A Comparison of the Use of Aspects and Objects for Modeling Distributed Embedded Real-Time Systems with RT-UML

Marco A. Wehrmeister\textsuperscript{1,4}, Edison P. Freitas\textsuperscript{1,5}, Dalimir Orfanus\textsuperscript{3}, Carlos E. Pereira\textsuperscript{1,2}, Franz J. Rammig\textsuperscript{4}

\textsuperscript{1}Instituto de Informática, \textsuperscript{2}Dep. Engenharia Elétrica
Universidade Federal do Rio Grande do Sul (UFRGS) – Porto Alegre – Brazil

\textsuperscript{3}Intl. Graudate School on Dynamic Intelligent Systems, \textsuperscript{4}Heinz Nixdorf Institute
University of Paderborn – Paderborn – Germany

\textsuperscript{5}School of Information Science, Computer and Electrical Engineering
Halmstad University – Halmstad – Sweden

mawehrmeister@inf.ufrgs.br, edison.pignaton@hh.se, orfanus@upb.de, cpereira@ece.ufrgs.br, franz@upb.de

Abstract. The growing design complexity of today’s 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 are been incorporated into modern embedded real-time systems, can require their deployment over different processing units (which can also be modernly designed) 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 bring 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. During integration phase, using of different languages can cause more problems of another kind, especially when some requirements were misunderstood.

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 the IEEE Intl. Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC) or the IEEE Intl. Workshop Object-Oriented Real-Time Dependable Systems (WORDS). However, the handling of Non-Functional Requirements (NFR) using pure OO concepts is not adequate because there are no special 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; Tarr, 1999] and aspect-oriented (AO) programming [Kiczales, 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, 2003] and/or Model-Driven Engineering (MDE) [Schmidt,
It is important to highlight that the use of models during design is not a completely new idea, for example engineering make use of models a long time ago. 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, 2008] 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 an Unmanned Aerial Vehicle (UAV); (iii) assess both UML models (OO and AO) through software engineering metrics [Halstead, 1977], showing the benefits and drawbacks of both approaches; (iv) promote the discussion on the use of AO concepts 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 (UAV control 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 and the obtained results are presented; finally, Section 5 presents some conclusions and future work.

2. Overview of Basic Concepts

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 classes, attributes and 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. In our case it represents FR. 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, which should also be indicated during the specification of base concerns.

Pointcut 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. Case Study: Unmanned Aerial Vehicle

In order to evaluate the benefits and drawback from AO and OO paradigms to specify the model of DERTS, the design of an UAV was used as case study. UAV is an aircraft that flies without having onboard pilot, which are used in activities where the human presence is avoided due to inherited risks, or simply to decrease costs. It can fly a pre-programmed route or be operated through a ground station. Reconnaissance support in natural disasters, monitoring and defect detection of transmission lines located in inhospitable places, and area vigilance are some examples of UAV applications. An UAV is compounded of several subsystems, such as video recording and transmission, navigation, mission management, collision avoidance, self-diagnostic, and movement control. Due to space constraints this case study focuses only on the movement control subsystem of an unmanned helicopter. The helicopter control system has two interconnected real-time processing nodes: one controls the main rotor and the other controls the back rotor, i.e. the control system is distributed over these two communicating nodes.

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] 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. Helicopter Movement 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 these models are related to the design phase, i.e. they have more design-related elements than an analysis model, which represents only UAV control system concepts.
3.1. UML Model using OO concepts

The static structure of UAV movement control is depicted using 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 movement control UML model. 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 <<SAre source>> 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, respectively with and without filling, in the same class diagram. NFR classes are annotated with “NFR_” stereotype, representing the handling of time, distribution and embedded NFR.

![Figure 1. Use case diagram of UAV movement control system](image)

The behavior of the UAV control system was specified using sequence diagrams, which show the interaction among objects. Nine different sequence diagrams were created: (i) Helicopter movement control; (ii) Back rotor actuation; (iii) Change control policy; (iv) Main rotor movement encoder; (v) Back rotor movement encoder; (vi) Environment data acquisition; (vii) Energy control; (viii) Task migration; and (ix) Alarm signalization.

