MDA and Real-Time Java: The HIDOORS Project

Jean-Noël Meunier (meunier@aonix.fr)
Frank Lippert (lippert@aonix.de)
Ravi Jadhav (jadhav@aonix.com)
Nigel Harding (nharding@aonix.co.uk)

June 2004, Aonix Europe, Partridge House, Newtown Road, Henley-on-Thames, OXON RG9 1HG

Keywords:
UML, RT modelling, MDA, Profile, Automatic code generation, Java, Safety critical applications, RMA

Abstract:

Embedded Systems very often consist of a number of concurrent tasks, which have to be executed in a given time frame. Special tools are needed to analyse the schedulabity and to detect the overrun of given time targets ("Worst-Case-Execution-Time-Analysis"). Standard Java lacks some of the special features, such as deterministic garbage collection, needed for Embedded Systems.

The HIDOORS (High Integrity Distributed Object-Oriented Real-time System) project is focused on the development of a RT-Java environment and is funded by the European commission. This includes a UML based modeling environment and MDA technology, which facilitate the transformation into high integrity real time systems in Java.

Aonix developed a special RT-Java profile for this project, which is based on the OMG's SPT Profile ("Schedulability, Performance and Time") for embedded systems. This profile is the central part of HIDOORS and is used by tools such as the modeling tool, the model checker, the WCET Analyser and model transformation to the Java environment.

This paper will give a brief overview of the HIDOORS project, discuss how UML 2.0 Profiles are used to describe the required aspects of embedded systems and explain the model transformation process with an example.

1. Introduction

HIDOORS (High Integrity Distributed Object-Oriented Real-time Systems, http://www.hidoors.org) is a 30-month project consisting of European companies and research institutions and is partially funded by the European Commission (IST 2001-32329). The main goal of the project is to bring Java to applications that are hard real-time, embedded, distributed and safety critical. Additionally, the project aims to include all technologies and tools related to the development of a hard real-time application, real-time modelling, real-time analysis and proof of correctness. The project can be considered as being divided into two parts. Firstly, it relates to real-time Java Platform with an aim to solve problems such as deterministic garbage collection, real-time network support, fast RMI (Remote Method Invocation) as a means to communicate between components in a distributed environment. Secondly, it relates to real-time modeling with an intention to consider the question: how can critical and embedded real-time applications be modeled? This document focuses on the latter part and tries to at least partially answer this question.

To model an application, whatever the domain (real-time or not), it seems impossible to ignore the UML notation since it is now a well-known and recognized standard from the OMG group [1]. One of the main advantages of UML is that it is a generic notation that can address almost any domain (real-time, business, web applications, etc.). But designers often see this as an important drawback because the notation appears to be too general and too ambiguous to be used easily and efficiently for a particular domain. Fortunately, UML provides general extension mechanisms by means of stereotypes, tagged values and constraints to adapt UML to a specific domain. This is part of the UML profile definition. A UML profile describes the context of use of UML for a given domain and is defined by a subset of UML and some UML extensions. Profiles help to reduce ambiguity, reduce the complexity of models and to enrich their semantics. Models are then easier to specify, read, and process (profiles enable better automatic code generation and better model validation). That is why in most domains, a UML profile needs to be defined and used.

For the real-time domain, a profile already exists; it is the "UML Profile for Schedulability, Performance and Time" [2] (hereafter referred to as SPT) and has been adopted by the OMG group. This profile provides the basic constructs for real-time modeling. The feedback from the HIDOORS project related to this SPT profile is that:

1) The profile is too general as it covers all real-time problems both soft and hard.
2) The profile mainly defines the fundamental concepts, in other words the syntax, but it does not provide any indications concerning ways to use them, just like a dictionary of language that gives the definition of words but without any indication about how to build sentences by using these words.
3) Some concepts such as the communication means between tasks (see the next section) are missing in the profile.

For all these reasons, the HIDOORS project introduces a new profile named "HIDOORS profile", compliant with OMG's SPT and which takes into account the HIDOORS feedback and addresses distributed, critical and embedded applications.

Section 2 deals with the HIDOORS profile. It presents the objectives of the profile and the two views that the profile aims to address: the Rate Monotonic Analysis view and the task / inter-task communication view. To make the paper clearer, an example related to the communication pattern is presented. Section 3 covers the automatic code generation (a Model Driven Architecture approach) that takes as input a real-time UML model and generates as output real-time Java source code. More particularly, it shows how the code generation takes into account the HIDOORS profile concepts and maps them into Java source code.

