Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Evaluating Aspect- and Object-Oriented Concepts to Model Distributed Embedded Real-Time Systems Using RT-UML

Marco A. Wehrmeister[1,4], Edison P. Freitas[1,5], Dalimir Orfanus[3], Carlos E. Pereira[2], Franz Rammig[4]

[1] Instituto de Informática, [2] Dep. Engenharia Elétrica, Federal University of Rio Grande do Sul, Porto Alegre, Brazil, e-mail: mawehrmeister@inf.ufrgs.br, cpereira@ece.ufrgs.br
[3] International Graduate School, [4] Heinz Nixdorf Institute, University of Paderborn, Paderborn, Germany, e-mail: {orfanus, franz}@upb.de
[5] School of Information Science, Computer and Electrical Engineering, Halmstad University, Halmstad, Sweden, e-mail: edison.pignaton@hh.se

Abstract: The growing design complexity of today's embedded real-time systems requires new techniques aiming the raising of the abstraction level since earlier stages of design in order to deal with such complexity in a suitable way. This paper reports a case study, which provides an assessment of two well-know high-level paradigms, namely Aspect- (AO) and Object-Oriented (OO) Paradigms. Concepts of both paradigms were applied at modeling phase of a Distributed Embedded Real-Time System (DERTS). The handling of DERTS' functional and non-functional requirements (at modeling level) using AO and OO concepts is discussed. Both paradigms are compared using of a set of software engineering metrics, which were adapted to be applied at modeling level. The presented results show the suitability of each paradigm for DERTS specification in terms of reusability quality of model elements.

1. INTRODUCTION

The number of functionalities, which have been incorporated into modern embedded real-time systems, can require their deployment over different processing units (which can also be physically separated) in order to fulfill system/design constraints, such as units processing capability, amount of available memory or even components cost. Distributed Embedded Real-Time Systems (DERTS) must perform time-bounded activities, i.e. both processing and communication must respect time constraints without violating other system's constraints and/or requirements. The non-functional nature of some important requirements of DERTS can lead to several problems, such as scattered and tangled handling. If they are not properly treated, these problems increase the overall complexity of design. In this case, reuse of previously developed artifacts (e.g. SW and HW IP blocks) becomes harder. Additionally, SW and HW components are usually designed concurrently with distinct languages and concepts, which increases design complexity too.

Several works propose the raising of abstraction level and separation of concerns in order to manage the growing complexity of DERTS design. Some of them propose the use of high-level concepts from the Object-Oriented (OO) paradigm, as those published in conferences such as IEE ISORC and WORDS. However, the handling of Non-Functional Requirements (NFR) using pure OO concepts is not adequate because there are no convenient abstractions to represent NFR handling. More precisely NFR treatment is done intermixed with the treatment of functional requirements (FR). That situation motivates some works, such as subject-oriented programming (Ossler and Tarr 1999) and aspect-oriented (AO) programming (Kiczales *et al.* 1997), which promote the separation of concerns at implementation level.

Following the idea of raising the abstraction level, it can be observed a trend for DERTS design: the so-called Model-Driven Design (MDD) (Selic 2002) and/or Model-Driven Engineering (MDE) (Schmidt 2006). It is important to highlight that the use of models during design is not a completely new idea, for example engineering have been using models since several years. MDD/MDE claims that models are the main artifacts of design, which should be used to generate (automatically) the system implementation through model transformations. However, at modeling level the mentioned problems of intermixing the handling of requirements from different natures still exist. Thus the separation of concerns with the treatment of FR and NFR should also occur at earlier design stages (e.g. modeling phase). This paper presents a case study focusing on the assessment of the suitability of AO and OO concepts for DERTS modeling, aiming at the treatment of FR and NFR. UML (OMG, 2007) was used to specify two models: one using pure OO concepts, and another one using concepts of AO. Thus, the goals of this paper are: (i) apply AO concepts together with UML at modeling level; (ii) demonstrate the use of UML to model a real DERTS, namely the control system of a products assembler system; (iii) assess both UML models (OO and AO) through software engineering metrics, comparing their strengths and weaknesses; (iv) promote the discussion on the use of AO within design of DERTS.

