ABSTRACT
The current trend in Model Driven Architecture is to use model transformation to refine a model from a platform-independent model to a platform-specific model, resulting in a linear development process. With the advent of commercially available realtime and safety critical Java implementations as a basis for platform neutral realtime systems, one could raise the level of reuse in modeling realtime systems by keeping platform independent information completely separate from platform specific information, so that model refinement is done by adding modeling aspects and maps between aspects rather than by model transformation. The result is a modeling process that is less order dependent, thereby supporting the full flexibility of Java Technology even for realtime system design.

Categories and Subject Descriptors
D.2.2 [Software Engineering]: Design Tools and Techniques

General Terms
Design

Keywords
Realtime Java, UML, MDA, Modeling

1. INTRODUCTION
Embedded Systems are becoming ubiquitous in daily life around the globe. For instance, they are essential to the modern features now available in washing machines, TVs, automobiles, airplanes, and medical devices. The automotive industry is a good example to summarize the trends in the development of embedded systems, where one can see an increasing share of software in products and processes. Much of the innovation and the competitive differentiation is based on software. Embedded systems are often distributed realtime systems integrated in an environment of other hardware and software systems and are characterized by timing and memory constraints. A modern car contains more than 50 embedded controllers coupled by multiple networks. More than 600,000 lines of code implement the functionality of the software. All in all, automotive systems, as with those in other industries, are experiencing ever increasing complexity of systems and processes.

1.1 Challenges of Modeling Realtime Systems

As the functionality of systems increases, so do the number and extent of decision taken without human intervention. Often these systems are safety critical, that is software systems whose failure may lead to loss of life or severe injury. To safeguard human life, safety critical systems in at least some industries are required to go through a formal certification process. Well defined in-house software development processes are also used by some companies to reduce the risk of legal liability for safety critical products that are as of yet unregulated or weakly regulated. Whether defined in-house or legally mandated, the goal is to clearly demonstrate that a given system meets a set of well defined requirements. This requires being able to trace requirements from initial model down through code generation to test and validation suites.
pectives are also rising. In a global market, this results in an ever decreasing expectation of “time to market” to keep up with the competition. Less time for higher quality software product (and with open source, less revenue) is straining the ability for companies in all industries to keep up. It is no longer sufficient to simply hand code applications using state-of-the-art programming languages. Rather, a more abstract view of the systems is required to manage complexity and to enable developers to focus on, and solve, one problem at a time. In addition, the properties of a system should be verified and analyzed as early as possible in the development process to avoid costly changes in later stages [11]: The verification of the systems properties and constraints, supported by an integrated tool chain, is the basis for the development of high quality embedded systems.

Software models, such as those expressed in the Unified Modeling Language [8] (UML) from the Object Management Group (OMG), provide such an abstract view of a system; however, developing software based on these abstract models is still a complicated and error prone task. Model Driven Development (MDD) tackles this issue by using abstract descriptions in the form of models to generate the actual system implementation; or at least as much as can be generated from these models. MDD is thus, a prime candidate for helping to cope with the complexity of software system development.

1.2 Model Driven Architecture

OMG has standardized one approach to MDD under the name of Model Driven Architecture (MDA) [6]. One of its main goals is separating design from infrastructure and implementation technologies. Using the MDA methodology, system requirements may first be defined as a conceptual design independent model (CIM). In a second step the CIM is refined into a platform independent model (PIM) expressed in UML. The PIM represents a conceptual design implementing the functional requirements and thus is independent of the specific technological platform used for implementation. In contrast, the platform specific model (PSM) represents a model of a software or business system that is linked to a specific technology (e.g. a specific programming language, operating system or database). The translations between PIM and PSM can be performed using tool support, for example model transformation tools compliant with the OMG standard QVT [7, 11]. The verification of non functional properties (NFP) of a system can thus be more easily specified for each stage of the software development process. This model of development is shown in Figure 1.

Promising though they are, the current standards for UML and MDA do not fully consider the capabilities of architecturally neutral realtime system [3] (ANRTS) [3] design that languages such as Realtime Java can provide. In particular, when such a language is used in critical systems, where non functional constraints such as response time and timeliness are critical aspects of correct system behavior, the current standards do not provide for adequate separation of design, topology, operating environment, and computational environment to ensure minimal redesign and reanalysis when any aspect of a system changes. As certification, or at least requirements based product life cycle control, becomes more prevalent, such flexibility will be crucial for minimizing costs and maintaining a competitive edge.

