraining its semantics to any execution platform or technology.

As an example, Triple Modular Redundancy (TMR) technique [31] could become a DERAF aspect to cope with the fault-tolerance non-functional requirement. TMR aspect would have the following adaptations: (i) a structural adaptation to triplicate the affected elements; (ii) other structural adaptation to include a voter element into the system; and (iii) a behavioral and/or structural adaptation that replaces the actions/interconnections of the affected elements by the equivalent actions/interconnections enclosed by the voter element. Obviously, such adaptations must be supported by any target platform, otherwise TMR aspect mapping rules cannot generate the appropriate source code. Finally, once the new aspects semantics are fully defined, they can be used in the UML model as described in the next section.

7.2. Using DERAF in UML models

In order to represent the non-functional dimension, AMoDE-RT proposes the use of two additional diagrams: (i) Aspect Crosscutting Overview Diagram (ACOD), which shows DERAF aspects and, additionally, crosscutting information added by an aspect into one or more classes; and (ii) Join Point Designation Diagram (JPDD), presented in [45,46], which emphasizes the point in which a given functional element (e.g., class, attribute, message exchange, etc.) is affected by one or more aspect.

A fragment of the ACOD diagram for the IPS case study is shown in Fig. 10. Aspects are represented as classes with a \(<\text{aspect} \rangle\) stereotype. They encapsulate all the elements necessary to handle the non-functional dimension. Such elements are woven into the functional dimension, which is handled by objects/classes. Aspect adaptations specify how “functional concerns” are adapted (i.e., enhanced, replaced, or even deleted) when an aspect affects them. For such specification, an aspect defines two types of adaptations: (i) Structural adaptations represent modifications in the structure of a concern, e.g., adding a new attribute or method to a class; and (ii) Behavioral adaptations specify changes in the behavior of a concern, e.g., inserting a specific behavior before or after sending a message, or replacing an entire behavior by another (see FrequencyControl in Fig. 10).

Other important element in the aspect definition is a pointcut. The pointcuts bind the aspects adaptations with join points, i.e., they indicate places where the aspect must perform a given adaptation (see pcFREQCtrl in Fig.10). Usually, one pointcut relates one adaptation to one or more join points. A pointcut also specifies a relative position, indicating if the adaptation should be applied before, after or around a join point.

As mentioned, join point is one of the pointcut parameters, along with the respective adaptation that will take place in that specific join point. The join point is graphically represented as an adapted version of the JPDD presented in [46]. Thus, JPDD has been enhanced to allow specifications of join points within controls and data flows, as well as possible state changes.

The main source of information to specify join points is the field “context” within non-functional requirements templates. Fig. 11 illustrates three examples of join points. Fig. 11A defines that all classes of active objects (those annotated with \(<\text{ScheduledResource} \rangle\) are selected in JPDD_ActiveClass join point. Fig. 11B specifies that the JPDD_Active_ObjectConstructor join point selects all constructor methods from active object classes. On the other hand, Fig. 11C shows a join point that selects the periodic activation of active objects behavior. Its selection criteria are: (i) all messages \("(\ldots):^*\) means any name, with any number of arguments and any kind of return) shall be annotated with \(<\text{TimerEvent} \rangle\); (ii) all messages, which matched with previous criterion, shall be sent from any Scheduler object (those annotated with \(<\text{Scheduler} \rangle\) to any active object; (iii) the select elements are behavior triggers (i.e., behavior invocation), as indicated by the \(<\text{JoinPoint} \rangle\) stereotype.

This join point (JPDD_PeriodicActivation) is bound to both Loop-Mechanism and FrequencyControl adaptations through, respectively, pCLoop and pcFREQCtrl pointcuts (see Fig. 10). Selection criteria are evaluated automatically during model transformation (step #6 in Fig. 1) and, as result, the selected elements are gathered and linked to the respective join point. Engineers do not need to indicate manually which elements should be selected in each join point; the selection is performed automatically (and it is based on the specified selection criteria) when the UML model is transformed into the intermediate platform-independent model.

The ACOD depicts crosscutting relations. These relations are made between classes and aspects as the ones decorated with a \(<\text{crosscut} \rangle\) stereotype. In fact, the semantic of these relations is not similar to the “regular” associations of classes. The crosscut relationship does not mean that an aspect is bound to any class. It only indicates the initial value for an attribute that is going to be inserted into that class.

Finally, it is worth mentioning that the notation used to represent such aspects, their internal elements, and the association between aspects and classes follow patterns compatible to those used in the aspect research community. Moreover, the adopted notation makes it easier to map concepts from design to implementation in, for example, [Java/AspectJ] language, such as “StructuralAdaptation” in “introduction” and “BehavioralAdaptation” in “advice”, contributing with project traceability [46].

8. Tool support for AMoDE-RT

According to Selic [43], an important issue to allow the use of MDE is tool support. Automatic transformation from a PIM to a PSM is a key issue to make models the main artifact during design 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 from meta-model to text representing source code in a target language. This section discusses the tools created to support AMoDE-RT in both model (informal) verification (see Fig. 1, step #8) and code generation (see Fig. 1, steps #9 – #11).

8.1. AT4U: test cases execution tool

The Automated Testing for UML (AT4U) tool aims at verifying the behavior of embedded and real-time systems in earlier design stages. The proposed approach is based on concepts presented in the family of code-driven testing frameworks, known as xUnit.

AT4U focus on finding specification errors through the direct execution of system behaviors by using test cases. For that, AT4U executes a set of test cases upon both individual elements (i.e., unit test) and groups of dependent elements (i.e., component test) that have been specified in the UML model. Further, as the execution of test cases is performed automatically by a software tool, a testing technology is not desirable since engineers need to translate high-level specification into code for the target execution platform before performing any kind of automated testing. In this situation, testing results may be affected by both errors of this translation process and errors of this translation process.

