Halmstad University Post-Print

A Case Study to Evaluate Pros/Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems

Marco A. Wehrmeister, Edison Freitas, Dalimir Orfanus, Carlos E. Pereira
and Franz Rammig

N.B.: When citing this work, cite the original article.

©2008 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Wehrmeister M, Pignaton de Freitas E, Orfanus D, Pereira C, Rammig F. A Case Study to Evaluate Pros/Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems. In: 5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, 2008. MOMPES 2008. Los Alamitos, California: IEEE Computer Society; 2008. p. 44-54.

DOI: <http://dx.doi.org/10.1109/MOMPES.2008.9>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-2056>

A Case Study to Evaluate Pros/Cons of Aspect- and Object-Oriented Paradigms to Model Distributed Embedded Real-Time Systems

Marco A. Wehrmeister^{1,4}, Edison Freitas¹, Dalimir Orfanus³, Carlos E. Pereira², Franz Rammig⁴

¹ Instituto de Informática, ² Dep. Engenharia Elétrica – UFRGS, Brazil

³ Intl. Graduate School, ⁴ Heinz Nixdorf Institute – University of Paderborn, Germany

{mawehrmeister, epfreitas}@inf.ufrgs.br, cpereira@ece.ufrgs.br, {orfanus, franz}@upb.de

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 are 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 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, or models) becomes harder. Additionally, SW and HW components are usually designed concurrently with distinct languages and concepts, which also increases

the design complexity. During integration phase, the use of different languages can cause more problems, especially when 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 [1] and aspect-oriented (AO) programming [2], 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) [3] and/or Model-Driven Engineering (MDE) [4]. It is important to highlight that the use of models during design is not a completely new idea, for example other engineering disciplines 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 [7] 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, 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 and discusses aspects within DERTS domain; 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 are explained and the obtained results presented; finally, section 5 draws some conclusions and future work.

2. Concerns within DERTS domain

This section begins with a brief overview of concepts of AO paradigm. Following a discussion on how aspects can help the designer with the NFR handling.

2.1. Overview of 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. During the specification of base concerns, the join points should also be indicated.

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 AO modeling can be found in [12] and [13].

2.2. Handling NFR with aspects

DERTS domain has very specific NFR, which must be taken into account during the design. Deadlines, activation periods, synchronization, communication, area and memory footprint and energy consumption are examples of such NFR. A detailed discussion on NFR of DERTS domain can be seen in [9].

In order to treat NFR since early phases, a framework of aspects named DERAf (Distributed Embedded Real-time Aspects Framework), has been created. DERAf is an extensible high-level aspects framework (based on the aspect orientation conceptual model proposed in [12]), which provides modularization in order to handle NFR (see [9]) since the modeling phase.

DERAf was intended to be used together with UML and the profile for real-time systems [8]. The main idea behind DERAf is to provide aspects, which enhance the modeled system by adding specific behavior and structure to handle NFR during the modeling phase, without binding the model with a specific implementation technology. To reach this implementation independence, details on how to implement the aspect adaptations were abstracted, i.e. the designer selects which aspects should be used (based on the high-level semantics of aspect adaptations), defining which elements are affected by the selected aspects. The set of aspects available in DERAf is depicted in Figure 1. It is important to highlight that the internal mechanisms of aspects (i.e. the implementation of aspects' adaptations) are not covered by this paper.

Each NFR can be handled by one or more aspect, thus a brief description of the behavioral/structural adaptations applied by DERAf aspects is necessary:

- **TimingAttributes:** adds timing attributes to active objects (e.g. deadline, priority, WCET, start/end time, and so on), and also the corresponding initialization of these attributes;
- **PeriodicTiming:** adds a periodic activation mechanism to active objects. This improvement requires the addition of an attribute representing the activation period and a way to control the 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 the scheduling list, as well as the execution of the feasibility test to verify if the scheduling list is schedulable;
- **TimeBoundedActivity:** adds the mechanism to restrict the maximum execution time for an activity

Figure 1. Set of aspects provided by DERAf

(e.g. limits the time which a shared resource can be locked by an active object). Time counting begins immediately before the starting of the activity and a way to interrupt and stop this execution must be provided if the time limit is reached;