<table>
<thead>
<tr>
<th>Profile</th>
<th>UML Profile</th>
<th>MARTE*</th>
<th>HIDOORS</th>
<th>SysML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Annotations</td>
<td>light weight</td>
<td>light weight</td>
<td>light weight</td>
<td>light weight</td>
</tr>
<tr>
<td>Schedulability</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Performance Analyze</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Quality of Service</td>
<td>(explicitly)</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Supports Defining Metrics</td>
<td>(explicitly)</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Fault Tolerance</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Formal Semantics</td>
<td>-</td>
<td>(time, resources)</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Supports Modeling Embedded Systems</td>
<td>-</td>
<td>(explicitly)</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Supports Modeling Realtime Systems</td>
<td>+</td>
<td>(explicitly)</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>Supports Requirements Engineering</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Supports MDA</td>
<td>-</td>
<td>(explicitly)</td>
<td>(explicitly)</td>
<td>+</td>
</tr>
<tr>
<td>UML 2.0 Compatible</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>OCL 2.0 Compatible</td>
<td>-</td>
<td>+</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Supports Design Retargeting</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

*Terminated proposal, not stable

Table 1: Overview features in existing profiles

2. UML FOR EMBEDDED SYSTEMS

UML provides a great degree of freedom in expressing design elements and their relationships. It is designed to be usable for as wide a range of applications in computational systems as possible. Unfortunately, to be able to cover many different applications, the base semantics is quite weak; therefore, the concept of a profile has been included to be able to adapt the semantics of UML to a specific domain. When using UML models as the basis for MDA, the specification of the semantics becomes even more important. Though much work has been done to provide profiles for many interesting areas, the work for embedded and realtime systems has not been very successful in producing an adequate profile, particularly for systems where portability is an issue.

In order to provide profiles, UML provides means for being extended. Construction of a UML Profile enables the user to build specific UML models for any particular domain. To this end, UML offers mechanisms such as stereotypes and tagged values that can be applied to all available UML elements (e.g. attributes, methods, links) without changing the metamodel itself. A UML profile is then a collection of such extensions that describes a particular modeling problem or domain based on a set of well defined stereotypes and tagged values.

Profiles have been developed using these mechanism both within and external to the OMG.

2.1 Current Profiles

There are only a hand full of profiles which take real-time consideration into account at all. The OMG produced the UML Profile for Schedulability, Performance and Time (USPT) which they expect to be replaced by the Modeling and Analysis of Real-Time and Embedded systems (MARTE) profile in the near future to address missing concerns and update to UML 2.0. The IST Project HIDOORS based its profile on USPT. The Systems Modeling Language (SysML) is also of interest for realtime. Though none is completely applicable for ANRTS development, each has interesting features.

The USPT[9] profile defines several constructs for specifying timing relationships and provides modeling techniques
for resources, time, concurrency, scheduling, and performance. These features proved to be useful; however, USPT is quite large which gives too many different ways to express the same feature. In addition, there are no semantics for supporting MDA based realtime verification and code generation. The other main drawback of USPT is that it is not compatible with UML 2.0 (the currently used UML version) and it lacks a formal semantics. It also does not provide techniques for modeling system hardware and due to its huge size it is also not easily applicable [5, 2].

When the OMG became aware of these problems, it started to work on a new profile, MARTE[10]. The main goal of the currently running proposals is addressing the above mentioned problems of USPT. Since the current status of MARTE is a “revised proposal” it cannot be considered stable yet.

Besides the OMG’s effort of providing profiles, industry and research institutions frequently publish ongoing improvements and new profiles. One of these profiles, based on USPT, was proposed in the HIDOORE High Integrity Distributed Object-Oriented Realtime Systems project[4]. This project advanced the concept of Quality Driven Design (QDD) as a combination of MDA and Model Driven Verification (MDV). The salient feature was to not only generate model refinements and code, but also models for verification. Although it improves the applicability due to its compactness and support for model level verification, it does not sufficiently address timing and has not been updated to be compatible with UML 2.0.

Finally, SysML[1] as a domain-specific modeling language for systems engineering applications provides methods that are also interesting for modeling embedded systems. These are mainly used blocks in block diagrams for modeling different purposes. Table 1 gives an overview of the features of the above mentioned profiles.

None of these profiles support the level of platform independency that Realtime Java offers, thereby limiting the ability to retarget and reuse existing designs and design elements.

