An Approach to Model-Driven Architecture Applied to Space Real-Time Software

Alessandro Gerlinger Romero[1] and Mauricio Gonçalves Vieira Ferreira[2]
National Institute for Space Research (INPE), São José dos Campos, São Paulo, 12227-010, Brazil

Real-time systems are commonplace in satellites. In this system type, software has become a crucial factor to satellite's success projects because its complexity quickly increases, along with cost. Some factors that contribute to increase complexity of software are: it interacts with different kind of hardware, it has several states and for each state commonly a different control law, it has hard-deadlines, and it must have a high level of reliability. Attitude and Orbit Control System (AOCS) is a good example for this type of system. With the necessity to launch more satellites, Brazilian National Institute for Space Research (INPE) has been carrying out research on modeling and verifying real-time software, like a lot of other space agencies and research institutes. The main focus is to obtain a better balance between dependability, schedule, and cost. However, instead of creating one more brand-new, one-of-a-kind approach, method or process, we are trying to use Object Management Group (OMG) specifications, which have been proposed and adopted by community in some degree. Another concern from this INPE research is to be independent from commercial tools establishing itself on open source software. This paper presents a detailed approach to implement Model-Driven Architecture (MDA) in real-time space software based strongly in OMG specifications. It shows how models are defined, linked, verified and transformed, as well as a set of tools for this. We place special emphasis on fUML (Semantics of a Foundational Subset for Executable UML Models) and MARTE (UML Profile for Modeling and Analysis of Real-Time Embedded Systems) that allow us to define a completely executable Platform Independent Model (PIM). At the end, a case study is presented, along with an assessment of the proposed approach. This assessment allowed us to conclude that MDA, following the proposal presented, has advantages versus the current approaches applied to real-time space software development.

I. Introduction

REAL-TIME systems are commonplace in satellites. In this system type, software has become a crucial factor to satellite's success project, because its complexity quickly increases and together cost. Some factors that contribute to increase complexity of software are: it interacts with different kind of hardware, has several states and for each state commonly a different control law, has hard-deadlines and needs to have a high level of reliability. Attitude and orbit control (AOCS) is a good example for this type of system.

According to Giese *et al.* (2010), the current practice is to follow a manual process for real-time software development. The manual process provides an opportunity for optimization through Model-Driven Architecture (MDA) (OMG, 2003).

According to the OMG (OMG, 2003), the goal of MDA is to provide an open, vendor neutral approach to the challenge of business and technology change. MDA (Model-Driven Architecture) is a key initiative to promote software productivity, portability and maintainability, which places modeling at heart of software development process. Platform Independent Model (PIM) is a crucial model in MDA allowing independence of platform and early verification.

[1] Doctoral student of course Space Engineering and Technology-ETE, Option Engineering and Management of Space Systems-CSE at INPE, Av. dos Astronautas, 1758, building CCS, São José dos Campos, 12227-010, Brazil, romgerale@yahoo.com.br

[2] Doctor Engineer/Researcher, CRC (Satellite Tracking and Control Center), Av. dos Astronautas, 1758, building CCS, São José dos Campos, 12227-010, Brazil, mauricio@ccs.inpe.br

Despite of MDA is not a new approach; it remains far from being commonly used on projects in real systems (Lettner and Tschernuth, 2010). This is even more outstanding in real-time software (Giese *et al.*, 2010) although MDA is defined in a very high level abstraction. It is necessary to define a comprehensive approach by using the OMG's specifications and profiles to put it into practice.

In this paper, we present a detailed approach to use the MDA for real-time software, highlighting how the models are defined and chained, and open source tools that can be used to do this.

Current paper uses Platform Independent Models (PIM) to model functional structure and behavior using fUML (Semantics of a Foundational Subset for Executable UML Models) (OMG, 2009a). fUML defines a complete computer language using a subset of UML so PIMs can be executed and verified. Non-functional properties are captured and modeled still in PIM using MARTE (UML Profile for Modeling and Analysis of Real-Time Embedded Systems) (OMG, 2009c). Test scenarios are defined using fUML and they support PIM verification.

Independently, target platform is defined using UML and two transformations specifications: MOF QVTO (Query/View/Transformation Operational) (OMG, 2008) and MOF M2T (Model to text) (OMG, 2008b).

Afterwards, PIM is transformed in a Platform Specific Model (PSM) using previous defined platform. Current paper uses explicit mapping, .i.e. building a model (PSM) that contains only concepts provided by the targeted platform. This transformation is accomplished using MOF QVTO.

PSM is transformed in an AADL model, and schedulability analysis is performed. Finally, code is generated.

The remainder of this paper is organized as follows. In the next section, related works are presented. Section III presents the proposed approach. In Section IV, a case study based on a part of AOCS is presented. Finally, conclusions are presented in Section V.

II. Related Works

Lettner and Tschernuth (2010) propose a MDA approach centered on a proprietary tool for embedded devices (mobile phones). They present the following considerations: (a) there is not a standard language for actions, which leads to the definition of proprietary languages; (b) lack of tools that give support MDA, such tools should be able to generate code based on an action language.

A study focused on defining transformations from model to model in the context of MDA is Zuo *et al.* (2010), where it is proposed a model transformation from PIM to PSM based on AADL (Architecture Analysis and Design Language). Action Specification Language (ASL) is the language used to define that transformation.

On the other hand, Feiler *et al.* (2007) use a custom UML profile for modeling PIMs using state machine diagrams and a proprietary language for actions. Afterwards, they transform PIM into PSM based on AADL using ATL (Atlas Transformation Language). From AADL, code is generated containing structure and behavior.

Chehade *et al.* (2011) show one way to reduce the cost of portability in MDA by providing domain-independent model transformations. Lin *et al.* (2011) present a framework and show how software code can be automatically generated from SysML models of multi-core embedded systems.

Buckl *et al.* (2010) focus on real-time embedded systems. They define and explore alternatives for specifying properties of time. The authors highlight that a modeling language should allow describe the system temporal constraints regardless of a specific solution. Furthermore, Buckl *et al.* (2010) explore and compare the MARTE (OMG, 2009c) with other alternatives for setting properties of time. Maes (2007) presents a way to transform UML models, instrumented with the MARTE, into AADL model using ATL. This paper checks non-functional properties based on an AADL model.