The paper is organized as follows: section 2 gives an overview of AO basic concepts; section 3 describes the case study (products assembler system) and depicts some

diagrams of OO and AO UML models; the assessment of both models is presented in section 4, where the used metrics are explained and the obtained results presented; finally, section 5 presents some conclusions and future work.

2. ASPECT vs. OBJECT-ORIENTATION

OO and AO modeling are based on the separation of concerns technique. The idea behind separation of concerns is to break down the system into small blocks, which are called concerns. A concern is a focus or interest in a system. In the case of OO, concerns are separated into structure (e.g. classes and attributes) and behavior (e.g. methods).

In opposite to OO, the AO paradigm distinguishes between aspect and base concerns. Base concerns are units of modularization formalizing non-crosscutting concerns, i.e. concerns which do not affect others but can be affected by several aspects. They represent functional requirements. On the other hand, aspects represent crosscutting concerns, i.e. concerns spread over other concerns, which cannot be easily decomposed into separated units. The places where aspects affect base concerns are called join points. During the specification of base concerns, the join points should also be indicated. (Van den Berg *et al.* 2005)

Pointcut (designator), which is part of aspect specification, describes the set of join points (possibly from more than one base concern) into which the aspect will perform adaptations. The process of composition of aspects with base concerns is called weaving. Weaving of aspect can be either static (at design time) or dynamic (at runtime). More exhaustive definitions and common reference model for aspect-oriented modeling can be found in (Schauerhuber *et al.* 2006) and (Van den Berg *et al.* 2005).

3. MODELING A PRODUCTS ASSEMBLER SYSTEM

In order to evaluate the benefits and drawbacks from AO and OO paradigms to specify the model of DERTS, the design of a products assembler system was used as case study. The presented case study was inspired on the packing system presented in (Hodges *et al.* 2006) and (Brusey *et al.* 2006). The system is composed of a robotic arm with a gripper, two conveyors, a storage unit and several sensors. The input conveyor brings individual parts, which are combined to form products. When the sensor detects the presence of a part, the conveyor stops. Then the robotic arm will either put it in the storage unit or use it to assemble a product. On the other side, the second conveyor brings empty boxes into which the parts are assembled. This conveyor remains moving until its sensor detects an empty box. When the product is completely assembled, the controller sends a command to the conveyor and it starts to move forward again. The controller is a periodic active object that verifies if there are products to assemble and/or parts to put into the storage unit. Whether the new product requires a part, which is physically located at the parts conveyor, this part is taken from there and used to mount the product. Otherwise the part is taken from the storage unit. The system was intended to be distributed, i.e. there are four different processing nodes: one responsible to control the products assembly process and the robotic arm; two nodes to control, respectively, the input parts conveyor and the assembled products output conveyor; and one to control the parts storage unit.

In order to use a widely accepted and standardized modeling language, UML was chosen to describe both AO and OO models. For the same reason, the UML profile for Schedulability, Performance and Time (OMG 2003) (also know as real-time profile) was used to represent real-time features of movement control subsystem. Figure 1 shows the functionalities present in the target subsystem. Some of them have NFR (e.g. Assembly cell control), which are depicted as stereotype annotating use cases (e.g. <<NFR_Timing>>). A detailed discussion on requirements of DERTS can be found in (Freitas *et al.* 2007). The following subsections give more details on the modeled subsystem using AO and OO concepts. It is important to highlight that theses models are related to the design phase, i.e. they have more design-related elements than an analysis model, which represents only the concepts of the assembler control system.

3.1 UML Model using OO concepts

The static structure of products assembler control system is depicted in a class diagram. This diagram shows classes, their attributes and methods, and the relationships among classes. Figure 2 presents the class diagram created for the OO version of the UML model of the products assembler control. Classes representing active objects (i.e. those which execute their behavior concurrently with other active objects) are annotated with the <<SAschedRes>> stereotype from the real-time profile. The <<SAresource>> stereotype represents classes of passive objects, which are accessed concurrently by active classes. Frequently, these objects need to have some concurrency control mechanism to assure the validity and integrity of their data. FR and NFR handling classes are shown in the same class diagram. NFR classes are annotated with "NFR_" stereotype, representing the handling of time, distribution and embedded requirements.