One goal of AT4U approach is to verify the high-level specifications of embedded and real-time systems in order to decrease costs associated with fixing specification problems at implementation phase. Usually, high-level specifications – such as UML models – are independent of any implementation technology or execution platform. The use of any platform specific testing generates a XML file containing individual results of each test case, i.e., success or failure to achieve expected outcomes as well as produced results and associated execution scenarios.

Fig. 10. Fragment of ACOD showing TimingAttributes and PeriodicTiming aspects.

AT4U approach is depicted in Fig. 12. As it can be seen, DERCS model and a set of test cases are the inputs to AT4U. Test cases are specified in a XML file. This tool reads this file and instantiate a test cases model by using AT4U meta-model. After creating this model, the tool executes automatically each test case on the UML model by using Framework for UML Model Behavior Simulation (FUMBeS) [55]. Once the set of test case is executed, the tool generates a XML file containing individual results of each test case, i.e., success or failure to achieve expected outcomes as well as produced results and associated execution scenarios.

Please cite this article in press as: Wehrmeister MA et al. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. Mechatronics (2014), http://dx.doi.org/10.1016/j.mechatronics.2013.12.008
In this sense, to allow automated testing for platform independent specifications, it is important to provide: (i) a platform independent description of test cases, and (ii) a mechanism to execute these platform independent test cases. AT4U proposes a test suite model, whose meta-model is based on concepts of the xUnit framework [5] and the UML Testing profile. This model represents the information on: (i) the test cases used to exercise system behavior; and (ii) test case results, which are produced during the execution of each test case. The Eclipse Modeling Framework (EMF) has been used to specify and implement the proposed model.

Thus, AT4U automates the execution of test cases on high-level specifications. Once the DERCS and test suite model are loaded, test cases are executed on the model. For each test case, three phases are performed: (i) initial scenario setup; (ii) execution of the methods set; and (iii) evaluation of the resulting scenario after executing the set of methods.

In the scenario initialization phase, the information provided within the input scenario of each test case is used to initialize the runtime state of DERCS objects. Each object described in the input scenario provides values to initialize DERCS objects, i.e., these values are directly assigned to runtime information of the corresponding attributes.

The second phase is to execute methods specified within the test case, checking if associated assertions are valid or not. The method testing phase is divided in two parts: (i) method setup and execution; and (ii) evaluation of assertion associated with the method under test. In the former one, values of input arguments (specified within the test case) are used to initialize the method parameters in DERCS model. Once all input arguments are set in the DERCS model, the FUMBeS framework simulates the execution of the behavior associated with the method under test. FUMBeS executes the set of actions that represent behavior specified in UML model (for details see [55]). After finishing the simulation, FUMBeS returns the value produced during the execution of the method under test. In the second part, assertions associated with the method under test are evaluated. Assertions can be made based on both expected results and/or scenarios. AT4U compares expected result with the one obtained after the method execution, using specified comparison operation.

The third phase is the evaluation of assertion related to an expected scenario of the whole test case. For that, AT4U compares the expected scenario described in the test case with the scenario obtained after executing the set of methods. Each object of the expected scenario is compared with its counterpart in DERCS model. If the state of both objects is similar, the next object is evaluated until all objects described in the expected scenario are evaluated.

Finally, the test case is marked as successful if all assertions are valid, i.e., those related to each individual method must be valid, along with those related to the whole test case. If any of these assertions is not valid, the test case failed and is marked accordingly. This results are represented using the AT4U meta-model.

8.2. GenERTiCA: code generation tool

Generation of Embedded Real-Time Code based on Aspects (GenERTiCA) is a code generation tool created to support AMoDE-RT. It uses information specified in UML models to generate code for different target platforms. For example, system structure, behavior, and handling of non-functional requirements using DERAf aspects that are transformed into elements available in the chosen platform.

In order to generate code as completely as possible (i.e., not only classes skeletons as most of the tools do), the following steps have to be carried out. First, it is important to define a set of modeling guidelines to be strictly followed by engineers. In addition, the amount of information from the input UML model must be well-defined by either (i) using a subset of the UML meta-model or (ii) to create a new PIM whose meta-model is simpler than the UML one. Considering the later case, a transformation from UML to the new PIM must be also defined. GenERTiCA approach uses (ii) instead of (i) due to the lack of concerns separation of the handling of functional and non-functional requirements using AOSD concepts in the current version of UML (2.4.1). Lastly, one shall define the mapping of meta-model elements into constructs of the target language, APIs, and/or components available in the chosen execution platform. Due to space constraints, the created PIM (i.e., DERCS model) as well as the UML-to-DERCS transformation heuristics are not discussed in this text. Interested readers should refer to [53].

Furthermore, GenERTiCA is a script based code generation tool. It takes as input UML model and set of mapping rules (described through a XML file) to generate code (see Fig. 13). Mapping rules are described in XML files, whose self-described format facilitates the reuse of previously developed scripts. By using XML engineers can create repositories to store implementation of DERAf aspects and mapping rules that were previously created and validated.

As mentioned, the goal of GenERTiCA is to generate code as completely as possible without constraining the tool to a specific target language. In order to keep the description of mapping rules as simple as possible, each script aims at generation of a code fragment representing one element of the model. In the XML file structure, leaves contain the mapping rules scripts, which perform the translation of model elements into source code; and also weave aspects adaptations for a target language. Moreover, the tool was intended to be flexible enough in order to be able to generate code for software and hardware (using HDL) components of an embedded and real-time system.