Wehrmeister (2009) prefers to model behavior using UML sequence diagrams. He also uses the GRM (Generic Resource Modeling) package of MARTE, in order to define schedulable resources, WCET (Worst Case Execution Time), and other properties at PIM. Moreover, PIM model is transformed into an intermediate model, and, eventually, code is generated. All transformations are defined using proprietary code.

During the specification of real-time software, Giese *et al.* (2010) state that functionality is developed regardless of the platform and its interfaces with the environment (A/D and D/A converters). Therefore, such models ignore properties such as WCET, hardware features, memory consumption or power. Focusing on the logical order of execution and data flow, this model allows verification and validation through simulation, using either no plant or a plant model as environment. The objective of this stage is to run a first proof of concept, and verification and validation of the overall project and control laws.

Giese *et al.* (2010), on page 43, also state that UML (OMG, 2009) as well as eight approaches suggested in the literature fail to provide an appropriate base for modeling real-time software since the suggested models are not sufficient to describe behavior of real-time system regardless of the platform. Giese *et al.* (2010) outline the work of Burmester *et al.* (2005) as an example that supports the modeling of real-time software using MDA. Burmester *et al.*

(2005) propose a profile called UML Mechatronic. They use UML components diagrams to define the interfaces between components (structure) and an extension of state machine diagrams to define behavior. According to the authors, an extension is necessary because the diagrams semantics of the state machine assumes that transitions take no time. Using the extension it is possible to specify deadlines for each transition.

Obermaisser and Kopetz (2009) define an architecture for embedded systems, which is a proposal for European reference architecture for embedded systems. In this work the MDA approach is present, using the concepts of PIM, PSM, and transformations. Another important point is that this study selected the MARTE UML profile (OMG, 2009c) as the common modeling language.

Lazar *et al.* (2009) introduce a fUML based action language and describe its concrete syntax; the action language uses only elements allowed by the fUML standard for its abstract syntax. Therefore, OMG works on ALF (Action Language for Foundational UML) (OMG, 2011) that is a specification to fUML concrete syntax.

The next section presents the detailed approach.

III. Proposed Approach

In a typical systems engineering process, this paper assumes that the overall architecture, system requirements and subsystems requirements are gathered, analyzed, detailed, and documented. These are the inputs to the specification phase when proposed approach starts. The specification phase is defined in Giese *et al.* (2010) as the phase where the software is decomposed into modules, defining their interfaces in a high level of abstraction where only functional properties are captured.

Therefore, proposed approach starts decomposing software into subsystems and defining their interfaces using signals. Afterwards, subsystem behavior is defined using fUML. Non-functional properties related with real-time are gathered and documented using MARTE during this phase. Using fUML, test scenarios are defined to exercise subsystems interface. Subsystems are weaved in a system. All is done using PIMs.

Independently, target platform is defined using a composition of PDMs and transformations.

With PIM and target platform, a processing is performed transforming PIM into PSM. PSM is transformed in AADL to schedulability analysis. Eventually, PSM is transformed in code.

Figure 1 illustrates the proposed approach, stating where the OMG specifications are used. Specifications used are: UML (OMG, 2009), fUML (OMG, 2009a), MARTE (OMG, 2009c), MOF QVT (OMG, 2008) and M2T MOF (OMG, 2008b).

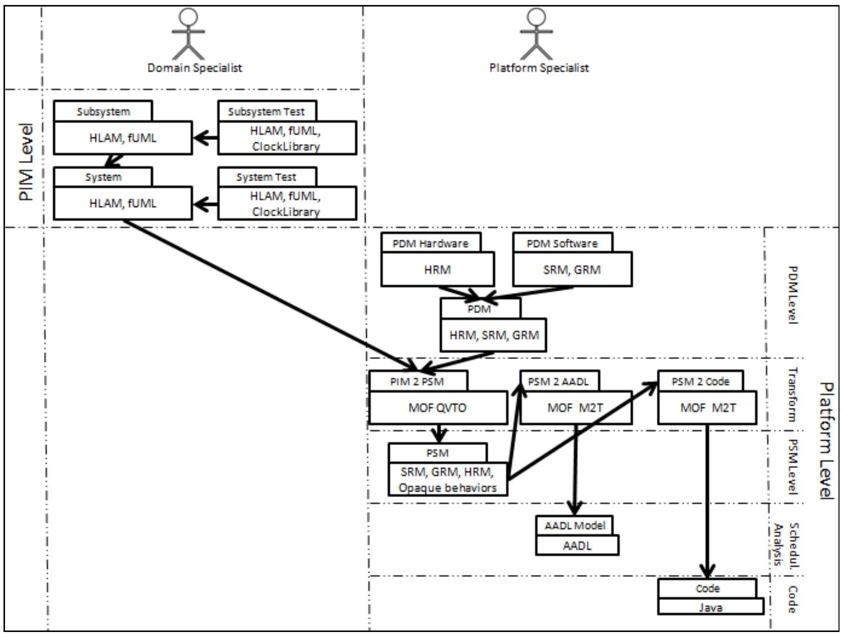


Figure 1. Proposed approach.

The following sections elaborate on the proposed approach, more details can be found in Romero (2010).

A. Platform Independent Model

The behavior of event-driven systems is better described when it is divided into a relatively small number of chunks, where event responses within each individual chunk indeed depend on only the current event-type but no longer on the sequence of past events (the context). UML (OMG, 2009) defines state machines diagram to model this kind of behavior. In UML, “chunks of behavior” are called states, and change of state is called a state transition. State machine diagram is essential for modeling state-dependent classes.

It is also necessary to model system structure.

Frequently, UML is used for: (a) model structure with class diagrams; (b) model behavior using two complementary diagrams, state machine diagrams, for state-dependent classes, and activity diagram, for describing algorithms and actions that occur within a given state.

However, behavior can only be defined using activities in fUML (OMG, 2009a). Due to this, each behavior must be described using activities in an activity diagram. States are described as structured activities (*StructuredActivity*), and transitions as actions of signal reception (*AcceptEventAction*). This alternative is aligned with the most appropriate strategy for modeling the behavior of reactive system, since the "chunks of behavior" are separated from transitions.

