Michael Benkel
Aonix GmbH
www.aonix.de
michael.benkel@aonix.de

Applying UML Modeling and MDA to Real-Time Software Development

The growing complexity of embedded real-time applications requires presentation in a high level of abstraction. UML[1], the Unified Modeling Language, provides standard abstractions to simplify the documentation, understanding, and maintenance of complex software systems. Additional extensions to standard UML are required to represent schedulability, performance, and timing of real-time software. For the implementation part OMGs Model Driven Architecture (MDA)[2] can help to transform the UML models to the target environment. To demonstrate the value of UML to developers of embedded real-time systems, this presentation describes a practical application of real-time modeling that has been implemented with the Ameos modeling tool. Based on this UML models MDA provides a very safe and coherent way to generate code.

Overview

The development of software technology has consistently advanced by raising the level of design abstraction. Each step in this process requires two basic elements: a method of describing the next higher abstraction with the accompanying semantics and an efficient transformation. The latter is required to convert or compile a program description in the new higher level abstraction to an existing abstraction closer to the target hardware. With UML and Model Driven Architecture (MDA) the OMG provides the standards to achieve this. UML is the current candidate for the next abstraction level and MDA provides the semantics and the transformations necessary to compile UML to program code.

UML helps to improve communication in the project team and to have a clear interface to other stakeholders in the project. It is very often extended to project-specific needs (i.e., for embedded systems) by the use of profiles. One example of these kind of extensions is described in *UML Profile for Schedulability, Performance, and Time Specification (SPT)*[3], published by the Object Management Group (OMG) in 2002. SPT is too general for a lot of aspects and does not provide any hints on how to implement the described extensions..

The High Integrity Profile (HIP) described in this article was developed by an industry consortium with the objective of advancing technologies to improve developer productivity and real-time software reliability. HIP is based on available standards and consists of standard UML design patterns for common real-time programming problems.

Available Standards for Real-Time Development

There are several standards available for the implementation of real-time applications like ARINC 653[4] and Ravenscar. The ARINC 653 standard comes from the avionics community. It defines a RTOS API for a specific avionics platform supporting space and time

partitioning. It also provides several patterns for implementing communication between independent processes. Communication Patterns like Blackboard, Buffer and Event are widely used in the market.

The Ravenscar profile was defined in 1997 and specified with reference to the Ada programming language. It is based on a language-independent set of building blocks that are suitable for constructing typical real-time systems, and as input to analysis tools that provide evidence that the concurrency requirements of the system have been met.

In terms of modelling UML as defined by the OMG is the standard notation today. It provides several diagrams for modeling the different aspects of a system. Nevertheless the detailed meaning of the diagrams and the semantics are application-specific. This means that UML can be used to model a wide range of software systems, but very often an extension to the standard notation is needed to add additional semantics. The standard UML extension mechanisms are profiles which mainly consist of predefined Stereotypes and Tagged Values.

Each MDA platform requires a UML Profile as its semantic underpinnings and it has an impact on the model transformation to the target system. MDA modeling starts on a higher level of abstraction with the creation of a Platform Independent Model (PIM). A PIM is always focused on the domain of the project and mostly UML is used as the modelling notation. In the next step the PIM is mapped to one ore more Platform Specific Models (PSM) by adding technical aspects. This mapping is done by a transformer and transformation rules implementing technical patterns. This PSM is then mapped to the implementation by using the same mechanism. Implementation patterns drive the transformation of the model to the target environment.

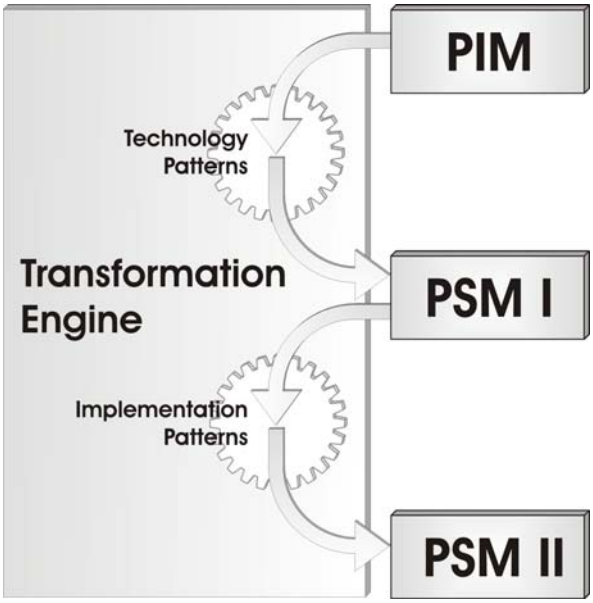

Figure 1: MDA Overview

One goal of MDA is to make the reuse of design models easier. Since platform dependencies are added later, the same design model can be used in many different settings. Platforms can be much more than just the underlying processor or operating system; rather they can be entire environments, such as middleware, component platforms, or libraries, thus adding to the flexibility of the design model.

