Modeling of Embedded Software on MDA Platform Models

Inali Wisniewski Soares^{1,2}, Luciane Telinski Wiedermann Agner^{1,2}, Paulo Cézar Stadzisz^{2}, Jean Marcelo Simão^{2}
^{1}Mid-West State University, DECOMP-UNICENTRO, Brazil
^{2}Federal University of Technology Paraná, CPGEI – UTFPR, Brazil
Email-id: inali@unicentro.br, lagner@unicentro.br, stadzisz@utfpr.edu.br, jeansimao@utfpr.edu.br

ABSTRACT
This study proposes the use of abstract software models in order to meet the diversity of embedded platforms. A UML 2.0 Profile for Modeling Application and Platform of Embedded Software (called PROAPES) is proposed. Such profile is intended to generically describe the services provided by a system platform that makes use of an RTOS. In addition, this study presents a Model Transformation (MT) based on the PROAPES profile, named MT-PROAPES. In this way, MT-PROAPES uses a Platform Model (PM), created on the basis of the proposed profile (PROAPES), and performs a transformation named Platform Independent Model (PIM)-behavior into Platform Specific Model (PSM)-behavior. Thus, the generation of reusable model transformations that are adaptable to different platform models is possible.

Keywords: Model Driven Architecture, Platform Model, UML Profile, Model Transformation, Real-Time Operating Systems.

1. INTRODUCTION

An embedded system can be defined as “any device that includes a programmable computer but is not itself intended to be a general-purpose computer” [1]. Currently, the growing need for new embedded products with additional functionalities is a tendency, thereby demanding the increment of more complex software components in the system. Therefore, the complexity of embedded software systems emphasizes the need for high-level development approaches, such as Model Driven Architecture (MDA).

MDA is an approach for software development proposed by the Object Management Group (OMG). In the MDA context, each software artifact is considered a model or a model element. MDA thus emphasizes the role of models as the primary development artifacts by providing a set of guidelines for model definition, as well as the transformations between such models [2].

One of the MDA principles is founded on the separation between the software architecture knowledge and the knowledge concerning the platform features needed for the software implementation. The goal of such approach consists in producing software assets that are more resilient to changes caused by emerging technologies [3].

The MDA development approach involves the following steps: 1) Specification of a Platform Independent Model (PIM); 2) Specification of a Platform Model (PM); 3) Selection of a specific platform for the system; and 4) Transformation of a PIM into a Platform Specific Model (PSM), based on a specific platform [2].

The term “platform” refers to any set of hardware or software mechanisms that enable the execution of software applications [4]. Throughout this paper, however, the term “platform” denotes Real-Time Operating Systems (RTOS) and their respective hardware platforms of embedded systems, based on specific processors required for their execution.

The PM provides a set of technical concepts that represents parts and services of a platform. In its turn, Model Transformation (MT) can be defined as the generation of a target model from a source model according to a set of rules that defines the link between their elements [2]. Within the scope of this paper, MT performs the PIM-into-PSM model transformation based on a specific PM. The transformation maps out the elements of a source model to elements of a target model, in accordance with the chosen PM.

Currently, the support offered by MDA for the development of embedded software, particularly for RTOS-based software, is still limited. MDA primarily focuses on middleware target platforms, such as EJB, Web Services, .NET, and CORBA [2]. These platforms allow applications to be projected independently from the type of operating system.

However, a typical embedded system comprises some features like concurrency, real-time processing, and limited resources. Such requirements are commonly supported by an RTOS, whereas the use of middleware is much more likely to fail. As a result, many embedded applications are designed and implemented to run directly upon the RTOS [5].

Particularly, embedded systems are more affected by the adopted platform due to the incorporated hardware features, as well as the constraints toward those systems. In order to achieve “platform independence”, the platform used must be defined in an abstract way, considering its main interest attributes. This means that the software design must perform the application modeling based on an abstract PM [4].

In addition, there is a wide variety of hardware platforms that can be used in the development of RTOS-based embedded software. This happens because of the large number of suppliers and technologies available, given that a platform includes aspects related to the operating system and to the associated hardware.

The *X Real-Time Kernel* [6] is an example of RTOS, presented in this paper as part of the target platform. This kernel can be employed in different hardware platforms, identified according to the microprocessor used: ARM7TDMI, ARM9, Cortex-M4, among others.

In order to represent software development models, OMG has defined the Unified Modeling Language (UML) [7] as the standard language of the MDA [2]. UML supports the description of the structure and the behavior of systems by using different types of model elements and corresponding diagrams [8].