The behavior of the control system was specified using sequence diagrams, which show the interaction among objects. Nine different sequence diagrams were created: (i) Products assembler control; (ii) Conveyor control; (iii) Item

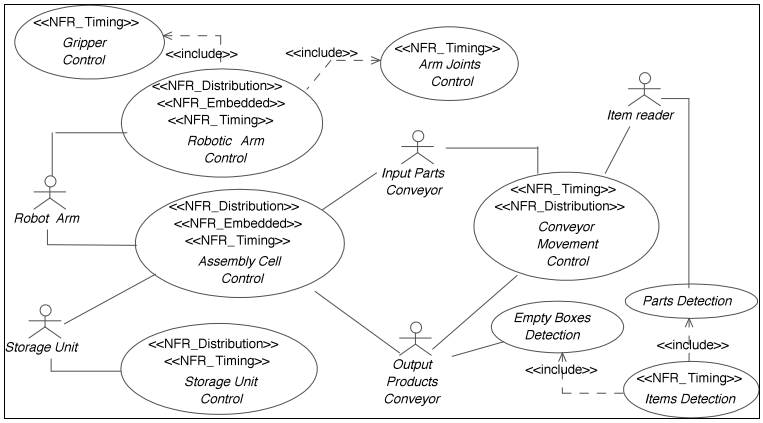

Figure 1. Use case diagram of products assembly cell

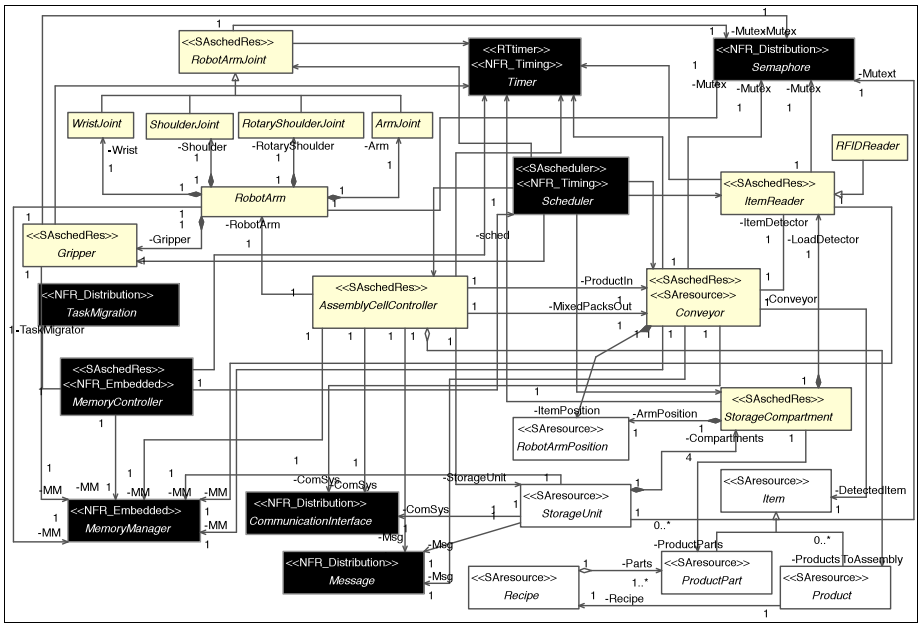

Figure 2. Class diagram of OO version of movement control

detection; (iv) Robotic arm joints control; (v) Gripper control; (vi) Robotic arm movements behavior; (vii) Storage unit control; (viii) Memory control and tasks migration; and (ix) Alarm handling.

Figure 3 shows two fragments of the sequence diagram of products assembler control: (a) the start of active object method responsible to control the products assembly, and (b) the end of active object method execution. The scheduler object periodically (each 5 seconds) sends an activation message to *AssemblyCellController* object. This message is annotated with the <<SAtrigger>> stereotype of real-time profile. A loop operator, indicating the looping nature of the control task, encloses the performed actions. The handling of timing and distribution NFR through respectively *Timer* and *CommunicationInterface* classes is shown in figure 3a. Timer's timeout value is set to the period value assigned to *AssemblyCellController* object. At the end of the controller method (figure 3b) the execution is held until the timeout occurrence (message 53) in order to control the execution frequency. Figure 3a also depicts the sending of messages to other nodes. As stated before, the product assembler control was spread into four nodes. The control task runs in the main node that must access information from the conveyors and storage nodes that are located in other node. Thus the products assembler control task must send messages to them in order to proceed with the products assembly and parts storage. Figure 3b shows the sending of message that requests the position in the storage unit, into which a part should be placed (messages 45-50). Additionally, a method regarding the memory control (message 52 and 54) is also shown.