- **Jitter:** measures the start/end of an activity, calculates its variation. If the tolerated variance was overran, corrective actions must be taken;
- **ToleratedDelay:** restricts the time to start the execution of an activity (e.g. limits the time which an active object can wait to acquire a lock on a shared resource). This aspect adds a time count mechanism, starting it immediately before the beginning of the execution and, if the maximum tolerated delay is reached, corrective actions must be performed;
- **DataFreshness:** associates timestamps representing the validity period to data, which should be verified before using them. After the writing of “validity controlled data”, the timestamp must be updated. Analogously, before reading them, 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);
- **ClockDrift:** measures the time at which an activity starts and compares it with the expected beginning of this activity. If the accumulated difference

exceeds the maximum tolerated clock drift, then corrective actions should be executed;

- **ConcurrentAccessControl:** adds a mechanism to control the access of shared resources. Before accessing a shared resource, a permission should be requested to the control mechanism and, after the use of the resource, the control mechanism must notify that the lock on the shared resource can be released;
- **MessageSynchronization:** adds a waiting mechanism which pauses the execution until the arrival of an acknowledge message. The waiting mechanism can be implemented either as a busy wait or by blocking the active object execution and calling the scheduler;
- **MessageAck:** adds a message delivery guarantee mechanism. This aspect has two facets: (i) at sender side, after a message sending, the communication mechanism must be notified that a message was sent and an acknowledge message must arrive; (ii) at receiver side, after delivering a message, an acknowledge message must be sent;
- **MessageIntegrity:** verifies the integrity of a received message. At sender side, before sending the message, a checker algorithm (e.g. parity, CRC, etc) must generate check information that will be included into the message. At receiver side, after receiving the message, the checker algorithm must generate the check information from the received message, comparing it with the information received within the message;
- **MessageCompression:** adds a compression mechanism to improve the bandwidth usage. At sender side, the message is compressed before its sending and, at receiver side, the message is decompressed before delivering it to application;
- **EnergyMonitoring:** inserts an energy monitoring mechanism to measure the energy consumption of an activity. Before the execution, the energy level is measured and, after the end of the execution, the energy level is measured again and the difference is calculated and stored;
- **EnergyControl:** adds a mechanism, which implements an energy control policy that performs control actions depending on the remaining energy level. Migration of active objects, to loosen temporal requirements, to decrease system frequency, or shutdown unnecessary hardware are

some examples of control actions which can be implemented.

- **MemoryUsageMonitoring:** inserts a mechanism to provide information on the total memory used by the system. Before every memory allocation/release, the amount of requested/released memory should be added/subtracted to 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 the use of FPGA area. If reconfiguration techniques are used, the total free area of the FPGA must be increased/decreased with the amount of area used in each reconfiguration;
- **HwAreaControl:** verifies if the requested hardware reconfiguration is possible and, if so, allows the reconfiguration;
- **TaskMigration:** provides a mechanism to migrate active objects between nodes, or from software to hardware, or the opposite. It is used by the aspects that control embedded concerns (*EnergyControl*, *MemoryUsageControl*, and *HwAreaControl*), which are responsible for the decision on migration;
- **NodeStatusRetrieval:** inserts a mechanism to retrieve information about processing load, message send/receive rate, and/or the node availability (i.e. "I'm alive" message). Before/after every start/end of execution of an active object, the processing load is calculated. Before/after every sent/received message, the message rate is computed. Additionally, the node availability message is sent at every "n" messages or periodically with an interval of "n" time units.

Same considerations regarding DERAf should be written. As stated previously, the goal of DERAf is to provide high-level aspects to be used at the modeling phase with UML in order to handle NFR. The implementation of DERAf aspects must be realized through either application or platform code, binding aspects with an implementation technology. The high-level semantics of "how" and "where" each aspect affects system elements are pre-defined and must be preserved, such that every implementation must follow these semantics in order to allow the reuse of previously developed aspects implementation. The idea is to build a library of aspects implementations, which could be easily reused in further designs, thus reducing the design effort.

Sometimes, it is also interesting to evaluate the impact of the aspects implementation into the

original DERTS at the modeling phase. Therefore, it is necessary to provide models to describe how each aspect implementation affects the original specification. On other words, it is necessary to provide an aspects model weaving, like the ideas presented in [11]. 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, which aims on the evaluation of AO and OO paradigms to model DERTS.

Finally, it is important to highlight that DERAf does not provide aspects to handle all NFR present in a DERTS design. Fault-tolerance is an important NFR that still does not have support in the initial version of DERAf. However, DERAf was intended as an extensible framework, so the support to fault-tolerance is possible and can be incorporated in further versions.