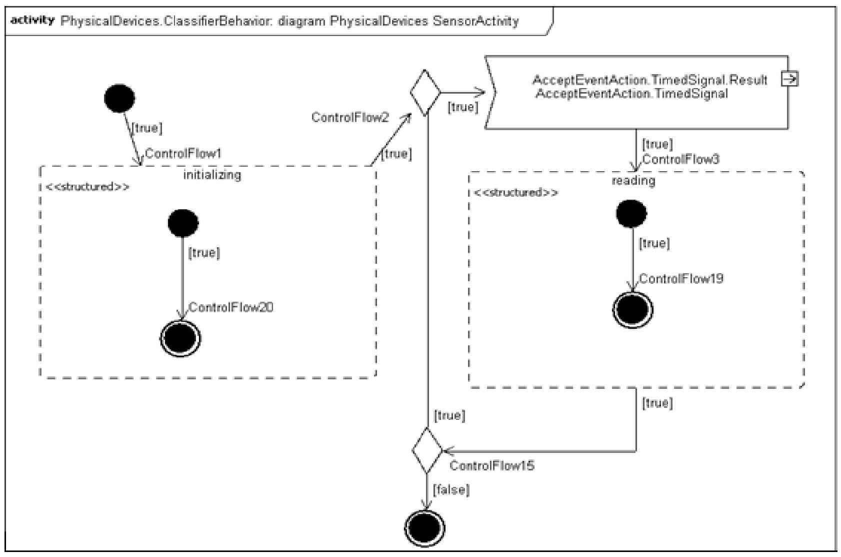


Figure 2. Used approach for activity diagrams.

According to the UML (OMG, 2009), all behavior in a modeled system is ultimately caused by actions performed by active objects. Active classes, which when instanced generate “active” objects, are the only kind of UML element allowed to be state-dependent classes by fUML. For each active class, it means, needs state management, this paper suggests an activity diagram as described in the preceding paragraph. It is not common operations in active classes, since its main purpose is to manage state. Algorithms are usually defined in passive classes.

An example of a UML activity diagram defined according to the approach described above is shown in Fig. 2. Since the behavior can be modeled with the approach described above, the structure is defined using class diagrams.

1. Non-functional properties

In PIM, real-time constraints are defined using HLAM (High-Level Application Modeling) package from MARTE (OMG, 2009c). HLAM defines several places where a real-time annotation can be defined in a UML model. This paper prefers to use annotations in real-time signals (UML *SignalEvent*), reinforcing the event-oriented approach. With this in mind, *RtFeature* stereotype is used to annotate *SignalEvents* with real-time features according to set of *RtSpecification.* This latter allows to define three essential attributes: (a) *occKind* - specifies the pattern of signal generation, for example, periodic, with period of 100 milliseconds and with uncertainty release (release jitter) from 0 microseconds - *(period = (100 ms), jitter = (0, ms))*; (b) miss - specifies the percentage of acceptance for missing the deadline, for example, for hard real-time deadlines *(0,%, max)*; (c) *relDl* - defines the relative deadline with reference to the instant of signal generation.

These non-functional properties are not analyzed in the PIM. The goal here is to have all functional and non-functional properties modeled in PIM.

2. Platform Independent Clock Library

The fUML does not restrict a single semantic to time. Thus, it enables various forms of time, including discrete time and continuous time. However, to model a real-time system and check it through the PIM, it is needed to define the time semantics. Aim at reaching this, the present paper defines a prototype library with fUML in the PIM level in order to define basic behaviors for a clock. The goal is to define a source and a mechanism for signal generation based on time. For purpose of this study, an ideal centralized clock is used to ensure platform independence and allow the simulation of time events.

The platform-independent time library (*ClockLibrary*) is comprised of a class called *GlobalIdealClock*. This class is annotated with *ClockType* stereotype from MARTE. In addition, it defines the operations *setTime* and *currentTime* using activities. Besides the two mentioned operations, a third operation was defined, called *waitTime*. This latter is responsible for increasing the time of the clock following a received parameter. Following this approach, the fUML assumes the hypothesis that the calculations and communications do not take time, i.e., the fUML is a synchronous language for modeling real-time systems.

3. Test cases

At this point, there is a PIM containing active classes that send and receive signals. There is a clock responsible for triggering timing signals and for providing information about current time. Thus, test scenarios can be modeled. As previously described, fUML only allows definition of behavior using activities. Therefore, test scenarios should be defined using activities as well.

In summary, it is possible to model PIMs, to model test scenarios, and to check these test scenarios. It is necessary to highlight that verification is focused only on functional properties. Non-functional property is not checked at PIM level, they will only be evaluated in platform level.

B. Target Platform

The following sections introduce the concepts and tasks necessary to define a target platform for real-time software.

1. PDM

In the present work, the PDM was subdivided into two complementary models, one that defines target hardware and another that characterizes target software.

The PDM Hardware defines hardware structure. It uses the package Hardware Resource Modeling (HRM) of the MARTE. This paper uses *ownedHw* property of the *hwProcessor* stereotype. Following the MARTE, *ownedHW* property specifies the elements of hardware maintained by *hwProcessor*. It is mandatory because it is used when AADL model is generated.

The PDM Software defines concepts of the target programming language. It represents the different kinds of parts and the services provided by it. It consists of classes and primitive types selected from target platform. For example, Java: *java.lang.Object*, *java.lang.Thread*, etc... The PDM software uses the GRM (Generic Resource Modeling) package of MARTE. If target programming language uses asynchronous approach to model real-time systems, which implies schedulability analysis, a Scheduler must be defined in PDM – Software. GRM offers a stereotype *Scheduler*. It has two important attributes: *isPreemptible* and *schedPolicy* (Scheduler Policy).