Simplicity was one of the goals for the description on how to map model elements into source code. For this reason, the Velocity Template Language (VTL) 11 was chosen as script language. VTL open source scripting engine allows VTL scripts to have access to GenERTiCA features such as the input PIM (i.e., DERCS instance of the system) and the code generation engine itself. GenERTiCA adopts an approach to create small scripts, which are executed in order to generate fragments of source code for each single element of DERCS, instead of generating the whole source code. In other words, to keep the aim and simplicity of each mapping rule, scripts in the XML tree leaves need only to access information from few elements (or just a single one).

GenERTiCA traverses the list of DERCS elements (e.g., classes, attributes, methods, behaviors, etc.) looking for a script (according the tree hierarchy) that matches with the selected element. Once the script is found, it is executed and a text fragment that represents the source code for the selected element is generated. In addition, GenERTiCA verifies if any DERAf aspect affects the

---

9 http://www.omg.org/spec/UTP/1.1/.
10 http://www.eclipse.org/modeling/.

Please cite this article in press as: Wehrmeister MA et al. Combining aspects and object-orientation in model-driven engineering for distributed industrial mechatronics systems. Mechatronics (2014), http://dx.doi.org/10.1016/j.mechatronics.2013.12.008
selected element, i.e., it verifies whether this element is contained in the selection list of any join point. If this is the case, scripts of all adaptations associated with the join point are executed and the recently generated code fragment is modified, i.e., the aspect weaving is performed. This algorithm repeats until all elements are evaluated.

It is important to highlight that, due to VTL characteristics, scripts allow the generation of fragments of source code for different languages such as C/C++, Java, OpenCL, CUDA, VHDL, SystemC, among others. These generated source code fragments are grouped, thus creating source code files that can be compiled or synthesized by external tools. As the scripts have full access to information about system objects using the meta-model of DERCS, it is possible to construct specialized and also complex mapping rules.

9. Results and assessment

This section presents an assessment conducted on using AMoDE-RT to design the IPS case study. This assessment comprises: (i) a comparison of two designs: one using OO concepts available in the "regular" UML versus the one using the AOSD-based approach proposed in AMoDE-RT; (ii) quantification of the source code generated for two distinct target platforms based on the same UML model; (iii) initial results on the simulation of IPS behavior specified in the UML model. An initial part of this assessment is to evaluate reusability quality of artifacts created using AMoDE-RT approach. Interested readers can find other examples of using AMoDE-RT in [56], which discusses the results obtained in applying AMoDE-RT approach in the design of three distinct automation and control systems, including IPS.

9.1. Object- versus aspect-oriented designs

The assessment of aspect- and object-oriented models of IPS was performed using a set of metrics for AOSD [40] and OO [15] development. However, a set of metrics is not enough to determine any system quality. It is also required knowing how those metrics are related to each other, in order to provide meaningful information about design quality. This work uses the assessment framework presented in [40] to infer the quality of such presented models by measuring its reusability. To provide a qualitative assessment of both models, a subset of metrics was chosen based on their suitability for modeling instead of coding phase. Additionally, it is important to highlight that this paper concentrates only on "reusability" instead of "reusability and maintainability" as proposed in the original assessment framework.

Initially, in order to provide a visual comparison between the object- (OO) and aspect-oriented (AO) versions of the IPS case study, Fig. 14 depicts the FR-7 specification, i.e. the item detection controller task. The AO version of this requirements specification has already been depicted in Fig. 8. As it can be noted, OO version of this diagram depicts 7 elements; it presents 3 additional

![Fig. 14. OO version of the item detection controller task.](image-url)
elements in comparison with the AO version. Besides, elements related to non-functional requirements (those with color filling) are specified intermixed with elements associated with functional requirements (those without color filling). Such a situation hinders understandability and reuse of the OO version of this diagram, since it is more complex (i.e. it presents a greater amount of lifelines, messages, and combined fragments) in comparison with the AO version.

Furthermore, the separation of concerns metrics depicted in Fig. 15A shows an improvement in the aspect-oriented version of IPS. It is also required knowing how those metrics are related to each other in order to provide meaningful information about design quality version for the industrial packing system. Concern Diffusion over Components (CDC) metrics have been reduced between 66% and 81%, while Concern Diffusion over Operations (CDO) between 16% and 75%. The other metrics exhibit similar improvements (see Fig. 15B): (i) Depth of Inheritance Tree (DIT) did not change, i.e., aspects do not modify classes hierarchy; (ii) aspect-oriented model is more cohesive, as indicated in the decrease of 47% in Coupling Between Components (CBC) metric; (iii) Vocabulary Size (VS) in aspect-oriented version indicates a small increase (i.e., one element) due to, in this version, memory requirements are handled by two elements (MemoryUsageControl and MemoryUsageMonitoring aspects) instead of one element as specified in OO version; (iv) Number Of Attributes (NOA), on the other hand, decrease almost 46% in aspect-oriented version, showing that in spite of the increase in VS metric, the number of classes internal elements has decreased.

Regarding model cohesion in aspect-oriented version, Lack of cohesion in Operations (LCOO) drastically decreases 58% when considering all kinds of methods, and 16% if “raw” get/set methods are excluded. This shows that, in spite of good cohesion in the OO version, the use of AOSD concepts definitely improves system cohesion.

For IPS, the reuse of previously created artifacts is highlighted in both AOSD-related elements specification and mapping rules. Considering the former, in addition to DERAF aspects reuse, JPDDs also have been reused. From the 13 used JPDDs in this case study, 11 have been reused from the other case study without any modification.

9.2 Code generation

The second part of this assessment focuses on the amount of code generated by GenERTiCA. Table 1 shows the amount of elements specified in the UML model. For functional requirements, 14 “regular” UML diagrams have been created, whereas for non-functional requirements, one ACOD (describing 9 DERAF aspects) and 13 JPDDs have been specified.