The standard profile for real-time modelling is UML Profile for Schedulability, Performance and Time (SPT) defined by the OMG is the standard profile for real-time applications. It

provides the basic concepts for real-time modelling and defines several constructs for specifying timing relationships. SPT provides several extensions to UML like the definition of start and end time tags or a duration tag which can be used to capture time budgets in the UML model. These budgets represent the maximum time allowed for an action or message (worst-case execution time). These tags are used in sequence diagrams and state charts to limit the time available for critical code.

Although SPT is a good reference, for high integrity applications it is too general and it mainly defines fundamental concepts. It does not give any indications on how to use them and some concepts such as the communication means between tasks are missing in the profile.

HIP Profile

The High Integrity Profile (HIP) is a subset of OMGs SPT with extensions for High Integrity applications. Special attention was paid to keeping the disparity between SPT and the HIP profile to a minimum. Therefore SPT elements are used wherever possible.

HIP aims at defining the minimal set of abstractions suitable for hard real-time systems conforming to the Ravenscar model. Some common asynchronous communication paradigms coming from the ARINC 653 have been added for concurrency synchronization. HIP has been designed to enable real-time analysis like RMA to be done on the model itself. RMA capability is mainly covered by three stereotypes in SPT:

- <<SAtrigger>>, defining activating events (periodic threads, triggers)
- << SAaction>>, defining atomic actions (methods).
- <<SAresource>>, defining shared resources (classes, objects, members, etc).

Since real-time software designers usually need to map software components onto processes, threads or tasks which are higher level than just triggers and actions, additional stereotypes have been introduced to represent concurrent units. First the abstract class stereotype <<HIautoT ask>> was defined as inheriting from the pattern <<SAaction>> in association of an event itself stereotyped <<SAtrigger>>. Then two concrete class stereotypes were defined:

- <<HIperiodic>>, which represents the set of actions (response) which are performed each time a periodic timing signal occurs (trigger).
- <<Hisporadic>>, which represents the set of actions (response) which are performed each time an event signal occurs (trigger).

For synchronizing such concurrent units new stereotypes dealing with resource sharing and asynchronous communication have been defined. The following patterns have been introduced:

- <<HIpriorityCeilingEmulation>>, for resource sharing
- <<HIbuffer>>, for the bounded buffer communication paradigm.
- <<HIblackboard>>, for the persistent data broadcast communication paradigm.
- <<HIevent>>, for the classical event notification paradigm.

With these stereotypes HIP covers the main areas concurrency, resource sharing, and asynchronous communication They can be used in class diagrams, to give a static view of the system class structure; sequence diagrams, to model the interaction between objects; and state diagrams, to model the dynamic behaviour within a class. This allows an architect to insert information about the number of real-time tasks, frequency of particular real-time tasks execution, and anticipated worst-case execution times for each task into the platform independent model assertions. Other diagram types such as use case diagrams, which are intended to document user interaction with the system, will not be extended by HIP. They can be used as additional documentation or as part of requirements analysis.

HIP Example

In this section a simple example illustrates how HIP profile is applied. This example consists of two independent classes (tasks) and a communication mechanism. A class Score defines a method `collect():String` to collect results and to display the results there is a class Result with method `display(result:String)`. Both classes are periodic threads with certain priorities, To exchange date between these two threads we use a “blackboard” structure, so that they can send and receive messages. Then the receiver always reads the last message and incoming messages always overwrite the last message.

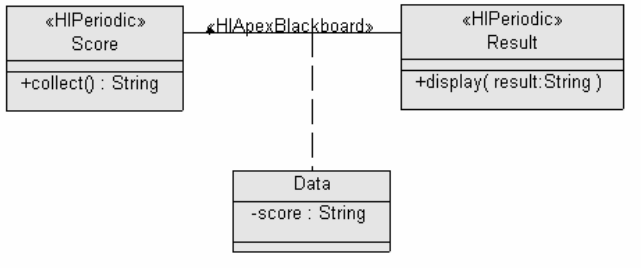

Figure2: Blackboard Example

Using the HIP profile it is very easy to model such an application. Since both classes are independent tasks, they are marked with Stereotype <<HIPeriodic >> or <<HISporadic>>. To use a blackboard as the communication mechanism, the association between two tasks is marked with the stereotype <<HIApexBlackboard>>. The static view Shown in Figure 2 shows what the UML model of this example looks like. The association class Data is used to define the type of the information exchanged between the two classes.