3.2 UML Model using AO concepts

The AO version of the presented case study uses AO concepts to specify the handling of NFR, i.e. NFR are treated within the scope of a single element instead of been spread over several elements. Figure 4 depicts the class diagram of AO version of products assembler control system. As can be observed, this diagram is simpler than the diagram presented in figure 2 due to the elimination of classes that are not related with the application itself. In order to specify the treatment of NFR, aspects of the Distributed Embedded Real-time Aspects Framework (DERAF) (Freitas *et al.* 2007) were used, and modeled using the Aspect Crosscutting Overview Diagram (ACOD) (a special type of class diagram). One may argue that the same simplification is achieved separating FR from NFR handling classes into two different class diagrams. However, the use of aspects brings other advantages, such as the decrease of number of attributes related to association between FR and NFR classes (see section 4). More details on ACOD modeling will be given in the following paragraphs.

Considering the behavior specification, the number of required sequence diagrams was reduced in one diagram. In the AO version the sequence diagram "Memory control and tasks migration" was removed because the treatment of

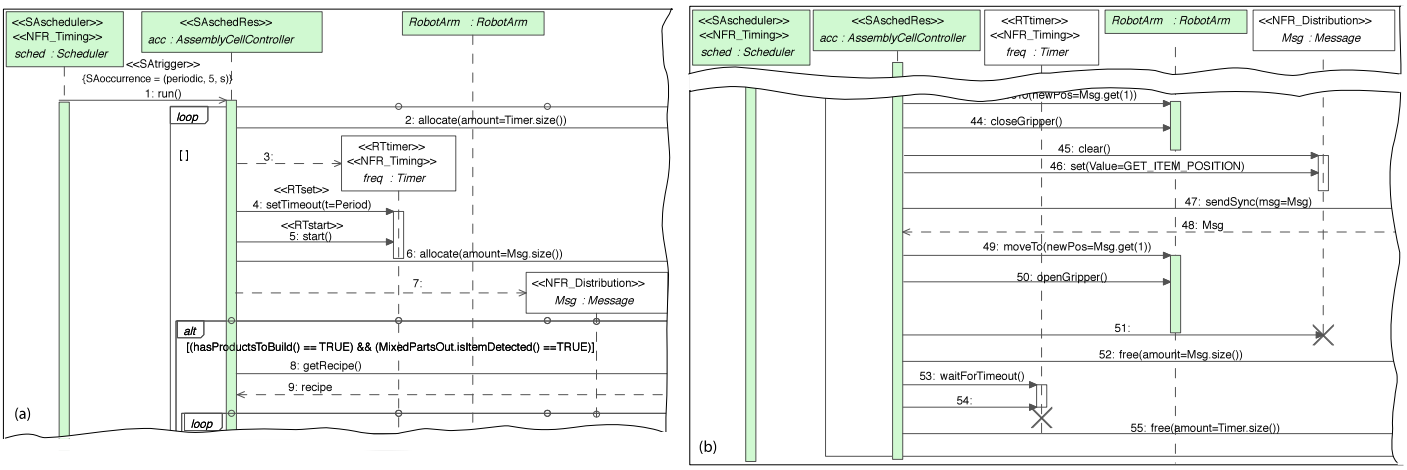

Figure 3. Fragments of OO version's assembly cell control sequence diagram

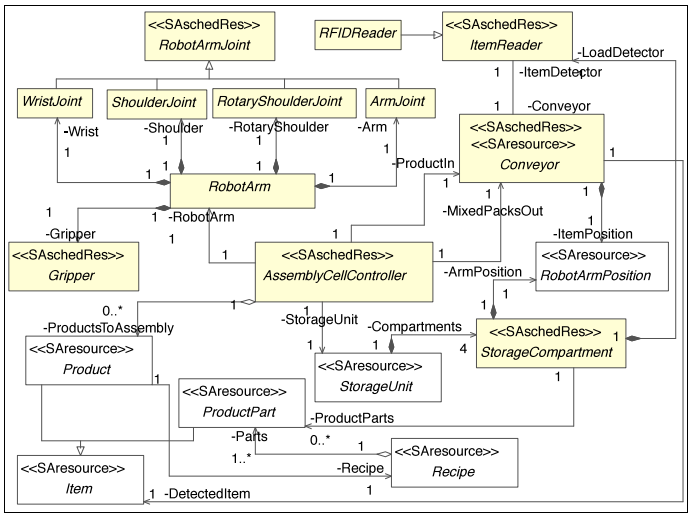

Figure 4. Class diagram of AO version of movement control

memory control and task migration NFR were delegated to, respectively, *MemoryUsageControl* and *TaskMigration* aspects of DERAF. DERAF provides a set of aspects with a pre-defined semantic to handle DERTS NFR, i.e. aspects provide structural and behavioral adaptations which should be used as black boxes at modeling level. A detailed description of DERAF is given in (Freitas *et al.* 2007). Figure 5 shows two fragments of the products assembler control diagram (which are equivalent to those presented in figure 3). As can be observed, all NFR handling were removed, reducing considerably the size of diagrams in terms of number of messages and lifelines (compared with its equivalent in OO version). The amount of messages within the diagram of Figure 5 is 36% smaller than in figure 3. The reduction of complexity in the description of the same feature of the system can be clearly perceived.