This paper presents the UML 2.0 Profile for Modeling Application and Platform of Embedded Software, named PROAPES, and the MT-PROAPES, a model transformation based on the PROAPES profile. PROAPES profile is composed of the following sub-profiles: *swxRTOS* – defines a set of stereotypes that enables annotations in the PM, indicating the RTOS services used in structural models (class diagram); and *dyxRTOS* – defines a set of stereotypes that enables annotations in the PIM, indicating the RTOS services used in behavioral models (sequence and activity diagrams).

This paper also presents a generic model transformation, named MT-PROAPES, which provides independence between the transformation rules and the platform features by means of the PROAPES profile. The MT-PROAPES defines a PIM-behavior into PSM-behavior model transformation. Both the PIM-behavior and the PM apply elements of the PROAPES, being explicitly defined and inserted as input parameters in the MT-PROAPES. As a result, the model transformation concerns are separated from the platform model concerns. The use of an explicitly defined PM enables the creation of transformations that are reusable in new platforms. Thus, the MT-PROAPES is reusable and adaptable to several embedded software platforms based on RTOS by means of PMs explicitly defined.

This paper is organized as follows: Section 2 presents the UML PROAPES profile; Section 3 presents the proposed MT-PROAPES; In turn, Section 4 shows an illustrative example of PROAPES application in the MT-PROAPES; Section 5 brings the related works, and finally, Section 6 provides conclusions and future works.

2. PROAPES PROFILE

PROAPES profile defines a set of stereotypes in order to abstractly describe the services provided by a platform that makes use of the RTOS and their respective hardware platforms of embedded systems, based on specific processors required for their execution. This profile is composed of the following sub-profiles: *swxRTOS* - used to describe PMs in structural models (i.e., class diagram); and *dyxRTOS* - used to describe PIMs in behavioral models (i.e., sequence and activity diagrams). The next sub-sections introduce the *swxRTOS* and *dyxRTOS* profiles.

2.1 swxRTOS Profile

The *swxRTOS* profile defines a set of stereotypes to describe platform models in structural models, such as the class diagram. This profile is composed of sub-profiles such as the *swxCoreRTOS*, which represents the basic concepts of the high-level constructs needed to support both concurrency and interactions. In its turn, the *ddxRTOS* is another example of a sub-profile of the *swxRTOS*, representing the concepts related to the physical microcontroller peripherals used in RTOS. Due to length limitation, in this section only fragments of the *swxCoreRTOS* and *ddxRTOS* sub-profiles will be considered and described in Tables 1 and 2, respectively.

Stereotype	**swxCore**
Description	concepts regarding the software description in concurrent execution contexts
Tagged Values	threadName: name of a task threadPriority: execution priority of a task timeSuspension: time of suspension sleepThreadFor: suspension of a task for a definite period activateThread: creation of a task threadStackSize: number of words of which the task stack is composed
Stereotype	**swxSemaphore**
Description	concepts regarding the creation and management of a semaphore
Tagged Values	num: semaphore initial value semaph: variable whose values are 0, 1 or negative checkActualVlr: checks the semaphore current value waitSemaphore: wait state of the semaphore
Stereotype	**swxTime**
Description	concepts regarding time values.
Tagged Values	nanosec: stores time related values in nanoseconds sec: stores time related values in seconds time: stores time related values getNanoSec: retrieves the current value of the nanosecond counter

Table 1. Stereotypes of the *swxCoreRTOS* sub-profile

Stereotype	**ddxDevice**
Description	represents general information regarding device drivers
Tagged Values	stateDD: represents the device driver state getStateDD: retrieves the device driver state startDD: starts a device after its configuration stopDD: stops a device
Stereotype	**ddxConfig**
Description	represents configuration information of device drivers
Tagged Values	numTypeDD: represents the device driver type code getNumTypeDD: retrieves the device driver type code
Stereotype	**ddxLCD**
Description	represents concepts regarding LCD device drivers
Tagged Values	line: display current line column: display current column message: group of characters to be displayed on the screen ddxSendCmd: sends a command to the display ddxPosic: sets the cursor according to the line-column positioning ddxWriteString: sends a group of characters to be displayed on the screen

Table 2. Stereotypes of the *ddxRTOS* sub-profile

2.2 dyxRTOS Profile

The *dyxRTOS* profile defines a set of stereotypes that are employed in a behavioral PIM defined in UML. Furthermore, these stereotypes are the link between the PIM and the PM. The «*dyxActionSwRTOS*» stereotype of the *dyxRTOS* profile represents the software general actions in RTOS software projects and extends the metaclasses: *Message* - which describes the messages exchanged between objects and entities involved in a system (used in Sequence Diagram); *OpaqueAction* – which is used to model the activities within a process (used in Activity Diagram). Tagged values of this stereotype are, in their turn, used to link the PM (annotated with the stereotypes of the *swxRTOS* profile) to the PIM (annotated with the stereotypes of the *dyxRTOS* profile). Such tagged values are described as follows:

- *opSwTarget*: indicates the property defined in the stereotype of the *swxRTOS* profile that corresponds to an action defined in the «*dyxActionSwRTOS*» stereotype.