Considering generated code, two different target platforms have been selected for this specific IPS: RT-FemtoJava [54], a real-time Java platform comprising an API based on a subset of the Real-Time Specification for Java (RTSJ) [9] and a Java microcontroller; and ORCOS [49], a PowerPC compatible Real-Time Operating System written in C++. Table 2 shows the code generation statistics of IPS. For each script line for the RT-FemtoJava platform, GenERTiCA generated 2.95 lines of Java code; whereas for the ORCOS platform, 4.04 lines of C++ code were generated per script line. In fact, it is important to highlight that this mapping rules files have been created in other case study and reused without changes in this IPS case study. Moreover, 52.00 (Java) and 61.05 (C++) lines of code have been generated (in average) per class.

Considering the use of GenERTiCA, it must be stated that the amount of generated code is directly proportional to the completeness of mapping rules scripts and diagrams specification. In other words, if the UML model can provide complete information about system structure and behavior (following AMoDE-RT modeling guidelines), and also mapping rules can map all elements available in the model into constructions available in a given target platform, it is likely that GenERTiCA can generate a large amount of correct source code.

Regarding the generated source code, source code files obtained after the code generation process are more complete than the ones obtained using available commercial or academic code generation tools, which usually only provide class skeletons and/or simple state machine related code. In addition, aspects weaving performed by GenERTiCA allows the use of aspect adaptations in non-aspect-oriented languages. In this case study, neither RT-Femtojava nor ORCOS supports AOSD concepts. Even though this issue, aspects were specified within the UML model; GenERTiCA identified aspects adaptations and generated corresponding code using APIs available in these platforms.

9.3 Early verification of the UML model: behavior simulation

As mentioned, AMoDE-RT allows the verification and validation of system specification by simulating the UML model during the specification process. In other words, once engineers complete the specification of some part of the system behavior (e.g. the behavior responsible for detecting if some part arrived at the end of a conveyor belt), he/she is able to simulate such a behavior, in order to check whether it behaves as expected. Therefore, to make the simulation practicable, the performance of the simulation tool must be acceptable, i.e., the simulation should perform behavior simulation as fast as possible.

For that, a small simulation prototype has been created using FUMBeS, which is the main component of AT4U tool (see Section 8.1). Its main functionality is to simulate the behavior of one method described in a DERCS model. FUMBeS has been developed in Java (JDK 1.6), as well as the simulation prototype. The primary goal of this prototype is to compare simulation performance against native implementation (i.e., direct Java implementation) of three methods of the IPS: (i) AssemblyCellController.run ( ), which controls the assembly of products that are composed by three distinct parts; (ii) Conveyor.run ( ), which controls the movement of conveyors according to the need of the AssemblyCellController object; (iii) StorageUnit.run ( ), which samples and controls the amount of parts stored in its compartments.

Thus, this performance assessment shows how fast the simulation of a behavior specified in an UML model can be executed. However, it is very important to highlight that FUMBeS is not intended to substitute the native implementation; it is a complementary tool to assist engineers in the verification and validation process of system specification. The experiments have been conducted on a MacBook with one Intel Core 2 Duo processor running at 2.16 MHz, 2 GB of RAM, Snow Leopard as operating system, and Java SE Runtime Environment version 1.6.0_26 (build 1.6.0_26-b03-384-10M3425). To execute the experiment, the system was reboot, all user-level applications closed, and then experiment executed in a bash session.

Table 3 presents results obtained in these experiments. The numbers represent an average time (in milliseconds) required to execute each method. Each method has been executed multiple times and the duration of each execution was used to calculate an average duration of the simulation. For the native implementations, each algorithm was executed 10,000 times, whereas for the simulation the number of executions was 1000 times per algorithm.

As it can be seen, the simulation duration for these IPS methods is very short. Although the native implementation performance is much faster, the duration of simulation is almost imperceptible to a human being. The average duration of simulation is
It should be noted that, as mentioned, FUMBeS is not intended to replace the native implementation for the final system. AT4U tool uses FUMBeS to simulate the behaviors that are exercised during the execution of the test cases set. Considering that, during the execution of the test cases set, AT4U allows for simulation capabilities during the execution of the test cases set, such an approach may improve the process of errors discovering, such an approach may improve the process of errors discovering, since the parts of the UML model can be tested in a controlled scenario. Furthermore, it is important to highlight that, at the specification phase, no effort/time was spend to obtain any native implementation intended for specification checking. Hence, the cost to verify the system is intuitively lower than the one for testing the system only at implementation phase.

Finally, it is important to mention that results regarding the use of AT4U for the IPS case study are still not available. The set of test cases is under development and, once they are available, the performance of AT4U is going to be evaluated on the context of the IPS case study. Preliminary results of using AT4U for testing other control system have been presented in [52], achieving encouraging results in terms of performance. For instance, for that case study, the average duration of one test case, which included the behavioral simulation of one or more methods, was 185.6 ms (25.96 ms per method on average). This performance is considered good for an automated early verification technique, since it allows the execution of all test cases in a quicker way. Therefore, the exhaustive repetition of test cases execution may be achieved, e.g. at every round of changes in the specification. Moreover, although we acknowledge that the execution time is very dependent on the individual behavior, we strongly believe that this results will be similar in the IPS case study.

Table 1
Statistics on the UML model of aspect-oriented version.

<table>
<thead>
<tr>
<th>Diagrams</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Behavioral</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>ACOD</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>JPDD</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>DERAF aspects</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>Structural adaptations</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>Behavioral adaptations</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

- Numbers within parentheses represent the number of bytecodes generated by the Java compiler; the first number represents the final binary size generated by the RT-FemtoJava synthesis tool [54].
- Numbers represent the amount of DERCS elements.
- Considering all elements that have individual scripts in the mapping rules. They are: classes, attributes, methods, actions, branches and loops, aspects, and their structural and behavioral adaptations.
- Considering the real-time java API [54] plus communication API. The first number represents generated files (required by the application), whereas the second number indicates total of files available in both APIs.