![Figure 2. Class diagram of OO version of UAV movement control](image)

Figure 3 shows two fragments of the helicopter movement control sequence diagram: (a) the start of active object method responsible to control the movement, and (b) the end of active object method execution. The scheduler object sends an activation message periodically (each 15 ms) to MovementController object. This message is annotated with the <<SA trigger>> 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 Semaphore classes, is shown in figure 3a. Timer’s timeout value is set to the period value assigned to MovementController object. At the end of the controller method (figure 3b) the execution is held until the timeout occurrence (message 40) in order to control the execution frequency. Figure 3a also depicts the synchronized access (using semaphore) to the shared resource MovementInformation object, which is written by the Movement Encoder active object and read by MovementController object. As stated before, the control system has one processing node at the main rotor and another one at the back rotor. The control task runs in the main rotor node while the back rotor actuation runs in its node. Thus the movement control task must send the calculated actuation values to the back rotor node. Figure 3b shows the handling of this communication NFR (messages 28-35), and also the application of actuation value for the main rotor. Additionally, a method regarding the energy control (message 39) is also shown.

![Figure 3. Fragments of the movement control sequence diagram of OO version](image)

### 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 movement control. 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. 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 the AO version the following sequence diagrams were eliminated: (i) Back rotor actuation; (ii) Back rotor movement encoder; (iii) Energy control; and (iv) Task migration. The last two diagrams (iii and iv) are useless because the treatment of energy control and task migration NFR were delegated to, respectively, EnergyControl 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]. Diagrams (i) and (ii) were merged with, respectively, “Helicopter Movement Control” and “Main Rotor Movement Encoder” sequence diagrams. Figure 5 shows two fragments of the movement 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 mentioned diagram incorporation is shown by the “par” operator which means that both interactions occur concurrently.
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]). Additionally, observing the presented diagram of both versions, the simplification of UAV control system specification can be clearly perceived.

![Figure 6. ACOD and JPDD to handle timing NFR](image)

### 3. Assessing the Models

The assessment of AO and OO models of UAV’s control system was performed using a set of metrics specific to AO development [Sant’anna et al., 2003], which was derived from the set of OO metrics presented in [Chidamber; Kemerer, 1994]. 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.

#### 4.1. Overview of Metrics

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):
   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:
   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:
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. number of attributes of each class, and pointcuts of aspect.

Reusability quality of a model can be seen through two factors: understandability and flexibility. The understandability factor is obtained through four key attributes: separation of concerns, coupling, cohesion and size. 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 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. 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 UML models of UAV is presented. To extract the metrics from the model, a plug-in to Magic Draw UML tool [No Magic, 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 concerns handling leads to a decrease in the scattering problem. The numbers presented confirm the simplification observed in the diagrams presented in section 3. The reduction ranges from 55% to 83% for the CDC and from 75% to 92% for the CDO metric. CDC/CDO became smaller in AO version because the way they are calculated (see section 4.1). For instance, in AO version, CDC for timing NFR considers only the following aspects from DERAF: PeriodicTiming, SchedulingSupport, TimingAttributes and TimeParametersAdapter. While in OO version, CDC takes into account the classes specifically related to timing NFR handling (Scheduler and Timer) plus those related to FR, which also deal with time issues (MovementController, MovementEncoder, EnvironmentDataSampler, BackRotorSensor Driver, BackRotor Actuator, Alarm and EnergyController). Thus, the intermixing of FR and NFR treatment, in OO model, causes the inclusion of some FR elements/methods as NFR elements/methods.

Table 1. Calculated Metrics

<table>
<thead>
<tr>
<th>Internal Attributes</th>
<th>Separation of Concerns</th>
<th>Coupling</th>
<th>Cohesion</th>
<th>Size</th>
</tr>
</thead>
</table>
|                     | Timing                 | Distribution | Embedded | CDC  | CDI | CDO | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC | CDC |CDC
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 issue such as separation of concerns, cohesion and coupling had an improvement of more than 50%. In spite of the number of components did not change, the number of attributes decreased ca. 50%. For flexibility factor, AO model elements are more cohesive and decoupled compared to OO model. Separations of concerns results show that elements in AO model have more specific and well-defined roles than in OO model.

3. 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. Through the use of DERAacets aspects, in AO model, 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 UAV case study, ranging from 55% up to 91%. 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 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 model 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 and HW IP blocks) to generate the complete DERTS source code.

References