- *paSwTarget*: indicates the properties defined in the stereotype of the *swxRTOS* profile that corresponds to the parameters defined in the «*dyxActionSwRTOS*» stereotype.
- *rtSwService*: indicates the profile name responsible for defining the abstract PM that will be linked to the PIM defined by the *dyxSwRTOS* profile.

Other stereotypes of the *dyxSwRTOS* profile, described in Table 3, are specializations of the «*dyxActionSwRTOS*» stereotype.

Stereotype	**Description**
dyxCheckMessage	message detection action in the inbox of a thread.
dyxInitSched	activation action of the scheduler.
dyxSendMessage	message sending action from a thread to another.
dyxReceiveMessage	message receiving action from a thread.
dyxActivateThread	task creation action.
dyxInitRTOS	initialization action of the internal structures of the RTOS.
dyxSleepThread	interruption action of a task for a definite length of time.
dyxGetIdThread	ID information action of a task.
dyxStartDD	device driver initialization.
dyxRegisterReceiverDD	receipt registration of the controlling thread ID
dyxSendCommandDD	sending the execution command to a device driver.
dyxPosicDD	setting the cursor at a specific line and column
dyxWriteStringDD	writing a string in the LCD display.

Table 3. Stereotypes of the *dyxSwRTOS* sub-profile

3. MT-PROAPES: PIM-BEHAVIOR INTO PSM-BEHAVIOR

This section presents a PIM-behavior into PSM-behavior model transformation that demonstrates the use and application of the PROAPES profile proposed in this paper. Such transformation makes use of an explicitly defined PM as input and performs a model refinement, which is a specific type of model transformation.

In a model refinement, a transformation copies most elements of the source model to the target model, whereas just some elements of the source model are supposed to be modified in order to set up the target model. Thus, the transformation preserves most parts of the source model [9].

The transformation proposed in this paper, called MT-PROAPES, was implemented in Atlas Transformation Language (ATL), a hybrid transformation language designed to express model transformations as required by model-driven approaches [10]. ATL is one of the most widely used model transformation languages, recognized as a standard solution for model transformations in the MDA context [11]. This language is an Eclipse Model-to-Model (M2M) component inside the Eclipse Modeling Project (EMP). Consequently, ATL consists of a set of tools built on top of the Eclipse platform.

ATL transformation consists of modules containing the transformation rules. In this solution, the UML2Copy.atl module, proposed by Wagelaar et al. [12], comprises the rules used to perform the copy of the PIM elements to the PSM. Refinement specific rules were implemented in the MT_PROAPES.atl module, carrying out necessary changes in the PIM elements by adding platform-specific details to it in order to generate the new elements of the PSM. Thus, the transformation rules of the UML2Copy.atl transformation module are reused and overridden, when necessary, by the MT_PROAPES.atl transformation module through the superimposition mechanism.

Superimposition is a composition technique for rule-based model transformation languages, e.g. ATL, that allows the composition of several transformations into one [13]. Through the employment of the superimposition technique in this research, it was possible to arrange the transformation in modules, namely: 1) UML2Copy.atl - module comprised of the rules for copying the PIM elements to the PSM; and 2) MT_PROAPES.atl - module comprised of the specific rules for creating the new PSM elements that contain platform-specific features.

The transformation proposed in this study was configured in the Eclipse environment. The models defined to set the transformation input are the PIM and the PM. The PSM model is defined to set the transformation output. The model defined to set the transformation output is the PSM model. The MT_PROAPES.atl module is detailed next. Further information on the UML2Copy.atl module can be found in [12].