2.2 Rationale for a New Profile

The currently available UML profiles lack several important features needed to support an architecturally neutral development of realtime and critical systems with MDA; therefore, a new profile needs to be developed. As many existing ideas and structures should be adapted from existing profiles as possible. The proposed SuReal profile collects characteristics of all previously mentioned profiles: USPT, MARTE, SysML, and HIDOORE. The main contribution of this profile is twofold: it both provides modeling techniques that support the MDA based separation of PIM and PSM models as well as enabling the modeling of system architectures which is barely addressed in existing profiles and techniques. To fully support ANRTS design, this new profile provides separate UML modeling views that enable the clear distinction between application and architecture details.

3. SuReal Profile Structure

The SuReal UML Profile needs to support both ANRTS development and QDD. The only way to accomplish both goals is to de-emphasize a linear refinement model and consider a more flexible mapping model. By separating function, from topology, from operating environment, from computational environment and using mappings to relate these for describing the intended system, the designer gains more freedom to alter each domain individually. In a pure refinement scheme, this would be much more difficult. This domain mapping is central to the SuReal methodology.

3.1 Separation of System Views

In order to provide the flexibility needed to support ANRTS design, the SuReal divides a system’s model into four aspects: design, topology, operating environment, and computational environment. The design describes the function of a system, as well as its non-functional constraints. The topology describes how functions are shared among functional units and what the communication links are. The operation environment describes the software infrastructure on which the system is intended to run, including operating system and middleware. The computational environment is the actual hardware elements that implement the topology and are to host the design. These model aspects are maintained separately so that a design could be easily rehosted on another hardware and a hardware can be reused for another design. Likewise, the operating environment and topology could also be changed without effecting the other aspects of the model. The relationship between these aspects can be described by mappings of elements of one aspect with elements of another. The goal is to minimize the reanalysis necessary for retargeting a design.

UML already supports a variety of diagrams for modeling a system’s design. For instance, class diagrams are used to describe the static structure of the class hierarchy, whereas state diagrams and sequence diagrams provide two different ways of describing the dynamic behavior of a system. This same mechanism can be used to model the four SuReal aspects by using sets of diagrams for each system aspect. A diagram can also be used to describe the mapping between two aspects.

UML models are also used to capture information for the analysis and verification of non-functional properties (NFPs) at different levels of abstraction. The MDA methodology starts with a PIM and refines it into a PSM. The PSM is then refined into an implementation through code generation. While the application aspects represent the platform-independent, conceptual design of a System, the infrastructure aspects provide the specific technology platform on which the system is to run.

Whereas MDA is based on successively refining a model through the production of a new model at each step, the SuReal profile keeps platform-independent information separ-
WCET is only used as an analysis result, whereas the term analysis results from modeling constraints. For instance, information is necessary. Care has been taken to distinguish aspects. The rest is more concrete where platform specific abstract and can be done on the PIM: design and topology given target platform done on the PSM.

One could think of the SuReal modeling space being divided into aspect quadrants, where the horizontal division is between function and architecture and the vertical division distinguishes between application and infrastructure. The result is the four aspects introduced above: design, topology, operating environment, and computational environment. When using an architecturally neutral realtime programming language, such as realtime or safety critical Java, code generation can be done directly from the application aspects. The rest of the aspects are then only needed for non functional analysis and system deployment. This is a natural way to separate system information, providing clarity and enabling efficient changes and reuse.

Figure 2 shows the horizontal and vertical divisions and the four resulting aspects. The design aspect focuses on system functionality. It also covers all possible NFPs on an abstract level. The topology aspect describes the platform-independent system architecture. The two platform-specific views link the system description to a specific hardware. The operating environment aspect defines the underlying support software, e.g., operating system and virtual machine. The computational environment view contains all the information of the concrete hardware, e.g., details about the processor or bus to be used. All platform-dependent information necessary to verify the non functional constraints need to be captured in the operating environment and computational environment aspects for software and hardware respectively.

The computational environment view contains all the information that could be obtained through analysis and its relationship to modeling elements.

3.2 Classification of Modeling Elements

In order to use a modeling process that is divided into the aforementioned aspects: design, topology, operating environment, and computational environment; one must know what information belongs in each of these modeling aspect. Furthermore, one needs a representation for system concepts and elements that may be used in each aspect; therefore, not only must all relevant design element be assigned to an aspect, but each element also needs to have an associate UML representation. Finally, mappings are needed to define so that analysis tools know what information to use when analyzing an ANRTS design.