2. The HIDOORS UML profile

The HIDOORS profile [3] aims to fulfil the following goals:

- to be compliant with the standard OMG's SPT profile
- to provide concepts that enable the specification of a RMA (Rate Monotonic Analysis) view of the model.
- to provide concepts that enable the specification of a task view (including inter-task communication) for the model.
- to provide a high level representation of asynchronous communication channels by introducing new patterns (for definition or more details on patterns, see [4] for general patterns and [5] for patterns related to real-time systems).
- to provide concepts that enable the specification of distribution concepts

Currently, concepts related to distribution have not yet been studied by the HIDOORS profile.

2.1. Rate Monotonic Analysis

There are two reasons for choosing the RMA view capability. First is the schedulability model, which is part of SPT profile and is mainly based on RMA. Secondly, for the HIDOORS project, one of the project validation applications is checked against RMA techniques.

The question related to the schedulability analysis is whether tasks can be executed such that all deadlines are met. RMA is often applied on the source code of a real-time system but performing such timing analysis at the model level enables the detection of potential specification errors earlier in the development process. If the rate monotonic analysis performed on the model concludes that the system is not schedulable, it is not worth continuing so long as the problem is not solved. However, the contrary is not true, that is if the rate monotonic analysis concludes that the system is schedulable, it does not mean that the final system will be schedulable. Still, performing this timing analysis at the model level can prevent many errors and has many benefits.

For a single processor/multiple threads system, the compliance of a model to rate monotonic analysis relies on describing the system from a concurrency point of view and as a set of scheduling jobs. Each job is composed of one trigger and one response. This description is called a real-time situation [6]. A *trigger* is principally described by an occurrence pattern (e.g. periodicity or statistical distribution) and figures as events of typical real-time modelling. A *response* is a set of sequential actions, which are principally described by their duration and the resources they need to access. They can be nested as sub-actions of an action, similar to the nested statements of a source code. A *resource* is any logical or physical item necessary to perform the action. Actually, it is not mandatory for designers to describe all resources used by the system. From the RMA point of view, the only relevant items are the shared resources that are resources, which are potentially used by several concurrent actions.

As a consequence, the goal of the HIDOORS profile is to define a set of elements and a set of rules that will allow the UML model to show scheduling jobs and consumed resources (see figure 1):

- A real-time system is a set of scheduling jobs.
- A scheduling job is made of one trigger and one response.
- A response is a set of actions.
- A trigger contains event timing information and is associated with one or more actions.
- An action contains duration information and is associated with zero or more resources. An action can also be made of sub-actions.

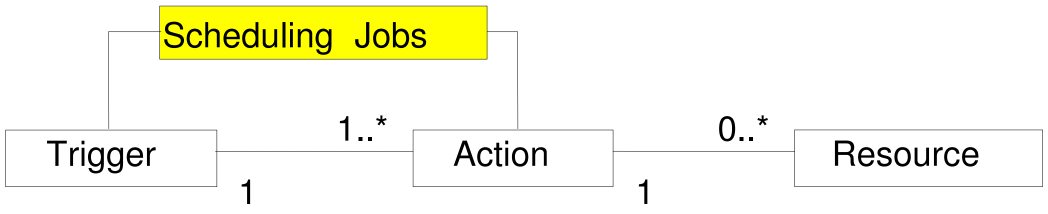

Figure 1. Scheduling jobs, triggers, actions and resources

The HIDOORS profile defines a set of elements based on the basic concepts of SPT: triggers are messages stereotyped as <<SATrigger>>, actions are messages stereotyped as <<SAAction>>, resources are objects stereotyped as <<SAResource>> (see figure 2). Thus, the HIDOORS profile gives more assistance to designers by providing these elements for re-use in models and defining which diagrams can be used for that purpose.