3.1 MT_PROAPES.atl Module

The MT_PROAPES.atl module contains a header section, helpers, and transformation rules. The header section (Fig. 1) defines the MT_PROAPES.atl module in which source and target models are declared, as follows: 1) IN model - refers to the PIM source model; 2) PM model - contains the platform model; and 3) OUT model - refers to the PSM, and is created as a result of the transformation. PIM, PM, and PSM models conform to the UML2 metamodel, intended to be bound to the Eclipse UML2 metamodel. Currently, the UML metamodel is widely used and is supported by several development tools [14].

```
module MT_PROAPES; -- superimposed on
create OUT : UML2 from IN : UML2, PM : UML2;
```

Fig. 1. MT_PROAPES.atl module header

The MT_PROAPES.atl module contains several helpers used to specify some transformation features. A helper is a query function that defines operations regarding the OCL specification [15] in the context of source model elements. The context defines the element type to which the helper can be applied. In addition, a helper is called by another helper or by transformation rules. The helpers defined in the MT_PROAPES.atl module are the following: *getTagVal*, *getPMClass*, *isStereotype*, *isdyxCheckMessage*, *isdyxInitSched*, *isdyxSendMessage*, *isdyxReceiveMessage*, *isdyxActivateThread*, *isdyxInitRTOS*, *isdyxSleepThread*, *isdyxGetIdThread*, *isdyxStartDD*, *isdyxRegisterReceiverDD*, *isdyxSendCommandDD*, *isdyxGoToLineColDD*, and *isdyxWriteStringDisplayDD*. In order to make it clearer, some of these helpers are depicted next.

The *getTagVal* helper retrieves the value of a specific property associated with a stereotype applied to a model element. Therefore, it brings back the value of the stereotype and of its corresponding properties as parameters.

The *getPMClass* helper searches in the PM for a class containing the applied stereotype, i.e., set as parameter. The *isStereotype* helper checks the existence of stereotypes applied to a specific message of the PIM model.

Finally, the *isdyxSendMessage* helper checks the existence of the «*dyxSendMessage*» stereotype applied to a specific message of the PIM model.

The mentioned helpers are illustrated in Fig. 2. All other helpers starting with the prefix "is" have similar functions, i.e., checking whether a specific stereotype is applied to a model message.

ATL legacy tool is used for defining the MT_PROAPES.atl module, enabling a rule to inherit the features of another rule. In so doing, properties that are common to several rules may be factored into a base rule, or parent rule. Consequently, other rules can stem from a parent rule. Each deriving rule, or sub-rule, comprises the parent rule properties, together with specific properties attached to it. Legacy thus improves the re-usability of the developed encoding [16].

The MT_PROAPES.atl module implements specific rules for each type of stereotype applied to a message of the PIM model, in addition to rules that extend the *MessagedyxRTOS* rule. The *MessagedyxRTOS* rule is a parent rule that manipulates the general attributes of a stereotyped message, independently from the stereotype applied to it. In their turn, the following rules extend the *MessagedyxRTOS* rule and include specific properties that perform platform-dependent changes based on the stereotype applied to the message: *MessagedyxCheckMessage*, *MessagedyxInitSched*, *MessagedyxSendMessage*, *MessagedyxReceiveMessage*, *MessagedyxActivateThread*, *MessagedyxInitRTOS*, *MessagedyxSleepThreadTime*, *MessagedyxGetIdThread*, *MessagedyxStartDD*, *MessagedyxRegisterReceiverDD*, *MessagedyxSendCommandDD*, *MessagedyxGoToLineColDD* and *MessagedyxWriteStringDisplayDD*.

Fig. 3 brings the *MessagedyxRTOS* parent rule, which is executed in the context of stereotyped messages belonging to the source model. This rule copies the message attributes that are platform independent, i.e., that do not change according to the adopted platform. The *MessagedyxRTOS* rule is defined as an abstract rule, extended by other rules of the MT_PROAPES.atl module by adding new properties to it. An abstract rule is a rule that provides a basic behavior that is used by rules that extend it.

```
helper context UML2!Element def: getTagVal(sName:String,
   pName:String): String =
   self.getValue(self.getAppliedStereotypes() ->
   select(e | e.name = sName).first(), pName);


helper def : getPMClass(sName : String) : UML2!Class =
   UML2!Class.allInstancesFrom('PM')->select(c |
   c.getAppliedStereotypes()->
   exists(st|st.name = sName)).first();


helper context UML2!Message def : isStereotype() : Boolean =
   self.getAppliedStereotypes()-> notEmpty();


helper context UML2!Message def : isdyxSendMessage () :
   Boolean =
   self.getAppliedStereotypes()->
   exists(st|st.name='dyxSendMessage ');
```