The SuReal profile uses the following diagram types to capture modeling information:

- class diagrams to model static views of the class structure;
- sequence diagram to model interactions between objects;
- state diagrams to model the dynamic behavior within a class; and
- composite structure diagrams to model the topology, operating environment, and computational environment of a system.

All but the last diagram type are used to describe the design aspect of the model. All other aspects use the composite structure.

The NFPs associated with the various modeling aspects must also be captured in their corresponding diagrams. UML elements are needed not only to support verification, but also code generation. UML provides Stereotypes and tagged values as mechanisms that can be used to extend its core concepts. The SuReal Profile uses UML stereotypes to describe the additional properties needed for ANRTS modeling.

In SuReal, model checking plays an important role in the verification process. Once the model is sufficiently specified, XMI can be generated from the annotated model and then be transformed into a timed automata for the checking process. This verification indicates whether the design is feasible, e.g. the modeling timing constraints can be met given that the time budgets are valid. This verification is independent of the target platform; the time budgets themselves are checked at a later stage when the target platform
is specified. A negative result at this stage implies that the system will not be feasible on any target platform while a positive result means that on a platform on which the time budgets are met, the timing constraints will hold.

Once the system model has passed the model verification, ANRTS code can be generated and a target platform can be defined or selected for deployment. The UML models are used to generate class skeletons that can be used as a basis for implementation. Platform specific analysis can now be performed. Analyzes such as worst case execution time analysis can only be done for a concrete platform. With SuReal aspect modeling and ANRTS, platform independent code can be generated and platform dependent analysis for any specific target environment can be done later.

4. ANRTS MODELING

Using mapping instead of model transformation has the advantage of providing greater flexibility. Given the division of the modeling effort into four aspects: design, topology, operating environment and computational environment, an ANRTS developer can model the PIM aspects and the PSM aspects independently and then relate them via maps. There are still parts of the QDD (MDA and MDV) process, such as Java code generation, which require transformation, but these remaining transformations do not bind the model to a given deployment. The generated code itself can be completely platform independent, but since the goal is to support real-time and safety systems, infrastructure information is needed to qualify the performance of the design in the given operating environment.

4.1 Design Modeling

The Design modeling aspect represents an abstract, platform independent view of the application. Thus, it focuses on tasks, messages, data structures and—most important for model checking time behavior—time and other resource budgets.

4.1.1 Stereotypes and Tagged Values

The Design subprofile is shown in Figure 5. This subprofile has three main stereotypes: SRTask, SRMessageBudget, and SRExecutionBudget which are further specified by inheritance, leading to the applicable leaves for design modeling. These stereotypes are defined by a UML profile diagram as depicted in Figure 5. As can be seen, the Design subprofile defines eight substereotypes as well.

Stereotype SRTask.

**SRTask:** any path through the code, which needs CPU-time and can be scheduled. It has a defined starting point. Five tagged values support this stereotype.

• **SRExecution budget:** time a task can use on an executing Resource.
• **SRTotal blocking time budget:** time budget which tasks wait for some resources other than the execution resource during a release.
• **SRA ctivation time budget:** The time from an event until the associated task begins execution. This is an artifact.
• **SRPreemption time budget:** time budget that can be spent waiting for the execution resource during a release.

![Figure 5: Design Modeling Aspect Extension](image)
• **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.

• **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by a earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

The stereotype `Priority Schedule Parameters` can be attached to an UML Class or UML Object Instance.

The stereotype `SRBackgroundTask`.

**SRBackground Task**: is a non-time-critical task that is scheduled in a simple round-robin schedule if the microcontroller is idle.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>E phạm</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRBackgroundTask</td>
<td><em>&lt;SRTask&gt;</em></td>
<td>SRClass UMLObjectInstance</td>
<td>SRClass UMLObjectInstance</td>
<td>SRExecutionTime Budget</td>
<td>SRDeadline</td>
</tr>
</tbody>
</table>

Five Tags are defined.

• **SRExecution budget**: time a task can use on an executing Resource.

• **SRTotal blocking time budget**: time budget which tasks wait for some resources other than the execution resource during a release.

• **SRActivation time budget**: The time from an event until the associated task begins execution. This is an artifact.

• **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.

• **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by an earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

The stereotype `SRBackgroundTask` can be attached to an UML Class or UML Object Instance.

The stereotype `SRTriggeredTask`.