DERAF aspects and join points (see section 2) are specified using a combination of ACOD and Join Point Designation Diagrams (JPDD) (Stein *et al.* 2006), as depicted in figure 6. ACOD (figure 6a) is a kind of a class diagram that shows aspects (classes annotated with <<Aspect>> stereotype) affecting FR handling classes. Associations among aspects and classes, those stereotyped with <<Crosscut>>, represent crosscutting concerns, which are handled through aspects adaptations. When some aspect affects the class' structure by adding a new attribute, the crosscut relation can be used to specify the value for the newly inserted attribute, as can be seen in figure 6a. It is important to highlight that crosscutting associations do not insert by themselves new attributes into participating elements (class or aspect) as normal associations do. Pointcuts, which links join points with adaptations, are specified with <<Pointcut>> stereotype (see aspects' highlighted elements in Figure 6a). Figures 6b and 6c show two JPDD representing, respectively, active class and periodic activation join points. Active class joint point represents the selection of all active object classes (i.e. those annotated with <<SAschedRes>> stereotype). Periodic activation represents the selection of all messages, which are annotated with <<SAtrigger>> stereotype, sent by the scheduler to some active object.

At the first impression, the specification of ACOD and JPDD seems to require more effort but it is not true. The generic nature of JPDDs allows their re-use from previous modeled projects, such as happened with this case study. Several JPDDs was simply re-used without modification from the model of a previous designed case study (see (Freitas *et al.* 2007)).

4. MODELS QUALITY ASSESSMENT

The assessment of AO and OO models of products assembler control system was performed using a set of metrics for AO (Sant'anna *et al*. 2003) and OO (Chidamber, Kemerer, 1994) development. A set of metrics is not enough to determine the quality of a system. It is also required to know how those metrics are related to each other, to provide meaningful information about the quality of design. This work uses the assessment framework presented in (Sant'anna *et al*. 2003) to infer the quality of the 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. Implementation related metrics were not used due to the focus of this paper, which does not cover the implementation phase. Additionally, it is important to highlight that this paper concentrates only on "reusability" instead of "reusability and maintainability" as proposed in the assessment framework.