3. Modeling DERTS using OO and AO

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

Figure 2. Use case diagram of movement control system

interconnected real-time processing nodes: one controls the main rotor and the other controls the back rotor. On other words the designed 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 [8] (also know as real-time profile) was used to represent real-time features of movement control subsystem. Figure 2 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>>). 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. UAV model using pure 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 3 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 <<SResource>> 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 requirements.

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 4 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 <<SAttrigger>> stereotype of real-time profile. A loop operator, indicating the repetition 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 4a. Timer’s timeout value is set to the period value assigned to *MovementController* object. At the end of the controller method (figure 4b) the execution is held until the timeout occurrence (message 40) in

Figure 3. Class diagram of OO version of movement control

Figure 4. Fragments of OO version's movement control sequence diagram

order to control the execution frequency. Figure 4a also depicts the synchronized access (using semaphore) to the shared resource *MovementInformation* object, which is written by the *MovementEncoder* active object and read by *MovementController* object. Therefore every time that the *MovementInformation* object is used, an exclusive access to this object is requested and also released immediately after its use.

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 own node. Thus the movement control task must send the calculated actuation values to the back rotor node. Figure 4b 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.

3.2. DERAf + UML: UAV model using AO concepts

The AO version of the presented case study uses DERAf aspects to specify the handling of NFR, i.e. a NFR is treated within the scope of a single element instead of been spread over several elements. Figure 5 depicts the class diagram of AO version of movement control. As can be observed, this diagram is simpler than the diagram presented in figure 3 due to the elimination of classes that are not related with the application itself. On other words, the handling of NFR is done using aspects from DERAf, which are specified 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 of OO version 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 (see section 3). As stated previously, the semantic of DERAf's aspects is pre-defined and the aspects should be used as black boxes at modeling level. Diagrams (i) and (ii) were merged with, respectively, "Helicopter Movement Control" and "Main Rotor Movement Encoder" sequence diagrams. Figure 6 shows two fragments of the movement control diagram (which are equivalent to

Figure 5. Class diagram of AO version of movement control

those presented in figure 4). 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 merge is shown through the “par” operator which means that both interactions occur concurrently.

DERAF aspects and join points (see section 2) are specified using a combination of ACOD and Join Point Designation Diagrams (JPDD) [10], as depicted in figures 7 and 8. ACOD (figures 7a and 8a) 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

The diagram shows a class hierarchy and associations. At the top, a purple box labeled `<<SASchedRes>>` contains the class `BackRotorActuator`. Below it is a green box `MainRotorActuator`, which has a directed association to a green box `RotorActuator` with multiplicity `1..*` at the `RotorActuator` end. To the left, a green box `SensorDriver` has a directed association to `RotorActuator` with multiplicity `1..*` at the `RotorActuator` end. Below `SensorDriver` is another green box `SensorDriver`. At the bottom, a green box `EnvironmentSensingSubsystem` has a directed association to `RotorActuator` with multiplicity `1` at the `RotorActuator` end. To the right, a purple box labeled `<<SASchedRes>>` contains the class `EnvironmentDataSampler`. It has a directed association to `SensorDriver` with multiplicity `1` at the `EnvironmentDataSampler` end and `1..*` at the `SensorDriver` end. There is also a self-association on `EnvironmentDataSampler` with multiplicity `1` at both ends. The diagram is titled "Environment control" at the bottom left.

Figure 6. Fragments of AO version's movement control sequence diagram

Figure 7. ACOD and JPDD to handle timing NFR

Figure 8. ACOD and JPDD to handle synchronization NFR

messages, which are annotated with `<<SATrigger>>` stereotype, sent by the scheduler to some active object.

Figure 8 shows the *ConcurrencyControl* aspect affecting passive objects, which store information that can be simultaneously accessed by more than one active object. *ConcurrencyControl* aspect affects the passive objects by assigning a concurrency control mechanism to each affected object during the creation of the object, whose join point is captured by the JPDD of figure 8b. Additionally, before every access (read or write) to a protected object, the access permission should be requested to the control mechanism, as well as released after their use. The write accesses are captured through the join point depicted by the figure 8c. It is important to highlight that following DERAf

premises of high-level aspects, at modeling phase, it does not matter if this inserted control mechanism is a new attribute for each affected classes or simply an new object, which is only associated with the protected shared object. These are implementation specific issues, which should be decided at implementation phase.

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 what happened in this case study. Several JPDDs was simply re-used without modification from the model of a previous designed case study (see [9]). Additionally, observing the presented diagram of both versions, the simplification of UAV control system specification can be clearly perceived.