Fig. 2. Some helpers used in the MT_PROAPES

```
abstract rule MessagedyxRTOS {
   from s : UML2!"uml::Message" (
   if thisModule.inElements->includes(s) and
   s.isStereotype()
   then
      s->oclIsTypeOf(UML2!"uml::Message")
   else false endif)
   to t : UML2!"uml::Message" mapsTo s (
      visibility <- s.visibility,
      messageSort <- s.messageSort,
      eAnnotations <- s.eAnnotations,
      ownedComment <- s.ownedComment,
      clientDependency <- s.clientDependency,
      nameExpression <- s.nameExpression,
      receiveEvent <- s.receiveEvent,
      sendEvent <- s.sendEvent,
      connector <- s.connector,
      argument <- s.argument)
}
```

Fig. 3. *MessagedyxRTOS* rule

Fig. 4 illustrates the *MessagedyxSendMessage* rule, valid only in the context of messages to which the «*dyxSendMessage*» stereotype is applied. Such rule is depicted in the four main steps below:

1. Makes use of the *getPMClass* helper to search in the PM for a class annotated with a stereotype of the *swxRTOS* profile, and whose name is the same as the value of the tagged value named *rtSwService*.
2. Next, uses the *getTagVal* helper to search in the PM annotated with a stereotype of the *swxRTOS* profile for an operation (service) whose name is the same as the value of the tagged value named *opSwTarget*.
3. After locating the operation (service) in the PM, it is replaced with the action (service) abstractly defined in the PIM.
4. Then, the parameters defined in the *paramSwTarget* tagged value are incorporated to the new operation. The tagged values *rtSwService*, *opSwTarget*, and *paramSwTarget* are defined in the «*dyxActionSwRTOS*» stereotype, bearing in mind that the «*dyxSendMessage*» is a specialization of the «*dyxActionSwRTOS*».

For instance, as the transformation is performed, the PIM properties annotated with the stereotypes of the *dyxRTOS* profile are replaced with properties defined in a specific PM annotated with the stereotypes of the *swxRTOS* profile. Thus, platform-specific models are generated. The other rules mentioned work similarly, performing the appropriate adjustments in the variables of their specific domain.

```
rule MessagedyxSendMessage extends MessagedyxRTOS {
   from s : UML2!"uml::Message" (
   if thisModule.inElements->includes(s) and
   s.isdyxSendMessage()
   then
   s->oclIsTypeOf(UML2!"uml::Message")
   else false endif)
   to t : UML2!"uml::Message" mapsTo s (
      name <- thisModule.
      getPMClass(s.getTagVal('dyxSendMessage',
      'rtSwService')).getTagVal(s.getTagVal('dyxSendMessage',
      'rtSwService'),s.getTagVal('dyxSendMessage',
      'opSwTarget')).name
      + '(' + s.getTagVal('dyxSendMessage', 'paramSwTarget')
      +  ')')
}
```

Fig. 4. *MessagedyxSendMessage* rule

4. EXEMPLIFICATION OF MT-PROAPES

In this section, a concise and partial example of a display system controller is presented. That aims to demonstrate the feasibility in the use of the PROAPES profile, including its use in the transformation of MT-PROAPES models under the MDA approach. This example presents a sequence diagram that contains some actions of a *display object*, set to manage the *display thread* that controls the *display state*.

Fig. 5, step 1, shows the PIM model, i.e., a sequence diagram that illustrates the occurrence of the "Incorrect Password" string of the *display object (oDisplay)*. Stereotypes of the *dyxRTOS* profile are employed in the PIM model by means of messages that indicate the existence of software services and device drivers services of the RTOS, respectively. In the PIM, however, target platform details are not pointed out. For instance, the *SendCmdDisp(LCD_CLEAR, 4, 20)* message is annotated with the «*dyxSendCommandDD*» stereotype, and represents the use of a device driver service, though not depicting the technical details of a specific platform.

Fig. 5, step 2, illustrates the *swxRTOS* profile used to define the «*swxCore*» (RTOS software services) and the «*ddxLCD*» (device driver services) stereotypes. These stereotypes are used to annotate services in the platform models, i.e, they represent an abstraction layer for the *X Real-Time Kernel*. In this example, the following platform models are illustrated: X – ARM7 Atmel and X – ARM7 NXP.