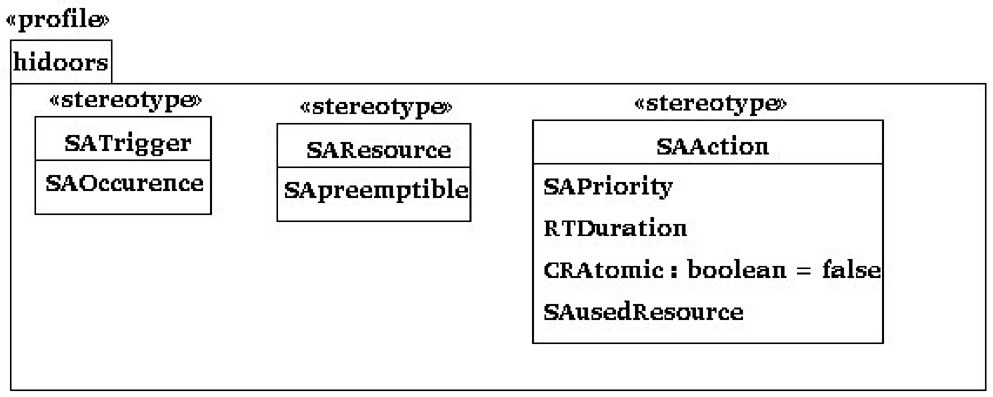

Figure 2. HIDOORS profile excerpt – triggers, actions and resources

2.2. Task view and inter-task communication

Another goal of the HIDOORS profile is to increase the level of abstraction of models thereby simplifying the effort of designers particularly when specifying asynchronous communication between tasks. New stereotypes, model elements and rules have been defined for this purpose. Three communication patterns are taken into account in accordance with the ARINC 653 standard in Avionics [7]:

- Buffer: messages are transmitted via queues with predefined capacity in FIFO order. This provides a communication channel with a "First In First Out" type of service (refer to ARINC 653 standard for more details).

- Blackboard: A message is put in a board and is either read by the receiver or overwritten by the next written message. There is no queuing of messages, but a message may be lost. This provides a communication channel with a "Last Message Only" type of service (refer to ARINC 653 standard for more details).

- Event: the event represents a simple synchronization channel (refer to ARINC 653 standard for more details) that can be used to notify another task that something happens. It works like a flag.

In the following, only the buffer communication pattern is presented, as the other communication patterns work in a similar manner.

The stereotype <<HIBuffer>> is an association between two classes representing both concurrent units (stereotyped <<HIConcurrent>>), conceptually using an instance of the class ARINCBuffer (see figure 3). It is important to understand that this template class instance is completely hidden, that it is implicit information that does not need to be specified in the model by designers and that will never appear in the generated Java source code. However, this information is useful for the automatic code generation to produce the correct source code corresponding to the communication pattern specification (see next section).

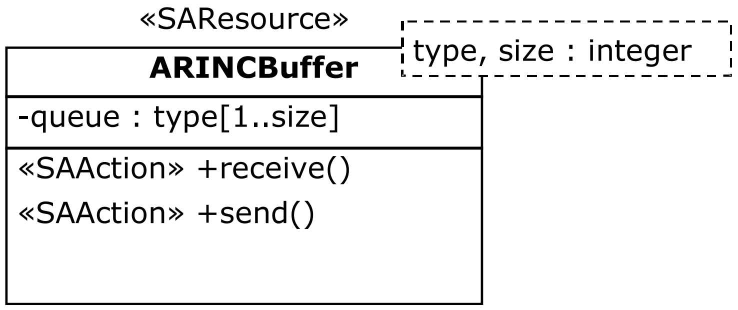

Figure 3. Implicit class for buffers

The *type* parameter of the ARINCBuffer template class must correspond to the type of the messages. This parameter can be set as a UML association class or as an association name. The *size* parameter should correspond to the maximum number of messages allowed simultaneously in the FIFO buffer. This value can be set from system specification or from simulation. The default value is infinite. This parameter can be set as a HIBufferSize UML tagged value, or within the association name (multiplicity). Figures 4 and 5 give an example of the use of this communication pattern. In the static view (figure 4), a task ("Sender") sends messages to another task ("Receiver"). The buffer size is set to 512. The message exchanged between the two tasks is of type "Message" (association class). In the dynamic view (figure 5), for a period the "Sender" sends two messages, however, the "Receiver" gets only one during that period (which means that the period of "Receiver" will have to be half that of "Sender" otherwise messages could be lost).