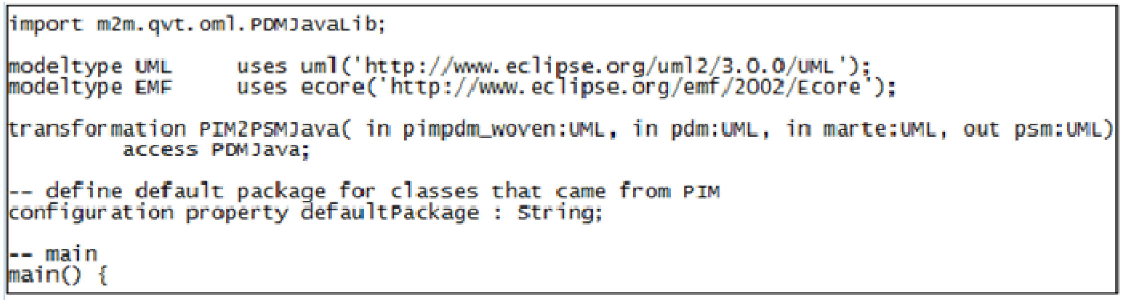
2. PIM to PSM Transformation

This paper opts for explicit mapping, where an intermediate model, containing only the concepts provided by the target platform is introduced. PSM is this model in MDA. In current paper, the specification MOF QVT is used one or more times to generate a PSM from PIM, taking into account a PDM. Afterwards, the resulting PSM is transformed into code, or other artifact using the specification MOF M2T. The following sections present each of the transformations.

a. MOF QVTO Transformation

This transformation is responsible for generating the PSM model. It defines the entire structure of the system and calls the *blackbox library* for behavior generation. It is important highlight that the transformation from PIM to PSM is dependent on the PDM because it is responsible for translating platform-independent concepts used in the PIM to platform dependent concepts defined in PDM. For example, this transformation is responsible for transforming active classes in classes that extend the *java.lang.Thread*, considering the Java platform. It is clear that a radical change in the PDM leads to a complete review of the transformation from PIM to PSM, i.e., this transformation complements a PDM forming a platform.

The OMG provides MOF QVTO specification in order to transform models into models. The use of the MOF QVTO to transform PIM to PSM, taking into account real-time software, has a similar code as shown in Fig. 3.



```
import m2m.qvt.oml.PDMJavaLib;

modeltype UML     uses uml('http://www.eclipse.org/uml2/3.0.0/UML');
modeltype EMF     uses ecore('http://www.eclipse.org/emf/2002/Ecore');

transformation PIM2PSMJava( in pimpdm_woven:UML, in pdm:UML, in marte:UML, out psm:UML)
        access PDMJava;

-- define default package for classes that came from PIM
configuration property defaultPackage : String;

-- main
main() {
```

Figure 3. Transformation signature PIM to PSM.

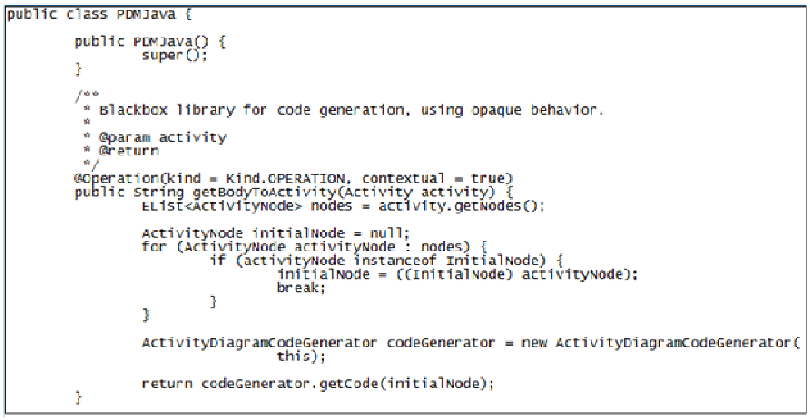
MOF QVTO transformation from PIM to PSM uses four key elements:

- PIM Meta-model defines the format of the PIM to be received, in the present work the UML meta-model;
- PDM specifies the characteristics of the target platform, it is directly and intensively used by the rules of this transformation;
- *OpaqueBehaviors* are used to store the behavior code at PSM (created by a *blackbox library*);
- SRM package from MARTE is used to annotate the elements of software with real-time features.

b. Blackbox library

It is responsible for behavior generation to the target platform. This is done analyzing the activities defined using fUML. There are some types of transformation that are hardly implemented using the formalism of MOF QVT specification, for this cases, the mapping called *blackbox* is recommended (OMG, 2008). The analysis of fUML activity diagrams and code generation for a target platform are examples of this kind of transformation.

Figure 4 displays a used method in a *blackbox library*. In the Eclipse Modeling Galileo (Eclipse Foundation, 2011), a *blackbox library* is defined as a plug-in.



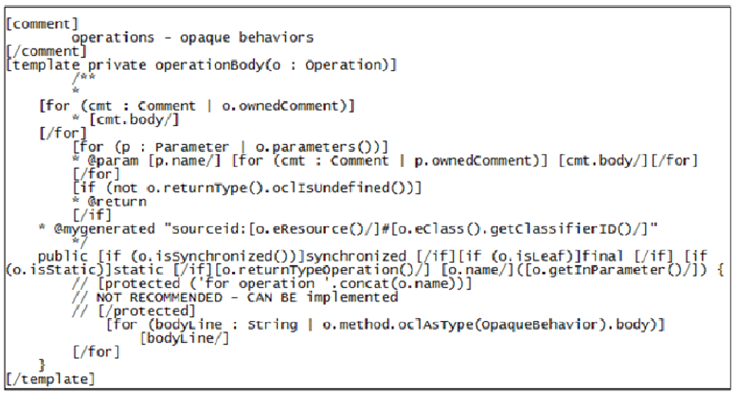
```java
public class PDMJava {

        public PDMJava() {
                super();
        }

        /**
         * Blackbox library for code generation, using opaque behavior.
         *
         * @param activity
         * @return
         */
        @Operation(kind = Kind.OPERATION, contextual = true)
        public String getBodyToActivity(Activity activity) {
                EList<ActivityNode> nodes = activity.getNodes();

                ActivityNode initialNode = null;
                for (ActivityNode activityNode : nodes) {
                        if (activityNode instanceof InitialNode) {
                                initialNode = ((InitialNode) activityNode);
                                break;
                        }
                }

                ActivityDiagramCodeGenerator codeGenerator = new ActivityDiagramCodeGenerator(
                                this);

                return codeGenerator.getCode(initialNode);
        }
```

Figure 4. *Blackbox library* operations.

It is important highlight that the *getBodyToActivity* method is a MOF QVTO operation (*@Operation*). It is contextual, i.e., it defines a new operation in the meta-class *UML Activity*. It returns a string containing the code for the target platform. This operation receives starting node of an activity diagram, which is evaluated to generate the code for the target platform.