Fig. 5, step 3, shows the MT-PROAPES model transformation, using the PIM annotated with the stereotypes of the *dyxRTOS* profile and the PM annotated with the stereotypes of the *swxRTOS* profile. These models are input parameters of the MT-PROAPES. The association between PIM and PM is made through tagged values described in the stereotypes used in the PIM. For instance, the *SendCmdDisp (LCD_Clear, 4,20)* message annotated with the «*dyxSendCommandDD*» stereotype, consists of the following tagged values: *opDDTarget = ddxSendCmd; paDDTarget = LCD_Clear, lin, col; and rtDDService = ddxLCD.* As a result, the link between PIM elements annotated with the *dyxRTOS* profile and PM elements annotated with the *swxRTOS* profile enables the MT-PROAPES execution. MT-PROAPES thus checks every element marked in the PIM and finds the corresponding element in the PM, enabling a PSM generation according to the platform selected in step 2.

Fig. 5, step 4, brings the generation of two platform-specific models: the X *Real-Time Kernel* for ARM7 Atmel and the X *Real-Time Kernel* for ARM7 NXP. The difference between both PSM models is that in the PSM – X ARM7 Atmel the *Command* message has three parameters: *LCD_CLEAR* – clear the LCD display; *lin* – number of lines in the LCD display; *col* – number of columns in the LCD display, while in the PSM – X ARM7 NXP the *Command* message has only the *LCD_CLEAR* parameter. Another difference refers to the *GotoLinColDisp(1,0)* message, illustrated in the PIM. As it is transformed into the PSM – X ARM7 Atmel, it is named *GotoLinCol(1,0)*, whereas by being transformed into the PSM – X ARM7 NXP, it is named *Posic(1,0)*.

5. RELATED WORK

Model transformation is one of the key points in MDA approaches. Therefore, in order to reduce the impact of changes in embedded software development, it is important that the Platform Model is explicitly defined in the model transformation, i.e., that the concerns of the Platform Model and the Model Transformation are defined separately. Most model-driven approaches for embedded software have focused on improving dedicated platform models [17][18]. Platforms are thus not described as input models for the development of model transformations. As a result, few works have aimed at embedded software development using explicit platform models in the MDA context [19][4].

One of the main obstacles in the MDA approach is the lack of adequate models for software and mechanism behavior so as to integrate behavioral models with structural models and with other behavioral models [20]. In its turn, this paper proposes a UML profile called PROAPES. This profile is used to create the behavioral models (PIMs) and structural models (PMs), explicitly defines the PM, and allows the link between application and platform.

The use of profiles for carrying out transformation of models is strongly encouraged in MDA-based software development, as well as in several existing approaches [21][22][23]. This study presented a model transformation based on the PROAPES profile that executes PIM-PSM model transformations into behavioral models, in the MDA context. This MT promotes a full separation of concerns that enables the creation of reusable, portable model transformations.

The approach described in [19] presents a model-driven methodology based on the TUT Profile. It is a UML profile targeted at embedded system design. TUT profile is intended to enable automated system design by using model transformation. However, this profile does not provide concepts for explicitly identifying platform resources and their services. As a result, the model transformation is a mixture of concerns. Selic describes a general UML profile for platform modeling and deployment of relationships between platforms and applications [4]. However, its profile does not provide specific artifacts to model RTOS execution resources. The UML profile Modeling and Analysis of Real-Time and Embedded systems (MARTE) [24] is a UML profile that provides a dedicated sub-profile for the development of Real-Time Embedded Systems, called Software Resource Modeling (SRM), permitting the description of the RTOS services. However, modeling for a specific RTOS requires the adaption of MARTE profile to the modeling conventions of that RTOS. Thus, the modeling will not always be straightforward.