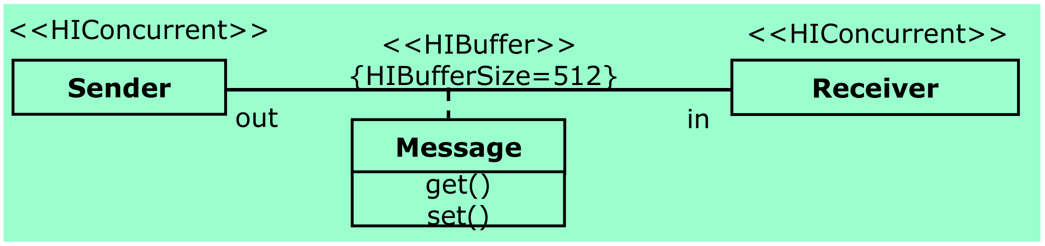

Figure 4. Example of buffer specification - static view

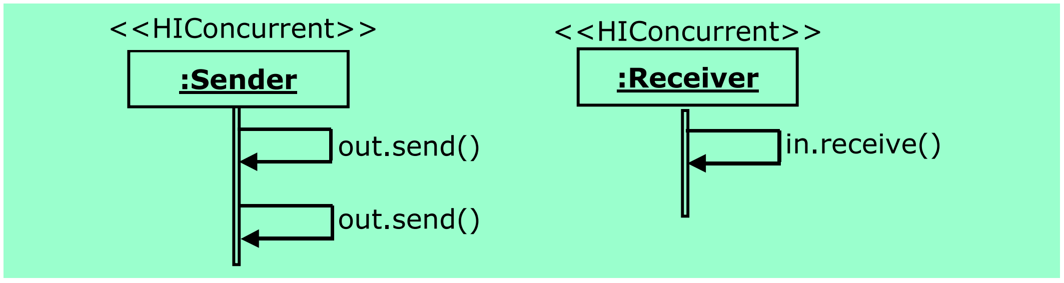

Figure 5. Example of buffer specification - dynamic view

Figure 6 gives an excerpt of the HIDOORS profile related to the three communication patterns: buffers, blackboards and events.

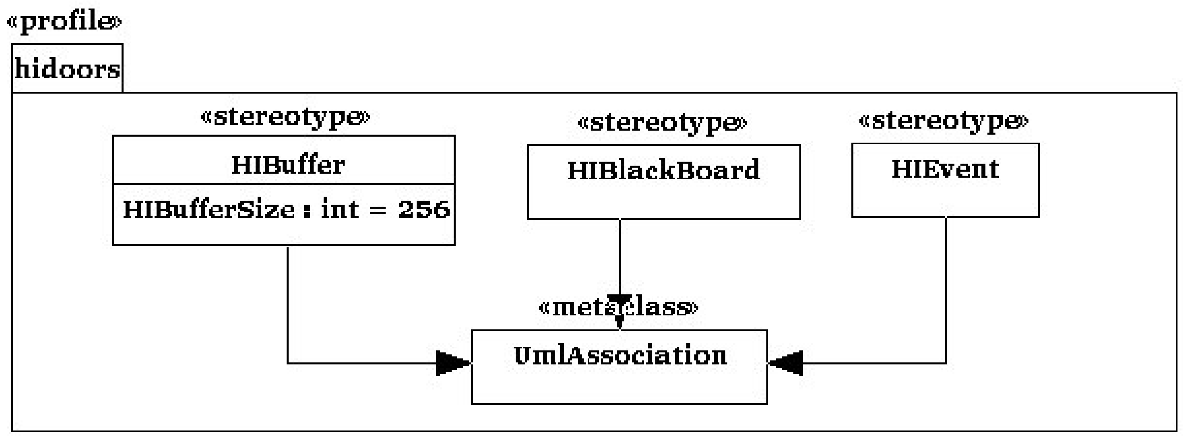

Figure 6. HIDOORS profile excerpt – the communication patterns

3. Automatic code generation

The Model Driven Architecture (MDA) approach is recommended by the OMG who recognised the need to improve software quality and to reduce development costs (see [8] for more details). The OMG place emphasis on model transformation and particularly the mapping of a Platform Independent Model (PIM) into a Platform Specific Model (PSM). The role of the automatic code generation is crucial to such an approach. In the HIDOORS project the automatic code generation consists of transforming the real-time model into real-time Java source code. The main work in the HIDOORS project is to add rules relating real-time behaviour to the general UML modeling approach (and also to translate UML concepts into Java concepts). The added value is then on the generation of source code from the HIDOORS profile constructs. More particularly, the role of automatic code generation is to break down the high level modelling and to make explicit constructs, which are implicit in the model. Figure 7 shows how the buffer communication pattern is handled. The abstract model of the example in figure 4 is mapped into a low level model taking into account the implicit information concerning the ARINCBuffer (see figure 3). Figures 8 and 9 give the resulting Java source code.