Table 2
Statistics on generated source code for the IPS case study.

<table>
<thead>
<tr>
<th></th>
<th>RT-FemtoJava</th>
<th>ORCOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DERCS*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Classes</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>Classes + Aspects</td>
<td>31</td>
<td>31</td>
</tr>
<tr>
<td>Total*</td>
<td>345</td>
<td>345</td>
</tr>
<tr>
<td>Script Lines*</td>
<td>388/803 lines</td>
<td>332/749 lines</td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source code files</td>
<td>22</td>
<td>42</td>
</tr>
<tr>
<td>Lines of code</td>
<td>1144</td>
<td>1343</td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>4.64 (29)</td>
<td>139</td>
</tr>
<tr>
<td>Stats per Study</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LoC/Class</td>
<td>52.00</td>
<td>61.05</td>
</tr>
<tr>
<td>LoC/C + A</td>
<td>36.90</td>
<td>43.32</td>
</tr>
<tr>
<td>LoC/Total</td>
<td>3.32</td>
<td>3.89</td>
</tr>
<tr>
<td>Platform</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source code files</td>
<td>21/38*</td>
<td>01/01</td>
</tr>
<tr>
<td>Lines of code</td>
<td>2931*</td>
<td>480</td>
</tr>
<tr>
<td>Binary size (KB)</td>
<td>6.12 (50)</td>
<td>462</td>
</tr>
</tbody>
</table>

- Numbers within parentheses represent the number of bytecodes generated by the Java compiler; the first number represents the final binary size generated by the RT-FemtoJava synthesis tool [54].
- Numbers represent the amount of DERCS elements.
- Considering all elements that have individual scripts in the mapping rules. They are: classes, attributes, methods, actions, branches and loops, aspects, and their structural and behavioral adaptations.
- Considering the real-time java API [54] plus communication API. The first number represents generated files (required by the application), whereas the second number indicates total of files available in both APIs.

Table 3
Duration of behavior simulation.

<table>
<thead>
<tr>
<th>Method</th>
<th>Native (ms)</th>
<th>FUMBeS (ms)</th>
<th>FUMBeS(^a) (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPS-ACC-Control</td>
<td>0.00420</td>
<td>7.294</td>
<td>5.206</td>
</tr>
<tr>
<td>IPS-CV-Control</td>
<td>0.00020</td>
<td>1.435</td>
<td>1.238</td>
</tr>
<tr>
<td>IPS-Std-Control</td>
<td>0.00020</td>
<td>1.007</td>
<td>0.735</td>
</tr>
</tbody>
</table>

- Generation of XML output file is disabled.

Fig. 15. Calculated metrics for the industrial packing system.

AMoDE-RT is a platform-based MDE approach, and hence, it relies on services provided by hardware/software components already developed. Therefore, the reuse of existing design artifacts (i.e. model elements, mapping rules, and hardware/software components) is part of the proposed approach, since, nowadays, it is improbable to design a complete industrial mechatronics system from scratch. AMoDE-RT reinforces the reuse as follows: (i) the created UML model specify system structure and behavior in a specification phase, (i) the implementation is unavailable, and (ii) the specification is not complete to allow the generation of usable code, a feasible way to check if the system behavior respects the functional requirements is simulation. As FUMBeS provides a quick way to execute and observe system behavior, and AT4U uses its simulation capabilities during the execution of the test cases set, such an approach may improve the process of errors discovering, since the parts of the UML model can be tested in a controlled scenario. Furthermore, it is important to highlight that, at the specification phase, no effort/time was spend to obtain any native implementation intended for specification checking. Hence, the cost to verify the system is intuitively lower than the one for testing the system only at implementation phase.

9.4. Discussion

AMoDE-RT is a platform-based MDE approach, and hence, it relies on services provided by hardware/software components already developed. Therefore, the reuse of existing design artifacts (i.e. model elements, mapping rules, and hardware/software components) is part of the proposed approach, since, nowadays, it is improbable to design a complete industrial mechatronics system from scratch. AMoDE-RT reinforces the reuse as follows: (i) the created UML model specify system structure and behavior in a specification phase, (i) the implementation is unavailable, and (ii) the specification is not complete to allow the generation of usable code, a feasible way to check if the system behavior respects the functional requirements is simulation. As FUMBeS provides a quick way to execute and observe system behavior, and AT4U uses its simulation capabilities during the execution of the test cases set, such an approach may improve the process of errors discovering, since the parts of the UML model can be tested in a controlled scenario. Furthermore, it is important to highlight that, at the specification phase, no effort/time was spend to obtain any native implementation intended for specification checking. Hence, the cost to verify the system is intuitively lower than the one for testing the system only at implementation phase.

Finally, it is important to mention that results regarding the use of AT4U for the IPS case study are still not available. The set of test cases is under development and, once they are available, the performance of AT4U is going to be evaluated on the context of the IPS case study. Preliminary results of using AT4U for testing other control system have been presented in [52], achieving encouraging results in terms of performance. For instance, for that case study, the average duration of one test case, which included the behavioral simulation of one or more methods, was 185.6 ms (25.96 ms per method on average). This performance is considered good for an automated early verification technique, since it allows the execution of all test cases in a quicker way. Therefore, the exhaustive repetition of test cases execution may be achieved, e.g. at every round of changes in the specification. Moreover, although we acknowledge that the execution time is very dependent on the individual behavior, we strongly believe that this results will be similar in the IPS case study.
platform independent fashion; (ii) DERAF aspects encapsulate the non-functional requirements handling, which is not mixed with functional requirements specifications; (iii) automatic model transformations, e.g. model-to-model transformations (UML-to-DERCS), and model-to-text transformations (i.e. code generation); and (iv) mapping rules to define the mapping between model elements and services/components provided in the system target platform.