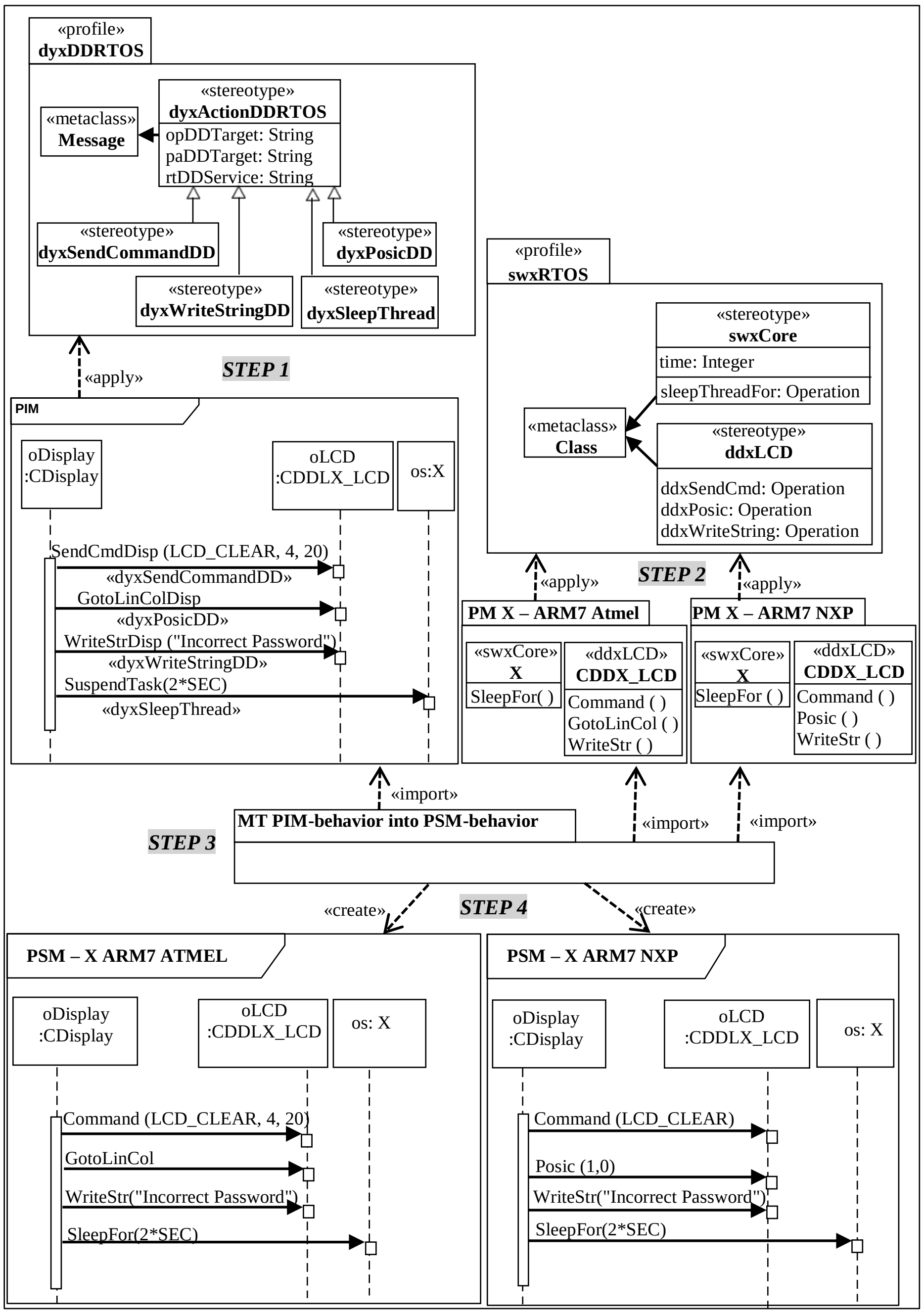

Fig. 5. Example of the PIM-behavior into PSM-behavior MT-PROAPES

6. CONCLUSION

Regarding the development of real-time embedded software based on the MDA approach, a crucial issue resides in the separation of concerns between application model and target platform. That is due to both the inherent complexity of this kind of software and the existence of a wide variety of applicable platforms.

This paper presented the PROAPES profile. It is used to represent PIMs and PMS, enabling RTOS services and attributes, as well as the corresponding associated hardware, to be used and depicted in a generic way.

Both the swxRTOS and the *dyxRTOS* are sub-profiles of the PROAPES profile. The former enables the annotation of PM structural models, whereas the latter enables the annotation of PIMs that represent the behavior of RTOS-based embedded software applications. In addition, the *dyxRTOS* subprofile provides stereotypes that make the association between application (PIM) and platform (PM) possible, i.e., the link between behavioral and structural models.

The main advantage in using this profile consists in generating well-defined platform models for a particular RTOS and their respective hardware platforms of embedded systems, based on specific processors required for their execution. This study also proposed the PIM-behavior into PSM-behavior MT based on the PROAPES profile, demonstrating the feasibility in the use of this profile in the context of the MDA approach. The Display Controller example demonstrates the use of the PROAPES profile and the MT-PROAPES in practice. In future works the PROAPES profile may be exemplified to other RTOS-based platforms. Furthermore, usage pattern description may be developed for this profile, in order to obtain an accurate description of the platform.

7. REFERENCES

[1] W. Wolf, "Computers as Components: Principles of Embedded Computer Systems Design", Morgan Kaufmann Publishers, 2001.

[2] OMG, June 2003, MDA Guide Version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01.

[3] M. Aksit, and I. Kurtev, "Elsevier special issue on foundations and applications of model driven architecture", Science Computer Programming, Vol. 73, No. 1, 2008, pp. 1-2.