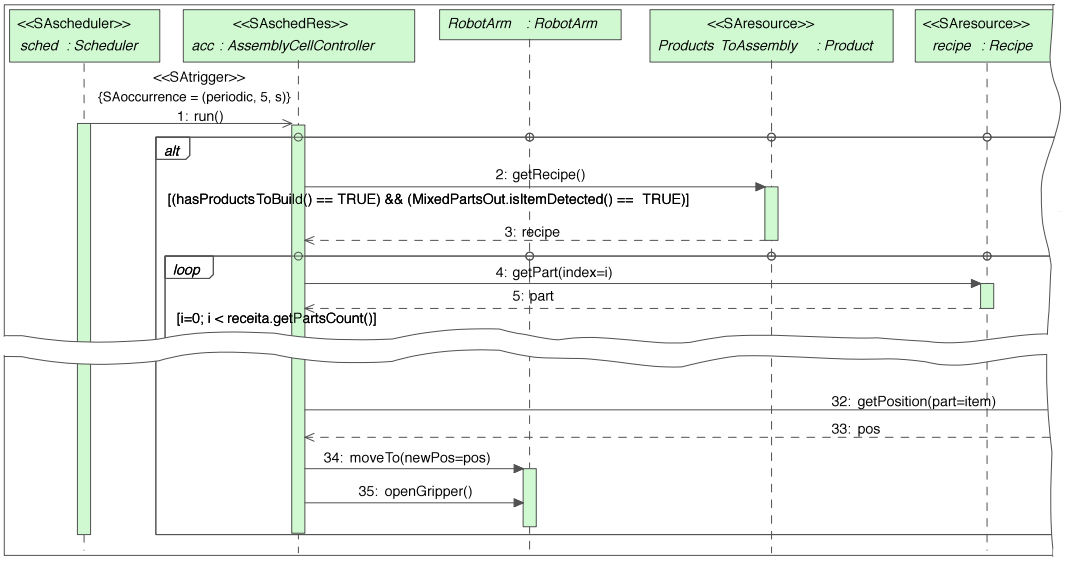

Figure 5. Fragments of AO version's movement control sequence diagram

4.1 Overview of Metrics

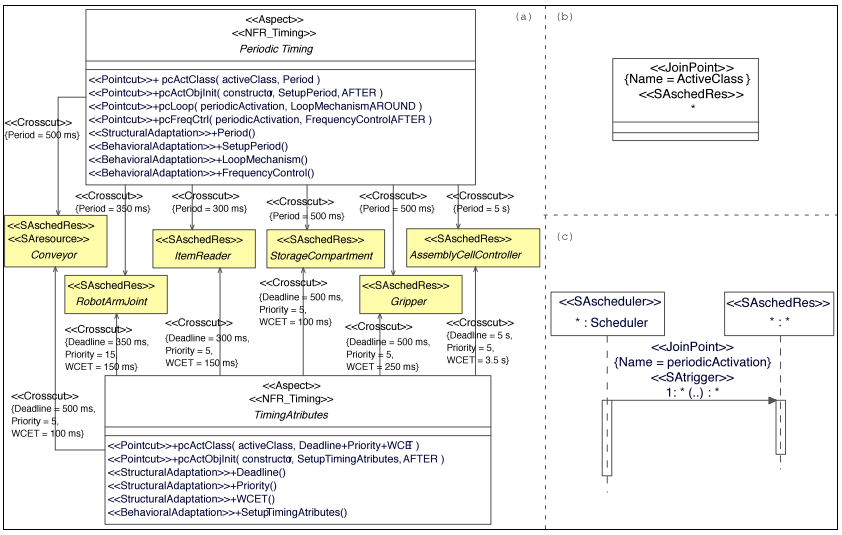

Figure 6. ACOD and JPDD to handle timing NFR

The metrics suite captures information about the design in terms of fundamental attributes such as separation of concerns, coupling, cohesion, and size. For each attribute there is a set of specific metrics, as follows (for details please see (Sant'anna *et al*. 2003)):

1) Separation of Concerns Metrics: they measure the ability to encapsulate the treatment of a concern (see section 2). Two metrics compose this attribute:

i) Concern Diffusion over Components (CDC): it counts the number of components (i.e. aspects or classes) engaged in the handling of a certain concern;

ii) Concern Diffusion over Operations (CDO): it counts the number of operations (i.e. methods or aspect adaptations) related with the handling of a concern.

2) Coupling Metrics: they measure how dependent is an element regarding other system's elements. Two metrics compose this attribute:

i) Coupling Between Components (CBC): it counts the amount of components, which are coupled to a component;

ii) Depth of Inheritance Tree (DIT): it measures the maximum length from a node to the root of inheritance tree.

3) Cohesion Metrics: cohesion is the closeness measure for the relationship of a component with its internal elements. It is translated by the following metric:

i) Lack of cohesion in Operations (LCOO): measures the amount of methods and aspect adaptations, which do not access the same instance attribute set.