In this sense, it can be stated that AMoDE-RT facilitates the reuse of both UML modeled elements and mapping rules for distinct target platforms. The reuse of UML modeled elements is enabled by (i), (ii) and (iii). Specifically, DERAF aspects have been created to be platform independent aspects, i.e. they provide high-level semantics on how to deal with one or more non-functional requirements without binding them to any implementation technology. For instance, if an industrial mechatronics system demands periodic tasks in its design, there are elements commonly found on any implementation, e.g. an element to control the execution frequency of the periodic tasks. Evidently, each of these elements may be implemented differently depending on the services offered by the target platform. As PeriodicTiming aspect provides such common elements in platform independent way, and hence, it can be reused in the model (without modifications) to specify periodicity non-functional requirement handling. PeriodicTiming and all other aspects have been reused in IPS and other case studies [56].

However, the reuse of aspects implies that new pointcuts must be defined for each different system design. As pointcuts define the link between aspect adaptations and the system elements affected by these adaptations (i.e. the join points), they are system-specific and require modifications on the aspects specification in ACOOS diagram. Nevertheless, AMoDE-RT uses JPDDs to specify join points, which are application and platform independent. For instance, all join points depicted in Fig. 11 specify selection criteria that are application independent, since they select UML elements related to active objects, which may appear in distinct industrial mechatronics systems. Consequently, in IPS case study, 11 out of 13 JPDD could be reused without modifications from other case studies.

Furthermore, UML diagrams related to functional requirements might also be reused. By using AOSD concepts, DERAF aspects, and JPDDs, AMoDE-RT approach separates the handling of functional and non-functional requirements in the UML diagrams specification, facilitating the reuse of such diagrams. For instance, the item detector sequence diagram (see Fig. 8) could be reused “as is” in other designs, since it does not specify any non-functional requirements handling or elements from any execution platform/technology. However, it is important to highlight that such a reuse is possible when similar functional requirements are present in the new system. In other words, it is probable that systems of the same application domain present similar characteristics and requirements, and hence, UML diagrams could be reused. Considering the arguments presented above and the evidences advocating for reusability of different modeling artifacts, it is possible to state that engineers could reuse modeling elements of the IPS case study in designing other industrial mechatronics systems, whether such systems present characteristics similar to IPS.

On the other hand, reuse of implementation artifacts is also obtained in AMoDE-RT. As mentioned, GenERTICA uses mapping rules to define how the model elements are implemented using services/components provided in distinct target platforms. If the mapping rules for the system target platform have been already created, they may be reused to generate the implementation of the specified industrial mechatronics systems. There is an effort to define such mapping rules in the first project that uses the target platform/technology, but, for new projects using the same platform, such an effort is drastically reduced. In IPS case study, we used the set of mapping rules already created in other case studies, and hence, the effort to generate the system implementation was lower.

As mentioned in Sections 8.2 and 9.2, mapping rules are specified as small VTL scripts stored in a XML file. Such an approach facilitates the reuse and adaptation of individual scripts, since they are isolated as distinct element in the XML tree organization. For instance, if a new service/components is added into an already mapped target platform, only the relevant parts of the mapping rules must be modified, and (possibly) a great part of the mapping rules may be reused without modifications. In addition, the mapping rules include the implementation of DERAF aspect adaptations using the services provided in the target platform. Thus, DERAF aspects implementations can be reused without modifications or with a minimum amount of them, considering the case in which there is a need for including a new service/component provided in the target platform. For instance, in the mapping rules for the RT-FemtoJava platform, FrequencyControl adaptation of PeriodicTiming aspect is implemented using a call for the method RealtimeThread.waitForNextPeriod() which is defined in the RTSJ API. If the platform does not provide such a service, the implementation of this adaptation should use a timer (or other service) to control the waiting period. Therefore, for this change in implementation, only this adaptation implementation needs to be modified, demanding lower effort in comparison to re-implement the whole PeriodicTiming aspect.

Despite the likelihood of both reusing design artifact and decrease in design effort, MDE approaches are still not spread in the industry. Recently, real-world industrial case studies have showed and proved useful the use of MDE during industrial systems development and deployment, like [25,33,29]. Applying a personalized approach in specific domains seems much more effective than using a generic approach for a several number of distinct industrial areas. However, this comes with the cost of developing metamodels and tools for the domain, customizing generic tools and integrating such tools into a tool chain [34]. Thus, such a scenario is only affordable for enterprises which could make use of these models in several (and complex) projects (and not for single and small projects), reducing also risk of failures during project development.

In this sense, teaching and training needs to be improved by a change in its paradigm, in order to develop and use MDE in an effective way. Engineers would need a first contact with MDE approach in early stages, during college, to possibly propose and use such a technique in industry. Such a modification would help changing the conservatism on adopting MDE, as the engineers would be trained to show MDE benefits. A list of these benefits listed by professionals in industry, including financial one, can be found in [26]. Taking the last point, our approach clearly shows this benefit throughout the assessment and metrics presented in this section.

In general, benefits of MDE include automation of parts of the engineering work, automatic production of software from high-level models, cut of costs, shortening time-to-market, a good way to document and structure involved systems, among others. Of course, there remain issues on finding a good tradeoff between high-level models and obtaining good code and performance. Trained Engineers would have the basis to evaluate current execution platforms (that are each day more and more heterogeneous) to plan such customized models to achieve acceptable performance over new processing units. For example, good results have been obtained towards the modeling of non-functional requirements, such as performance, not only as presented in this paper but also as reported in other related works [44,8].