Implementation of the Profile

HIP also defines the implementation of the stereotypes. With HIP it is possible to focus on high level UML diagrams to describe the application being built. This description can be used independent from the target language. The transformation is based on MDA and special templates describe how the high level model is mapped to the target environment. This makes mapping very flexible and new requirements are easy to implement. The following example shows what an implementation in Java can look like. Based on the stereotypes used, several constructs are generated automatically in the source code. Score extends PeriodicThread; three additional attributes are generated and initialized, and a special constructo and a method public void handlePeriod() are generated.

```
public class Score extends PeriodicThread {
  private static int priority = 5;
  private static RelativeTime period = new RelativeTime(100 * 1000,0);
  private ScoreResultBlackboard scoreResultBlackboard;
  ..
  public Score(ScoreResultBlackboard connector) {
    ..
    super(priority, period);
    scoreResultBlackboard = connector;
  }

  public void handlePeriod() {
    ..
  }
}
```

Attributes *priority* and *period* are initialized with the tagged values and attribute scoreResultBlackboard is the connection to the artificially created blackboard class (see

below). It is set in the constructor above. The constructor invokes the superclass constructor with *priority* and *period* of class Score.
In the method body of handlePeriod(),some code has to be filled in manually. The run() method of superclass PeriodicThread invokes handlePeriod() periodically with the time value specified in attribute *period*. This is what the handlePeriod() method might look like:

```
public void handlePeriod() {
  Data data = new Data(collect());
  scoreResultBlackboard.write(data);
}
```

A new Data object is constructed and written to the blackboard instance. In collect(), everything could be implemented; for now, a dummy implementation might be sufficient.

```
public String collect() {
  String retVal = "";
  retVal = new Date(System.currentTimeMillis()).toString();
  return retVal;
}
```

To automate as much as possible in the HIP development process a full tool chain is available, starting with the Ameos modeling tool where the user can describe a high level view of the system under construction. All stereotypes of HIP are defined in the Ameos profile editor and assigned to the appropriate elements of the UML meta-model. This assures easy and correct usage in the modeling tool. The transformation in the target environment is carried out by the MDA generator of Ameos. As described earlier, this is a two step process. During the first step a platform specific model is generated by implementing technical pattern like the blackboard. Finally this PSM is mapped to the target environment by implementing it in Java for example.

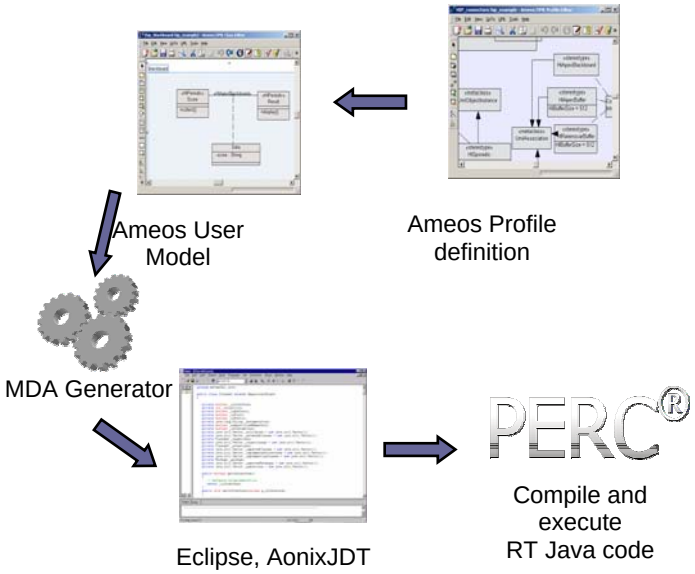

Figure 3: HIP Tool Chain

In an IDE like Eclipse or AonixJDT the user can then implement the missing pieces of the application. The Java code can then be executed with the PERC Virtual Machine (VM) supporting the execution of Java™ platform applications in embedded systems.

Summary

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

HIP represents an implementation of modeling real-time software for embedded systems using UML and Model Driven Architecture. This approach demonstrates that it is possible to automatically generate source code for real-time applications from the high-level architectural models. Model checking capabilities allow design inconsistencies to be identified and corrected early in the development process. This profile tries to be compliant with standards and at the same time meet the specific needs for high integrity applications. Therefore it uses parts of the SPT real-time profile developed by the OMG and communication patterns from the ARINC 653 standard. The MDA based code generation brings all the benefits of MDA stated by the OMG to the real-time market. High level problem specific models are transferred by a transformer in safe and coherent source code.

As a result, HIP is a lightweight and easy to use UML profile, which can be used for a wide range of applications in many different branches of industry such as avionics, automotive and telecommunication.

References

[1] "OMG Unified Modelling Language Specification", OMG group, (http://www.omg.org/uml)

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

[3] "UML Profile for Schedulability, Performance and Time", OMG group

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