4) Size Metrics: measure the size of the model:

i) Vocabulary Size (VS): it counts the number of system components, i.e. the amount of classes and aspects.

ii) Number Of Attributes (NOA): it counts the internal vocabulary of each component, i.e. the number of attributes of each class, and pointcuts of each aspect.

Reusability quality of a model can be seen through two factors: understandability and flexibility. The understandability factor is obtained through separation of concerns, coupling, cohesion and size attributes. Separation of concerns directly affects the understandability of a system, because the more localized the concerns are, the easier is finding and understanding them. The cohesion and coupling indicate the level of independency of one element regarding others. The more independent an element is the easier is to understand it. Model size impacts on understandability due to the amount of elements that should be understood. For the flexibility factor, the key attributes are coupling, cohesion, and separation of concerns. A component is flexible if it is independent or almost independent of the rest of the system, meaning that it represents a specialized part of the system with a specific and well-defined mission. These characteristics are translated into low coupling and high cohesion (i.e. it has a low dependence on other parts of the system) and a good separation of concerns (i.e. the component is responsible for a well defined mission).

4.2 Applying the Metrics Set to Models

As stated above, the application of the described metrics to a system model can provide useful information about system quality related to the reusability. In order to verify the improvement of this system quality, a comparison between the presented models of the control system of the products assembler system is presented. To extract the metrics from the model, a plug-in to Magic Draw UML tool (Magic Draw 2008) was implemented, which can calculate automatically all metrics described in section 4.1.

Considering the separation of concerns metrics, Table 1 shows how effective was the application of aspects from DERAF to handle time, distribution and embedded concerns. All NFR have better separation of treatment in the AO model compared to the OO model, i.e. the smaller number of elements (classes and/or aspects) handling a concern, the better separation of concerns a system has. Therefore, separated concern handling leads to a decrease in the scattering problem. The numbers presented only confirm the simplification observed in the diagrams presented in section 3. The reduction ranges from 66% to 81% for the CDC and from 16% to 75% for the CDO metric. The reason of this decrease is due to CDC/CDO taking into account all elements/methods created specifically to handle a concern, plus all elements referring to them, such as attributes, method calls, method parameters, etc. Thus, the intermixing of FR and NFR treatment, in OO model, causes the inclusion of some FR elements/methods as NFR elements/methods.

	Time		Distribution		Embedded	
	CDC	CDO	CDC	CDO	CDC	CDO
OO	9	12	13	19	11	12
AO	3	10	4	10	2	3

Table 1. Results for Separation of Concerns Metrics

Considering the other metrics, Table 2 depicts the results obtained. Analyzing coupling metrics, DIT results show that the use of aspects did not modified the inheritance tree. The

results of CBC show a decrease of almost than 47% in the AO model. CBC takes into account each reference (e.g. attribute, method call, parameter) to another class/aspect. Thus classes/aspects in AO version are more modular than in OO version mainly due to the intermixed treatment of FR and NFR. Observing the size metric, VS did not change, while NOA has also a decrease of 46%. This happened because several NFR-related attributes were moved to aspects, which are later woven into all affected classes.

Regarding cohesion, the difference of LCOO between AO and OO models is more than 58%. This decrease is primarily caused by elimination of get/set methods for NFR attributes in AO model. LCOO metric does not distinguish two kinds of get/set methods: (i) "raw" which have minimum impact on real cohesion; (ii) with computations which have significant impact on real cohesion. Thus, to provide a fair assessment, we recomputed LCOO for OO with excluded first kind of set/get methods. Even with this exclusion, the decrease of LCOO is still almost 17% in AO model. This shows that even if the OO version is sufficiently cohesive, cohesion of the model can be improved through the use of points.

Internal Attributes	Coupling		Cohesion		Size	
Metric	**CBC**	**DIT**	**LCOO**	**LCOO***	**VS**	**NOA**
OO	203	1	193	96	27	89
AO	108	1	80	80	27	48

Table 2. Results for Coupling, Cohesion, and Size Metrics

Taking into account the results obtained, it can be stated that AO model improve the reusability quality. Almost all metrics have better values for AO model comparing to OO model. Considering the understandability factor, key issues such as separation of concerns, cohesion and coupling had an improvement of more than 55% in average. In spite of the number of components did not change, the number of attributes decreased ca. 45%. For flexibility factor, AO model elements are more cohesive and decoupled compared to OO model. Results regarding separation of concerns show that elements in AO model have more specific and well-defined roles than in OO model.