3. PSM to Code Transformation

The transformation of the PSM to code is very simple because its responsibility is to translate the PSM model into a set of files in the format defined by the target language. It is defined using MOF M2T. Figure 6 displays a code snippet where a method is being generated in the Java language based on operations in PSM.



```
[comment]
        operations - opaque behaviors
[/comment]
[template private operationBody(o : Operation)]
        /**
        *
    [for (cmt : Comment | o.ownedComment)]
        * [cmt.body/]
    [/for]
        [for (p : Parameter | o.parameters())]
        * @param [p.name/] [for (cmt : Comment | p.ownedComment)] [cmt.body/][/for]
        [/for]
        [if (not o.returnType().oclIsUndefined())]
        * @return
        [/if]
    * @mygenerated "sourceid:[o.eResource()/]#[o.eClass().getClassifierID()/]"
        */
    public [if (o.isSynchronized())]synchronized [/if][if (o.isLeaf)]final [/if] [if
(o.isStatic)]static [/if][o.returnTypeOperation()/] [o.name/]([o.getInParameter()/]) {
        // [protected ('for operation '.concat(o.name))]
        // NOT RECOMMENDED - CAN BE implemented
        // [/protected]
            [for (bodyLine : String | o.method.oclAsType(OpaqueBehavior).body)]
                [bodyLine/]
        [/for]
    }
[/template]
```

Figure 5. PSM to Code transformation.

4. Schedulability Analysis

Once the PIM model was checked for its functional properties, a key issue, in real-time software, is to evaluate whether its tasks (threads) are scalable. To perform this verification key, this paper proposes to evaluate the PSM, once it has exactly the structure and behavior of software on the target platform and such information are key factors in this type of analysis.

However, this paper has not explored techniques or tools to determine the WCET, which is essential for the schedulability analysis. Thus, an expert in software should complement the PSM with additional information, e.g., the definition of mandatory activation time (WCET) of a class annotated automatically with the stereotype *SwSchedulableResource*. The present work suggests the use of historical information or evaluations of small pieces of code on the target platform to obtain this information.

Since there is a PSM model, and this was complemented with information about the activation time of each element which contains the stereotype *SwSchedulableResource*. It is possible to perform schedulability analysis. This paper has chosen to transform the PSM into an AADL model to allow the assessment of schedulability in any tool able to interpret AADL models. To generate an AADL model a MOF M2T transformation is applied on PSM. This transformation is simple, since the PSM contains all information necessary for generating the AADL, standing out period, time and activation time for each task as well as shared objects between them.

The next section presents the case study used to evaluate the proposed approach.

IV. Case Study

A case study was developed to evaluate the proposed approach, and selected tools. Specifications, models, and transformations discussed above were applied to part of the attitude control of the Multi-Mission Platform (MMP) from INPE (National Institute for Space Research) in nominal mode presented by Moreira (2006). The modeled and verified part was the sensor reading. Sensor reading subsystem encompasses: collect measures, computation of a simple transfer function, and sending results to other system modules. This module was selected by combining algorithmic simplicity (only a simple transfer function), and coverage (it would be possible to exercise all elements addressed in the current article). The target platform selected for case study was Java language, and part of characteristics of MMP. It should be noted that case study is not intended to generate code applicable to MMP but it assesses whether the proposed approach is feasible.

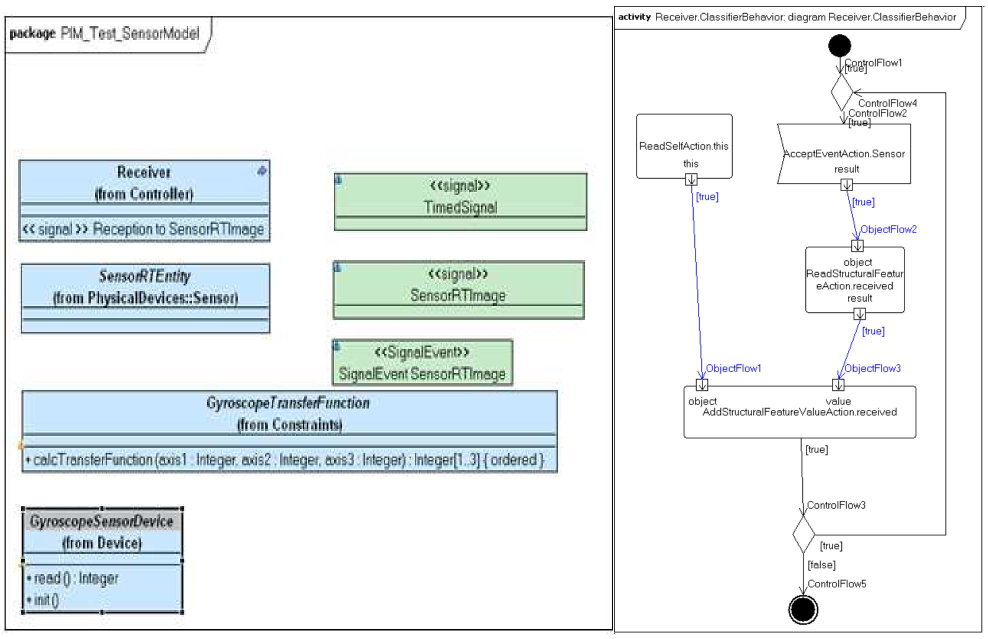
In order to allow comparison of the proposed approach, the same system using the same level of detail was modeled using an approach where all information is defined in a single model containing characteristics of the target platform. This model is called detailed design model.

A. Modeling according proposed approach

As proposed by the approach outlined above, case study began with two independent activities: definition and verification of platform-independent model, and target platform definition.

1. Platform Independent Model

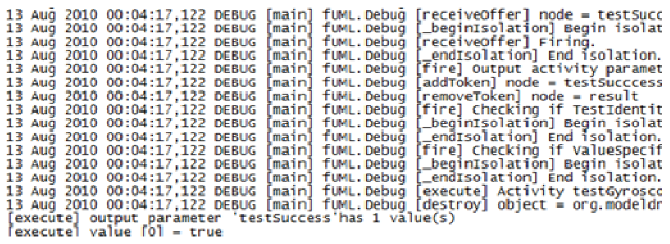
First, the subsystem structure of the sensor reading was defined. Figure 6.a shows part of this structure. Afterwards, behaviors and the test scenarios were modeled using fUML for this subsystem. Figure 6.b presents a behavior defined for verification purpose.


package PIM_Test_SensorModel
Receiver (from Controller)
<< signal >> Reception to SensorRTImage
<<signal>> TimedSignal
SensorRTEntity (from PhysicalDevices::Sensor)
<<signal>> SensorRTImage
<<SignalEvent>> SignalEvent SensorRTImage
GyroscopeTransferFunction (from Constraints)
+ calcTransferFunction (axis1 : Integer, axis2 : Integer, axis3 : Integer) : Integer[1..3] { ordered }
GyroscopeSensorDevice (from Device)
+ read () : Integer
+ init ()
activity Receiver.ClassifierBehavior: diagram Receiver.ClassifierBehavior
ControlFlow1
[true]
ControlFlow4
ControlFlow2
ReadSelfAction.this
this
AcceptEventAction.Sensor
result
ObjectFlow2
object ReadStructuralFeatureAction.received
result
ObjectFlow1
ObjectFlow3
object
value
AddStructuralFeatureValueAction.received
ControlFlow3
[false]
ControlFlow5

(a) (b)

Figure 6. PIM – (a) Structure; (b) Behavior.