In addition, considering that industrial mechatronics systems have mechanical components which are usually integrated with
electronic and software components, as well as they interact with the physical environment, other relevant issue associated with MDE is the semantic gap between models. Since a model is a representation of the whole system (or part of it) in a given viewpoint, the model presents specific semantics and/or a given model of computation, e.g. continuous or discrete, synchronous or asynchronous, based on events or interactions, dataflow or control-flow driven, among others. The semantic gaps between models need to be fulfilled, in order to provide a consistent and integrated MDE approach. In isolation, mechanical, electrical, and computing engineering communities have been doing a lot of interesting and valuable research. Recently, the term “Cyber-Physical Systems” (CPS) [30,39] has been coined as a research field in which the problem of dealing with the integration of computation and physical processes are addressed. Some works can be found in the literature covering such a field, e.g. [19], however, there still many open issues on filling the mentioned semantic gap. Thus, such a situation also hinders a wider adoption of MDE in practice in the industry.

10. Conclusions and future work

This work addresses problems associated with the increasing complexity and handling of non-functional requirements of embedded and real-time systems found in the domain of industrial mechatronics systems. We proposed the use of AMoDE-RT in order to deal with such issues. It increases the abstraction level carried out during design by using principles of MDE along with concepts of AOSD.

AMoDE-RT uses a wide accepted and standardized modeling language such as UML and the MARTE profile. In addition, DERAf aspects are used to handle crosscutting non-functional requirements commonly found in the domain of embedded and real-time systems. Furthermore, AMoDE-RT is supported by two automated tools: (i) AT4U, which executes a set of test cases on the behavior modeled in UML; and (ii) GenERTICA, which generates source code from the UML model for distinct target platforms. This paper presented AMoDE-RT through a case study using a real-world industrial mechatronics system: the control system of a products assembler industrial cell, composed of a robotic arm, two conveyors belts and a storage unit. Considering the requirements engineering, RT-FRIDA provides suitable tools that have assisted the engineers in identifying and specifying IPS functional and non-functional requirements. During the specification phase, in the aspect-oriented version of IPS case study, the created UML model consists in two diagrams to specify the system structure and twelve diagrams to specify the system behavior. To specify the IPS non-functional requirements handling, one ACoD diagram have been created, including nine aspects reused from DERAf. In addition, although eleven JPDD have been reused from other case studies, two additional JPDD have been created to specify all the required join points. The part of the specified IPS behavior have been simulated, allowing the engineers analyze the simulation traces, in order to verify if the selected behaviors behaved as expected. Source code for two distinct platforms, namely RT-FemtoJava and ORCOS, have been successfully generated from the UML model.

An assessment of using high-level concepts of aspect- and object-oriented paradigms has been presented. The IPS case study have been designed using the two mentioned approaches, i.e. one created a “pure” object-oriented specification of IPS requirements, whereas the other one created the aspect-oriented specification. Both specification (i.e. UML model) have been evaluated in order to assess the reusability quality of each design. Embedded and real-time systems have specific non-functional requirements that must be properly handled to manage the increasing design complexity. Based on the presented results, AOSD can clearly help in such a quest. In the aspect-oriented model, due to the use of DERAf, encapsulated handling of non-functional requirements into aspects avoids the spread treatment of these requirements. It could be observed that aspects can impact positively in embedded systems design. Several metrics have a substantial decrease in aspect-oriented model of the case study, ranging from 17% up to 81%. A design is better understood if it has its functional and non-functional concerns well separated. In this case, the aspect-oriented version is much easier to understand than object-oriented one. Moreover, the elements of a design can be reused in other designs with less effort if they are cohesive and decoupled. It is expected that previously developed components can easily be reused in order to decrease the effort and shorten the time required to design a mechatronics system. Results show that aspects can help to achieve this goal, decreasing coupling and increasing cohesion.

Additionally, AMoDE-RT is supported by CASE tools. GenERTICA is a code generation tool capable of dealing with aspects during code generation process. The adopted approach uses small mapping rules scripts to produce code fragments that are merged to create source code files. Besides code generation, GenERTICA also performs aspects weaving at source code levels. This code generation approach supports the use of AOSD concepts together with non-aspect-oriented languages (e.g., C/C++, OpenCL, SystemsC, VHDL), since AOSD concepts are specified in the model and aspects are implemented as mapping rules, which use constructs/services provided in the target language/platform.

Furthermore, the AT4U tool automates the execution of test cases on UML models by using the FUMBes framework, which allows the simulation of (parts of) system behavior. In other words, AT4U executes (the parts of) the embedded and real-time system under controlled and specific situations that have been defined in test cases. Test cases are represented in a platform independent way by means of a test suite model. AT4U tool reports the results of test cases execution in a XML file. This report provides information such as the initial, expected, and resulting scenarios used in such test cases, as well as data produced by the executed behaviors and results of the assertions evaluation. The experiments provide indications that AT4U approach is suitable for the purpose of an early assessment of system behavior. Engineers may verify system specification and also evaluate different solutions, while specification is being created in early stages of design.

Regarding on-going and future work, GenERTICA is being extended to support other target platforms; for instance, VHDL mapping rules are being developed. DERAf needs to be extended to include other non-functional requirements which are currently not supported like, for example, security and fault-tolerance. Another research direction points to a generic framework for expressing aspects in DERAf. By using AOSD-related elements already provided in the DERCS model, it would be possible to define operations for model-level aspects (and the algebra behind it), enabling, for instance, aspect weaving in the DERCS model. Hence, the impact of aspects could be individually evaluated earlier in the design process. In this sense, applying DERAf and AOSD concepts in design space exploration will be a future work.