4. Evaluating both UML models

The assessment of AO and OO models of UAV's control system was performed using a set of

metrics specific to AO development [5], which was derived from the set of OO metrics presented in [6]. 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 [5] to infer the quality of the presented models by measuring its reusability quality. 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. Following, it will be given a small description of the used metrics set and also the results obtained from applying it in the presented models.

4.1. A brief overview on the used metrics set

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 [5]):

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 (see figure 9). 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 concerns are, the easier is finding and

Figure 9. Reusability quality model

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. Obtained results

As stated above, the use 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 quality, a comparison between the presented models of the control system of UAV is presented. To extract the metrics from the model, a plug-in to Magic Draw UML tool [14] was implemented, which can calculate automatically all metrics described in section 4.1.

Considering the separation of concerns metrics, figure 10 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

Figure 10. Results for Separation of Concerns Metrics

following aspects from DERAf: *PeriodicTiming*, *SchedulingSupport*, *TimingAttributes* and *TimeParameters Adapter*. 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*, *BackRotorSensorDriver*, *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.

Considering the other metrics, figure 11 depicts the results obtained. Analyzing coupling metrics, DIT results show that the use of aspect did not modified the inheritance tree. The results of CBC show, again, a decrease of more than 55% 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 that happens in OO version. Observing the size metric, VS did not change, while NOA has a decrease of 52%. This happened because several NFR-related attributes were moved from classes to aspects, which are later woven into all affected classes.

Figure 11. Results for Coupling, Cohesion, and Size Metrics

Regarding cohesion, the difference of LCOO between AO and OO models is more than 91%. This decrease is primarily caused by elimination of get/set methods for attributes related to NFR handling. 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 “raw” set/get methods (LCOO* in figure 11). Even with this exclusion, the decrease of LCOO is 75% in AO model. The obtained results show that using aspect the cohesion of model is improved.

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.

5. Conclusions and Future Work

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 UML profile for Schedulability, Performance and Time. 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 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 4. 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 and HW IP blocks) to generate the complete DERTS source code.

7. References

- [1] Ossler, H.; Tarr, P., "Using subject-oriented programming to overcome common problems in object-oriented software development/evolution", Proc. of 21st Int. Conf. on Software Engineering, IEEE Computer Society Press, 1999, pp. 687-688.
- [2] Kiczales, G. et al., "Aspect-Oriented Programming", Proc. of European Conference for Object-Oriented Programming, ECOOP, LNCS 1241, Springer-Verlag, (1997), pp. 220-240
- [3] B. Selic, "The Pragmatics of Model-Driven Development", IEEE Software, v. 20, n. 5, pp. 19-25, Sept.-Oct., 2003
- [4] D.C. Schmidt, "Gest Editor's Introduction: Model-Driven Engineering", IEEE Computer, v. 39, n.2, pp.25-31, Feb. 2006
- [5] C. N. Sant'anna et al, "On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework", Proc of Brazilian Symp. Software Engineering, Brazil, 2003, pp. 19-34.
- [6] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object-Oriented Design", IEEE Trans. Software Engineering, v.20, n.6, 1994, pp.476-493.
- [7] Object Management Group, "The Unified Modeling Language", in <http://www.omg.org/technology/documents/formal/uml.htm>
- [8] Object Management Group, "UML profile for Schedulability, Performance and Time Specification", available in: <http://www.omg.org/cgi-bin/doc?ptc/02-03-03>
- [9] Freitas, E.P. et al. "Using Aspect-Oriented Concepts in the Requirements Analysis of Distributed Real-time Embedded Systems", Proc. of IFIP International Embedded Systems Symposium (IESS'07), Springer, 2007, pp.221-230.
- [10] D. Stein et al. "Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design", Proc. of 5th Int. Conf. Aspect-Oriented Software Development, ACM Press, 2006, pp.15-26
- [11] D. Stein, et al. "A UML-based Aspect-Oriented Design Notation for AspectJ", Proc. of Intl. Conference on Aspect-Oriented Software Development, ACM Press, 2002, 106-112
- [12] A. Schauerhuber et al., "Towards a common reference Architecture for Aspect-Oriented Modeling", Proc. of 8th Intl. Workshop on Aspect-Oriented Modeling, in conjunction with AOSD'06, Bonn, Germany, March, 2006.
- [13] Van den Berg, K. et al, "AOSD Ontology 1.0 – Public Ontology of Aspect-Oriented", Technical Report AOSD-Europe-UT-01, AOSD Europe, May 2005.
- [14] MagicDraw Modeling Tool, <http://www.magicdraw.com/>