5. CONCLUSION

This paper presented a case study, which evaluates the use of high-level concepts from AO and OO paradigms, in order to specify DERTS using wide accepted and standardized modeling language such as UML and the real-time profile. DERTS have specific NFR that must be properly handled to manage the increasing of design complexity. It could be seen that AO can help in such quest. In the AO model, through the use of DERAF aspects, the specification simplification of some important diagrams can be an indication for this claim. Moreover, the encapsulation of NFR handling into single units avoids the spread treatment of these requirements.

Regarding the calculated metrics, it could be observed that aspects can impact positively in DERTS design. Several metrics have a substantial decrease in AO model of the case study, ranging from 17% up to 81%. A design is better understood if it has its FR and NFR concerns well separated. This can be seen in sequence diagrams presented in section 3. Here too, the AO version is much easier to understand than OO version. The elements of a design can be reused in other designs with less effort if they are cohesive and decoupled. It is expected that a previously developed component can easily be reused in order to decrease the effort and shorten the time required to design a DERTS. The results show that aspects can help in such quest, decreasing the coupling and increasing cohesion.

Following a MDD approach, the intended future work is to implement a tool that can generate source code for HW and SW components of DERTS. The code should be as complete as possible, i.e. not just code for class skeletons. To support this idea it is necessary to have a tool capable of extracting unambiguous information (FR and NFR handling elements) from UML model. Taking this information as input the code generation tool will apply a set of mapping rules (describing pre-developed APIs e HW IP blocks) to generate the complete DERTS source code.

6. REFERENCES

Brusey, J. *et al*. (2006). *Auto-ID Based Control Demonstration. Phase 2: Pick and Place Packing with Holonic Control.* In: http://www.autoidlabs.org/uploads/media/CAM-AUTOID-WH011.pdf

Chidamber, S.R., Kemerer, C.F. (1994). A Metrics Suite for Object-Oriented Design. In: *IEEE Transactions on Software Engineering*, v.20, n.6, 476-493.

Freitas, E.P. *et al.* (2007). DERAF: A High-Level Aspects Framework for Distributed Embedded Real-Time Systems Design. In: *Proc. of 10th International Workshop on Early Aspects*, 55-74. Springer.

Hodges, S. *et al*. (2006). *Auto-ID Based Control Demonstration. Phase 1: Pick and Place Packing with Conventional Control*. In: http://www.autoidlabs.org/uploads/media/CAM-AUTOID-WH-006.pdf

Kiczales, G. *et al.* (1997). Aspect-Oriented Programming. In: *Proc. of European Conference for Object-Oriented Programming*, 220-240. Springer-Verlag.

MagicDraw Modeling Tool (2008), http://www.magicdraw.com/

OMG (2003). *UML profile for Schedulability, Performance and Time Specification*. http://www.omg.org/cgi-bin/doc?ptc/02-03-03

OMG (2007). *The Unified Modeling Language*. In: http://www.omg.org/technology/documents/formal/uml.htm

Ossler, H., Tarr, P. (1999). Using subject-oriented programming to overcome common problems in object-oriented software development/evolution. In: *Proc. of 21st Int. Conf. on Software Engineering*, 687-688. IEEE Computer Society Press.

Sant'anna, C.N. *et al*.(2003). On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. In: *Proc of Brazilian Symposium on Software Engineering*, 19-34.

Schauerhuber, A. *et al.* (2006). Towards a common reference Architecture for Aspect-Oriented Modeling. In: *Proc. of 8th Intl. Workshop on Aspect-Oriented Modeling*, AOSD'06

Schmidt, D.C.(2006). Gest Editor's Introduction: Model-Driven Engineering. In: *IEEE Computer*,v.39,n.2, 25-31.

Selic, B. (2003). The Pragmatics of Model-Driven Development. In: *IEEE Software*, v. 20, n. 5, 19-25.

Stein, D. *et al*. (2006). Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design. In: *Proc. of 5th International Conference Aspect-Oriented Software Development*, 15-26. ACM Press.

Van den Berg, K. *et al*. (2005). *AOSD Ontology 1.0 – Public Ontology of Aspect-Orientation*. Technical Report AOSD-Europe-UT-01, AOSD Europe.