Finally, the combination of AMoDE-RT, DERAf, AT4U, and GenERTICA is what at first sight might seem like a counter-intuitive idea: explore the design phase by following a set of steps to produce detailed and correct code that executes with good performance instead of coding directly and deeply (which is the current approach by engineers). However, we have shown that this combination is highly reusable, cohesive, and generates excellent code quality. Results presented here clearly encourage a change on actual paradigm used by engineers as we evolve towards complex, parallel, and distributed execution platforms with different
co-processors, allowing even hybrid implementations. The combination of the developed tools extrapolates the current state-of-the-art research and delivers good results for real-world engineering applications.

Acknowledgement

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

Appendix A. Glossary

ACOD: Aspect Crossing Overview Diagram is an extension of the UML class diagram that depicts aspects and their crossing information. The latter represent the information that are inserted by the aspects into system elements, which are represented as “regular” classes. ACOD allows the specification of pointcuts, which are the elements that describe the binding between join points and aspects adaptations.

AMoDE-RT: Aspect-oriented Model-Driven Engineering for Real-Time systems is a UML-based MDE approach that proposes the separation of concerns on handling functional and non-functional requirements by using concepts of AOSE. This approach covers the early stages of design up to the implementation of embedded real-time systems used as part of industrial mechatronics systems. AMoDE-RT process is supported by several tool: checklist, templates and mapping tables for requirements engineering; third-party UML modeling tool for system specification; AT4U tool for specification verification; GenERTICA for code generation; and third-party tools to generate/synthesize the system hardware and software components.

AOSD: Aspect-Oriented Software Development is a design approach based on the Aspect-Oriented (AO) paradigm. Such a paradigm provides units of encapsulation named aspects for crosscutting concerns, which are usually related to non-functional requirements. In traditional approaches such as object-orientation or structured/procedural approaches, such concerns are normally found scattered and tangled into the base concerns, i.e. those that represent the functional requirements.

AT4U: Automated Testing for UML is a verification approach for UML models based on the execution of test cases. The test cases are specified in platform independent fashion, enabling their execution on high-level representation such as UML models. AT4U tool automates the execution of the test cases set. It takes as input the system DERCS model and the set of test cases. The system behaviors indicated in the test cases are executed via simulation, using the input scenario and arguments specified in each test case. The results obtained from the simulation are compared with the test case expected results, indicating whether the behavior passed the test.

DERAF: Distributed Embedded Real-Time Aspects Framework is a framework of aspects created to specify the handling of non-functional requirements in UML diagrams. They represent a subset of requirements commonly found in real-time and embedded system projects. DERAFl aspects are platform independent and provide a high-level semantic on “how” and “where” the aspects adaptations must be applied to handle their related non-functional requirements. In other words, the details about how to implement each aspect adaptations is defined only in the mapping rules for the chosen target platform.

DERCS: Distributed Embedded Real-Time Compact Specification is a PIM that represent the system structure and behavior, using concepts of both object- and aspect-oriented paradigms. A DERCS model is obtained from an UML model through an automatic modeto-model transformation. DERCS meta-model is smaller than the UML one (i.e. it has fewer elements), but it provides the same amount of information specified in the original UML model. Therefore, DERCS is used in the GenERTICA tool to generate source code for a target platform. Moreover, although informal, DERCS provides behavioral semantics that allows the behavior execution simulation that performed by AT4U tool.

FUMBeS: Framework for UML Model Behavior Simulation is the simulation engine used in AT4U tool. This framework uses DERCS behavior execution semantics to simulate the execution of a given behavior, including the manipulation of the system objects runtime information such as attributes and local variables.

IPS: Industrial Packing System is the case study presented in this work to demonstrate the use of AMoDE-RT to design industrial mechatronics systems.

JPDD: Join Point Designation Diagram is the extension of both sequence and class diagrams and is used to represent join points. Join points indicate the elements (from the UML model) that are affected by DERAFl aspect adaptations. During the UML-to-DERCS transformation, the join point selection criteria is evaluated and, if the element matches with such criteria, it is selected and collected in the join point elements list.

MARTE: Modeling and Analysis of Real-Time and Embedded Systems is a UML profile that provides stereotypes to represent element commonly found in the embedded and real-time domains.

GenERTICA: Generation of Embedded Real-Time Code based on Aspects is a script-based code generation tool. GenERTICA uses both DERCS as the input model and scripts as mapping rules to generated source code for distinct target platforms. GenERTICA generates code for system structure and behavior that represent the handling of functional requirements. In addition, GenERTICA weaves the aspects adaptations into the generated code. In other words, it identifies the DERAFl aspects specified in the input model, and executes the aspect adaptation scripts, adding the code to handle the non-functional requirements.

ORCOS: Organic Reconfigurable Operating System is a customizable object-oriented Real-Time Operating System (RTOS) written in C++. ORCOS executes on several platforms, including PowerPC and ARM processors.

PIM: Platform Independent Model is a model that represents system structure and behavior without any implementation details that are related specific to the chosen target execution platform.

PSM: Platform Specific Model is a model that represents system structure and behavior using concepts/elements from a given target execution platform.

RT-FRIDA: Real-Time FRIDA is the requirements engineering approach adopted in AMoDE-RT. It provides a sequence of activities supported by several tools to assist the engineers to identify and specify system functional and non-functional requirements in a separated and consistent way.

RTSJ: Real-Time Specification for Java is the specification that enables real-time application to execute on the Java platform, i.e. it provides a standard API and defines the real-time behavior for the Java Virtual Machine.

TMR: Triple Modular Redundancy is a redundancy technique applied to tolerate system faults by adding three copies of the same computing element. Their outputs are processed on a majority-voting system to produce a single output.

UML: Unified Modeling Language is a modeling language based on the object-oriented paradigm. It is an extensible and standard modeling language whose specification is controlled and managed by the Object Management Group (OMG).

VTL: Velocity Template Language is the script language used in GenERTICA for the specification of mapping rules. VTL is an open source software project directed by the Apache Software Foundation.
References