Moreover, test scenarios were executed to verify the expected functional properties. Figure 7 shows execution results for a test scenario, *testGyroscopeSensorDevice*.



```
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [receiveOffer] node = testSucc
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_beginIsolation] Begin isolat
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [receiveOffer] Firing.
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_endIsolation] End isolation.
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [fire] Output activity paramet
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [addToken] node = testSucccess
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [removeToken] node = result
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [fire] Checking if TestIdentit
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_beginIsolation] Begin isolat
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_endIsolation] End isolation.
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [fire] Checking if ValueSpecif
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_beginIsolation] Begin isolat
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [_endIsolation] End isolation.
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [execute] Activity testGyrosco
13 Aug 2010 00:04:17,122 DEBUG [main] fUML.Debug [destroy] object = org.modeldr
[execute] output parameter 'testSuccess'has 1 value(s)
[execute] value [0] = true
```

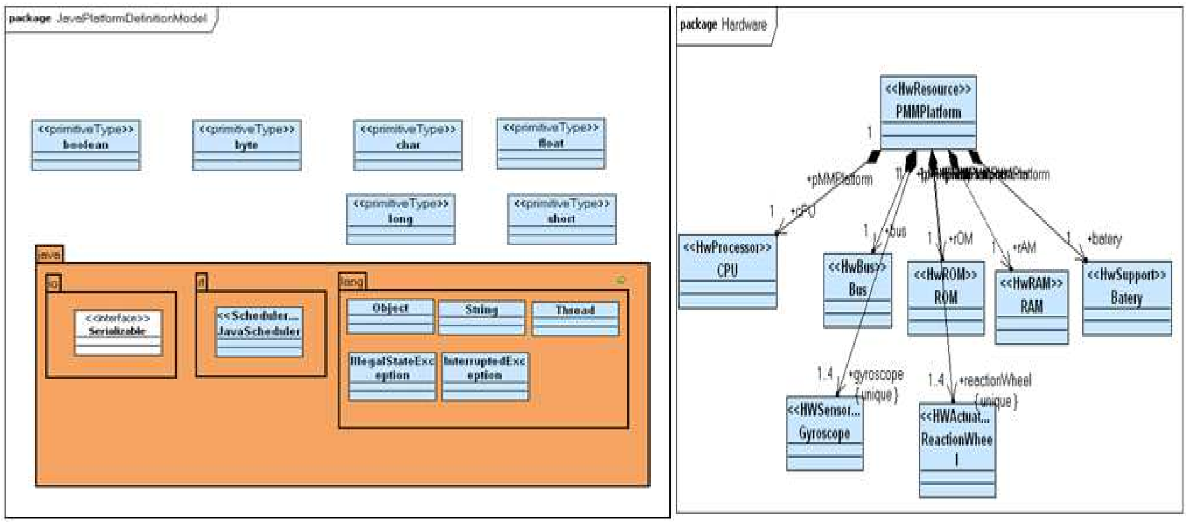
Figure 7. Platform independent test cases execution

At this point, PIM for reading sensor subsystem was modeled. This model was functionally verified through test scenarios execution. All test scenarios were executed using fUML- Reference Implementation 0.4 (ModelDriven, 2009). It is important to notice that non-functional properties were defined in this model, but they have not been verified since they do not belong to the functional domain.

2. *Target Platform Definition*

Concurrently, a target platform was defined following previous described characteristics.

This activity developed two models: (a) *PDM - Software* containing the essential part of the Java language that it is required for the transformation from PIM to PSM; (b) PDM - Hardware containing a description of some MMP real characteristics. Figure 8.a and 8.b shows part of these two models.



(a) (b)

Figure 8. (a) PDM – Software; (b) PDM – Hardware.

Afterwards, transformations were defined:

- m2m – model to model, it transforms the structure of a PIM (following the UML meta-model) to PSM (considering PDM for Java language as a target software platform);
- *blackbox library* – it is triggered by m2m transformation aim at transforming fUML activity diagrams into behavior in Java. Results are stored in PSM;
- m2text – responsible for transforming the PSM model into Java code.

3. Using target platform

At this point, we had target platform defined so it is possible to transform PIM (following the meta-model expected by the target platform) into PSM; eventually, code and AADL model can be generated.

Transformations from PIM to PSM and PSM to code were executed successfully. Figure 9 shows size of models through evolution of proposed approach. It is possible to see how big the PSM is compared to PIM. Indeed, this is due to transformation from PIM to PSM.

The transformation from PSM to AADL was performed, which allowed schedulability analysis by using the automatic evaluation of the model AADL with Cheddar (LISyC, 2010). This analysis indicated that the PSM model was scalable. As a result, generated code (LOC = 1255) could be used with indications of functional and temporal correction.