[4] B. Selic, "On software platforms, their modeling with UML 2, and platform-independent design", Proc. of the 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, USA, 2005, pp. 15-21.

[5] M. Becker, G. Di Guglielmo, F. Fummi, W. Mueller, G. Pravadelli, and T. Xie, "RTOS-aware refinement for TLM2.0-based HW/SW designs", Proc. of the Conference on Design, Automation and Test in Europe, Belgium, 2010, pp. 1053-1058.

[6] D.P.B. Renaux, R.E. Goes, and R.R. Linhares, "Performance characterization of real-time operating systems for systems-on-silicon", Proc. of the 12th Brazilian Workshop on Real-Time and Embedded Systems, Brazil, 2010. http://sbrc2010.inf.ufrgs.br/anais/data/pdf/wtr/st03_02_wtr.pdf.

[7] OMG, Unified Modeling Language (UML): Superstructure, 2011, http://www.omg.org/spec/UML/2.4/Superstructure/Beta2/PDF.

[8] A. Alti, T. Khammaci, and A. Smeda, "Integrating Software Architecture Concepts into the MDA Platform with UML Profile", J. Comput. Sci., Vol. 3, 2007, pp. 793-802.

[9] R. Van Der Straeten, V. Jonckers, and T. Mens, "A formal approach to model refactoring and model refinement", Software and System Modeling, Vol. 6, No. 2, 2007, pp. 139-162.

[10] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool", Science of Computer Programming, Vol. 72, No. 1-2, 2008, pp. 31–39.

[11] J. Troya, and A. Vallecillo, "A Rewriting Logic Semantics for ATL", Journal of Object Technology, Vol. 10, No.5, 2011, pp. 1-29.

[12] D. Wagelaar, R. Van Der Straeten, and D. Deridder, "Module superimposition: a composition technique for rule-based model transformation languages", Software and Systems Modeling, Vol. 9, No. 3, 2010, pp. 285-309.

[13] D. Wagelaar, "Composition techniques for rule-based model transformation languages", Proc. International Conference on Model Transformation, Switzerland (2008), Lecture Notes in Computer Science, Springer, Vol. 5063, 2008, pp. 152-167.

[14] R. France, and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap"; Proc. Future of Software Engineering (FOSE '07), Washington, DC, USA, 2007, pp. 37-54, IEEE Computer Society.

[15] OMGb. Object Constraint Language Specification, 2011.http://www.omg.org/spec/OCL/2.3/Beta2/PDF.

[16] M.D. Del Fabro, and F. Jouault, "Model Transformation and Weaving in the AMMA Platform", Proc. of the Generative and Transformational Techniques in Software Engineering (GTTSE'05), Braga, Portugal, 2005, pp. 71-77.

[17] S. Jeon, J. Hong, I. Song, and D. Bae, "Developing platform specific model for MPSoC architecture from UML-based embedded software models", The Journal of Systems and Software, Vol. 82, No. 10, 2009, pp. 1695-1708.

[18] G. Karsai, S. Neema, and D. Sharp, "Model-Driven Architecture for embedded software: A synopsis and an example", Science of Computer Programming, Vol. 73, No. 1, 2008, pp. 26-38.

[19] P. Kukkala, J. Riihimâki, M. Hamalainen, and K. Kronlof, "UML 2.0 Profile for embedded system design", Proc. of the Conference on Design, Automation and Test in Europe USA, 2005, pp. 710-715, IEEE Computer Society.

[20] E. Riccobene, and P. Scandurra, "Weaving executability into UML class models at PIM level", Proc. 1th. Workshop on Behaviour Modellingin Model-Driven Architecture, ACM, New York, NY, USA, 2009.

[21] K. Anastasakis, B. Bordbar, G. Georg, and I. Ray, "On challenges of model transformation from UML to alloy", Software and System Modeling, Vol. 9, No. 1, 2010, pp. 69-86.

[22] J.T.E Timm, and G.C. Gannod, "A Model-Driven Approach for specifying semantic web services", Proc. of the IEEE International Conference on Web Services, USA, 2005, pp. 313-320, IEEE Computer Society.

[23] R. Silaghi, F. Fondement, and A. Strohmeier, "Towards an MDA-Oriented UML profile for distribution", Proc. of the 8th IEEE International Enterprise Distributed Object Computing Conference, Monterey, California, USA, 2004, pp. 227-239.

[24] OMGc. June 2011, A UML profile for modeling and analysis of real time embedded systems, Version 1.1. http://www.omg.org/spec/MARTE/1.1/PDF

Received: August 2012. Accepted: September 2012.