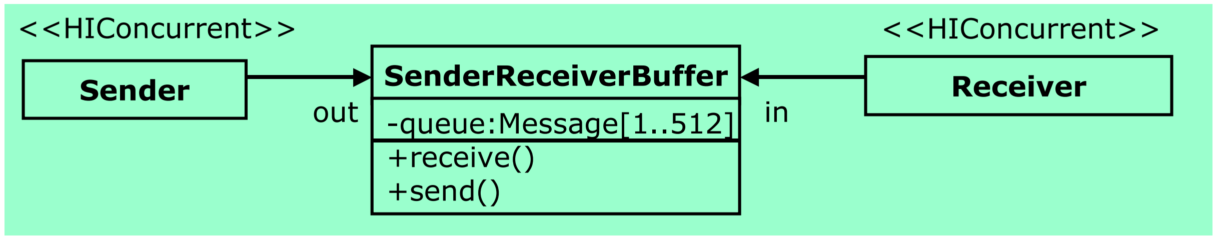

Figure 7. Part of model transformation - the example of the buffer communication pattern

```
public class Sender {

  // ------------------------------------------------------------
  // instance attributes
  // ------------------------------------------------------------
  private SenderReceiverBuffer out;

  //#ACD# M(UDAT::UID_65c15e75-0000067a-3ee5acf3-000626c6-00000004)
  //user defined code to be added here ...

  //#end ACD#
  ...
}

public class Receiver {

  // ------------------------------------------------------------
  // instance attributes
  // ------------------------------------------------------------
  private SenderReceiverBuffer in;

  //#ACD# M(UDAT::UID_65c15e75-0000067a-3ee5acfa-000aca45-0000000b)
  //user defined code to be added here ...

  //#end ACD#
  ...
}
```

Figure 8. Java source code for the Sender and Receiver class

```
public class SenderReceiverBuffer {

  // ------------------------------------------------------------
  // instance attributes
  // ------------------------------------------------------------
  /**
   * The buffer array holding the messages.
   */
  private Data[] queue = null;

  // ------------------------------------------------------------
  // methods
  // ------------------------------------------------------------
  /**
   * Obtains the next message from the message FIFO queue.
   */
  public void receive() {
    ...
  }

  /**
   * Puts a message at the last position in the message FIFO queue.
   */
  public void send() {
    ...
  }
```

Figure 9. Java source code for the SenderReceiverBuffer class

4. Conclusion

UML has a standard way to extend its semantics by stereotypes, constraints and tagged values. A collection of these is called a 'profile'. With the help of profiles, UML can be adapted to application domains for which standard UML is not specific enough.

This paper shows the development and application of a UML profile suitable for real-time modelling. The profile is largely compliant with standards and at the same time meets the specific needs of the HIDOORS project. To meet these requirements it uses a subset of the SPT real-time profile developed by the OMG and communication patterns from the ARINC 653 standard.

From an implementation point of view, the profile is implemented in a modelling tool through a profile editor as specified in the UML 2.0 standard. Also implemented is a Java code generator that makes use of the real-time profile by evaluating extensibility items applied to model elements. The modelling tool and the code generator StP (Software through Pictures), like the other tools for the HIDOORS project, are integrated into the Eclipse framework [9].

The HIDOORS profile and its corresponding automatic code generation are currently both used and tested by one of the three real-time applications aiming to validate the HIDOORS project.

One topic not covered by this paper is the analysis and validation of real-time models using a Worst Case Execution Time (WCET) tool . This work is performed by a HIDOORS partner.

5. Bibliography

[1] "OMG Unified Modelling Language Specification", Version 1.5, OMG group, March 2003, (http://www.omg.org/cgi-bin/doc?formal/03-03-01)

[2] "UML Profile for Schedulability, Performance and Time", Proposed Available Specification, OMG group, April 2003, (http://www.omg.org/cgi-bin/doc?ptc/2003-03-02)

[3] "A UML Profile for High Integrity Distributed Object-Oriented Real-time Systems (HIDOORS)", internal document, HIDOORS project, January 2003

[4] Gamma E., Heml R., Johnson R., Vlissides J., "Design Patterns", Addison-Wesley, 1995

[5] Douglass B. P., "Real-Time Design Patterns", Addison-Wesley, 2002

[6] Klein M. H., Ralya T., Pollak B., Obenza R., Gonzalez Harbour M., "A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems", Kluwer Academic Publishers, 1993

[7] ARINC specifications 653: http://www.arinc.com

[8] Model Driven Architecture (MDA) resources: http://www.omg.org/mda/

[9] The Eclipse platform website: http://www.eclipse.org