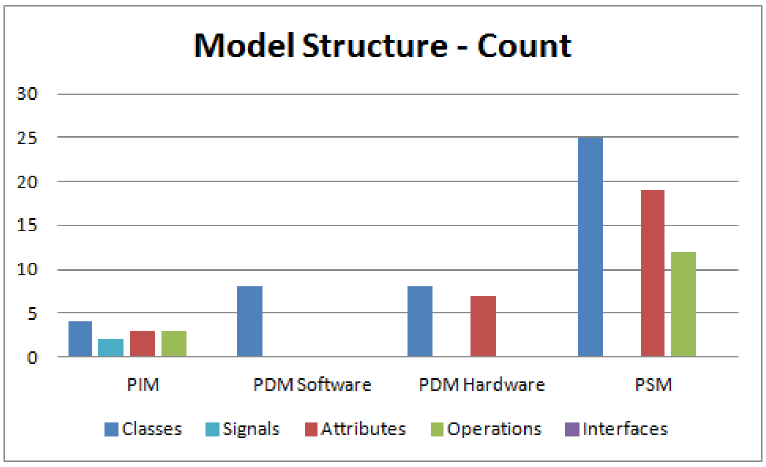


Figure 9. Model structure size evolution through proposed approach

B. Modeling using a detailed design model

Usually, real-time software requires high level of reliability. One used technique is to design it using some graphical notation, for example, UML. This is done in order to express some level of technical details. In line with this point and to allow comparison between approaches, the same system using the same level of detail was modeled using an approach where all information (detailed structure and behavior) are defined in a single model containing characteristics of target platform. To allow a comparative view of approaches, behavior was modeled using activity diagrams, considering only actions provided by fUML.

fUML was not created for this type of detailed modeling but it was used to allow the comparison. It is important to notice that the manual translation of activity diagrams to code can cause errors and requires an unnecessary effort from the point of view of proposed approach. Another alternative would be to use a tool that is able to transform activity diagrams into target language.

V. Conclusion

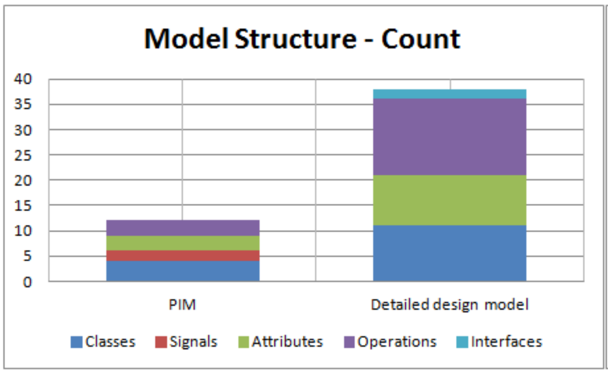
During case study, it has been possible to exercise the feasibility of the proposed approach. An important point of this work is its independence from commercial tools since the proposed approach is based on vendor neutral specifications. Another important point is that the case study used only open source tools, namely:

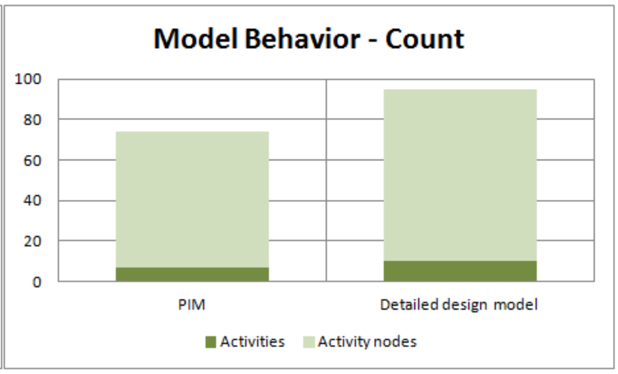
- TOCASED 3.3 (TOPCASED, 2011) – for UML;
- Eclipse Galileo Modeling (Eclipse Foundation, 2011) – to develop and to execute model to model and model to text transformations;
- fUML Reference Implementation 0.4 (ModelDriven, 2009) – to run PIM models defined using fUML;
- Cheddar 2.1 (LISyC, 2010) – to perform the schedulability analysis for an AADL model;

Approaches were compared considering the work of Monperrus *et al.* (2007) so a set of metrics for counting were selected, as follow:

- Structure: classes count, amount of signals (existing only in PIM), attributes count, operations count, and interfaces count (it only exists in the detailed model of the project due to the fact that fUML exclude *Interfaces* and target platform does not use them).
- Behavior: activities count, and activity nodes count (*ActivityNode*) required to model the behavior.

Figure 10 presents results for measures defined for the structure (10.a) and behavior (10.b). From data in Fig. 10, there is indication that proposed approach is complete, and it reduces size of model manipulated by users. It is complete because it defines the structure, behavior and non-functional properties of time on a single high level model. Considering PIM as reference, PIM structure is 175% lower with respect to count classes; moreover, it reduces operations count and attributes count. Regarding the behavior, reduction is less significant but it is still considerable. Activities count is 42% lower in PIM.





(a) (b)

Figure 10. Comparison between approaches; (a) Structure; (b) Behavior.

From our point of view, the proposed approach is advantageous whether we evaluate together with safety-critical software requirements that requires in detail structure and behavior modeling. Moreover, the approach still has the following benefits: (a) alignment indicated (without manual intervention) by the transformations between the PIM and PSM; (b) evaluation of non-functional properties based on models (schedulability); reuse of models and transformations; definition and use of hardware aspects.

In conclusion, this paper proposes a vendor independent approach that uses:

- PIM defined with fUML and MARTE (HLAM) – it allows modeling of structure, behavior and time requirements;
- PDM - Hardware and PDM - Software defined with MARTE (GRM, HRM e SRM) – they allow modeling of hardware elements and characterization of the target platform;
- MOF QVTO – it performs the transformation of a PIM into a PSM model considering the PDMs, evaluating fUML to generate behavior on the target platform;
- MOF M2T – it generates code for the target platform and an AADL model for schedulability analysis.

Future studies are related with the weaknesses of the current proposal. The most important are:

- It is difficult to model behavior using fUML through activity diagrams. They are very extensive and complicated. It is necessary to integrate the ALF (Foundational Action Language for UML), a textual language for concrete actions defined by the OMG (OMG, 2011);
- It is difficult to model continuous systems. Control algorithms are more complex when they are modeled using fUML. It is necessary to incorporate some type of value specification language (VSL), as defined in MARTE;
- Test scenarios in PIM are manually defined by domain experts focused on functional model; therefore, this creates the possibility of uncovered test scenarios. It is necessary the integration of the formal mechanisms to PIM model verification. Another concern is to use a UML Testing profile (OMG, 2012) to define them.
- It is difficult to ensure transformation correctness so it is important to evaluate alternatives to certification.
- Target platform definition requires high efforts. It is necessary to evaluate alternatives to parameterize transformations in order to increase your chance of reuse. An alternative is work of Chehade *et al.* (2011).

We believe that presented approach gives an important contribution to the definition of real-time software, since it has a well-defined path to: (a) model and test PIMs placing emphasis in model size reduction, interoperability, portability, and reuse; (b) indicated alignment and reuse of transformations; (c) evaluation of non-functional properties (schedulability). These points contribute to the certification process. Moreover, it allows extensions as listed as future work without invalidating the approach proposed here.

This work is part of Brazilian National Institute for Space Research (INPE) research about modeling and verifying real-time software, which the main focus is to obtain a better balance between dependability, schedule, and cost. As it was stated by OMG (2003), MDA is a key initiative to reduce costs while it can increase dependability. Results presented in this paper reinforce this potential.

References

Buckl, C.; Gaponova, I.; Geisenger, M.; Knoll, A.; Lee, E. A. (2010). Model-Based Specification of Timing Requirements. In Proceedings… EMSOFT 2010 Proceedings of the tenth ACM international conference on Embedded software.

Burmester, S.; Giese, H.; Schafer,W.; (2005). Model-driven architecture for hard real-time systems: From platform independent models to code. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 25–40. Springer, Heidelberg (2005).

Chehade, W. E. H.; Radermacher, A.; Terrier, F.; Selicy, B.; Gerard, S. (2011). A Model-Driven Framework for the Development of Portable Real-time Embedded Systems. In proceedings… 2011 16th IEEE International Conference on Engineering of Complex Computer Systems.

Eclipse Foundation (2011). Eclipse Site. Available at: <http://www.eclipse.org>. Accessed on: 29 June 2011.

Feiler, P.; Niz, D.; Raistrick, C.; Lewis, B. (2007). From PIMs to PSMs. In Proceedings... IEEE International Conference on Engineering Complex Computer Systems, n. 23, 2007, Auckland, New Zealand.

Giese, H.; Karsai, G.; Lee, E.A.; Rumpe, B.; Schätz, B. (2010). Model-Based Engineering of Embedded Real-Time Systems. International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. ISBN 978-3-642-16276-3

Lazar, C.L.; Lazar, I.; Parv, B.; Motogna, S.; Czibula, G. (2009). Using a fUML Action Language to construct UML models . In proceedings… 2009 11th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing.

Lettner, M.; Tschernuth, M. (2010). Applied MDA for Embedded Devices: Software design and code generation for alow-cost mobile phone. In proceedings… 2010 34th Annual IEEE Computer Software and Applications Conference Workshops.

Lin, Chao-Sheng; Lu, Chun-Hsien; Lin, Shang-Wei; Chen, Yean-Ru; Hsiung, Pao-Ann. (2011) VERTAF/Multi-Core: A SysML-based application framework for multi-core embedded software development. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 26(3): 448–462 May 2011. DOI 0.1007/s11390-011-1146-3.

LISyC (2010). Cheddar Site. LISyC, Université de Brest. 2010. Available at: <http://beru.univ-brest.fr/~singhoff/cheddar/index-fr.html>. Accessed on: 29 June 2011.

Maes, E. (2007). Validation de systèmes temps-réel et embarqué à partir d'un modèle MARTE. France: Thales Research and Technology, 2007. 24 p.

ModelDriven. (2009). fUML Reference Implementation Site. ModelDriven. Available at: <http://www.modeldriven.org>. Accessed on: 29 June 2011.

Monperrus, M.; Champeau, J.; Hoeltzener, B. (2007). Counts count. In proceedings… 2nd workshop on Model Size Metrics Co-located with MODELS 2007 1 October 2007 Nashville, USA.

Moreira M. L. B. (2006). Design and simulation of a discrete controller for the Multi-Mission Platform and its migration to a real-time operational system. 2006. 181 p. (INPE-14202-TDI/1103). Master dissertation (Space Engineering and Technology-ETE, Option Space Mechanics and Control -CSE) - National Institute for Space Research (INPE), São José dos Campos, Brazil, 2006.

Obermaisser, R.; Kopetz, H. (2009). Genesys – A candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems. 2009. Available at: <http://www.genesys-platform.eu/genesys_book.pdf> Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2003). Model-Driven Architecture. USA: OMG, 2003. 62 p. Available at: <http://www.omg.org/mda>. Accessed on: 17 May 2009.

OBJECT MANAGEMENT GROUP (OMG) (2008). Meta Object Facility (MOF 2.0) Query/View/Transformation Specification: Version 1.0. USA: OMG, 2008. 240 p. Available at: <http://www.omg.org/spec/QVT/1.0/>. Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2008b). MOF Model to Text Transformation Language: Version 1.0. USA: OMG, 2008. 48 p. Available at: <http://www.omg.org/spec/MOFM2T/1.0/>. Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2009). Unified Modeling Language, Superstructure: Version 2.2. USA: OMG, 2009. 740 p. Available at: <http://www.uml.org/>. Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2009a). Semantics of a Foundational Subset for Executable UML Models: Version FTF Beta 2. USA: OMG, 2009. 349 p. Available at: <http://www.omg.org/spec/FUML/>. Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2009c). UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems: Version 1. USA: OMG, 2009. 738 p. Available at: <http://www.omgmarte.org/>. Accessed on: 17 May 2011.

OBJECT MANAGEMENT GROUP (OMG) (2011). Concrete Syntax for UML Action Language (Action Language for Foundational UML - ALF): Version: 2nd FTF. USA: OMG, 2011. Available at: <http://www.omg.org/spec/ALF/>. Accessed on: 24 April 2012.

OBJECT MANAGEMENT GROUP (OMG) (2012). UML Testing profile: Version: 1.1. USA: OMG, 2012. Available at: < http://www.omg.org/spec/UTP/1.1/PDF/>. Accessed on: 24 April 2012.

Romero, A. G. (2010). An approach to model-driven architecture applied to space embedded real-time software. Version: 2010-12-13. 203 p. Master dissertation (Space Engineering and Technology-ETE, Option Engineering and Management of Space Systems-CSE) - National Institute for Space Research (INPE), São José dos Campos, Brazil, 2010.

TOPCASED (2011). TOPCASED Site. Available at: <http://www.topcased.org>. Accessed on: 29 June 2011.

Wehrmeister, M. A. (2009). An aspect-oriented model-driven engineering approach for distributed embedded real-time systems. 2009. 206 p. Doctoral thesis – Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil. 2009.

Zuo, W.; Feng, J.; Zhang, J. (2010). Model Transformation from xUML PIMs to AADL PSMs. In proceedings… 2010 International Conference on Computing, Control and Industrial Engineering.