**SRTriggered Task**: is a Task without own known activation. The activation occurs through a specified trigger (which can be the result of a task).

It also has five tags.

• **SRExecution budget**: time a task can use on an executing Resource.

• **SRTotal blocking time budget**: time budget which tasks wait for some resources other than the execution resource during a release.

• **SRActivation time budget**: The time from an event until the associated task begins execution. This is an artifact.

• **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.

• **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by an earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

This stereotype cannot be attached to any UML element.

The stereotype `SRTriggeredTask` can be attached to an UML Class or UML Object Instance.

Sporadic tasks have eight parameters.

• **SRMinimum interarrival time**: the minimum time between the arrival of two tasks.

• **SRMaximum interarrival time**: the maximum time between the arrival of two tasks.

• **SRJitter**: the maximum variation of some value.

• **SRExecution budget**: time a task can use on an executing Resource.

• **SRTotal blocking time budget**: time budget which tasks wait for some resources other than the execution resource during a release.

• **SRActivation time budget**: The time from an event until the associated task begins execution. This is an artifact.

• **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.

• **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by an earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

The stereotype `SRSporadicTask` can be attached to an UML Class or UML Object Instance.
Stereotype SRPeriodicTask.

SRPeriodic Task: Task with known activation cycle. The activation will be strictly periodic.

Periodic Tasks have the following tagged values:

- **SRTrigger**: is an event that causes the release of a task.
- **SRMinimum interarrival time**: the minimum time between the arrival of two tasks.
- **SRMaximum interarrival time**: the maximum time between the arrival of two tasks.
- **SRJitter**: the maximum variation of some value.
- **SRExecution budget**: time a task can use on an executing Resource.
- **SRTotal blocking time budget**: time budget which tasks wait for some resources other than the execution resource during a release.
- **SRAcquisition time budget**: The time from an event until the associated task begins execution. This is an artifact.
- **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.
- **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by a earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

The stereotype SRPeriodicTask can be attached to an UML Class or UML Object Instance.

Stereotype SRDataStructure.

**SRData structure**: This is a class without any methods used to describe objects such as data transmission packets.

Stereotype SRExecutionBudget.

**SRExecution budget**: time a task can use on an executing Resource.

Only one tagged value is defined.

- **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by a earliest deadline first (EDF) scheduler. Usually it is given relative to some event.

The stereotype SRExecutionBudget can be attached to an UML Transition or UML Operation.

Stereotype SRReleaseBudget.

**SRRelease budget**: the time budget for triggering some task by some event.

Releases are a bit more complex and so have five tagged values.

- **SRTotal blocking time budget**: time budget which tasks wait for some resources other than the execution resource during a release.
- **SRAcquisition time budget**: The time from an event until the associated task begins execution. This is an artifact.
- **SRPreemption time budget**: time budget that can be spent waiting for the execution resource during a release.
- **SRDeadline**: the specified point in time by which a task must be finished. It is used explicitly by a earliest deadline first (EDF) scheduler. Usually it is given relative to some event.
- **SRExecution budget**: time a task can use on an executing Resource.

The stereotype SRReleaseBudget can be attached to an UML Class, Object Instance, Transition or Operation.
4.2 Topology Modeling

The Topology modeling aspect is rather simple. It describes the application’s hardware in a platform independent way. It specifies the interconnection between nodes, the links, and specifies the paths for messages. Only the Composite Structure Diagram\(^1\) is needed to model topology.

4.2.1 Stereotypes and Tagged Values

The Topology subprofile is shown in Figure 6. It has two applicable stereotypes: SRNode and SRLink.

![Figure 6: Topology Modeling Aspect Extension](image)

Figure 6: Topology Modeling Aspect Extension

**Stereotype SRNode.**

SRNode: is an abstract execution unit for tasks.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>UMLClass</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRNode&gt;&gt;</td>
<td>x</td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRNode can be attached to an UML Component.

![SRNode](image)

**Stereotype SRLink.**

SRLink: is a common communication platform for multiple nodes.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>UMLClass</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRLink&gt;&gt;</td>
<td>x</td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRLink can be attached to an UML Component.

![SRLink](image)

4.3 Operating Environment Modeling

The operating environment aspect describes the software part of the target system. The most common element is the operating system itself, though other support software such as network protocols, virtual machines, and middleware may be included here.

4.3.1 Stereotypes and Tagged Values

The following stereotypes are provided in the Operating Environment subprofile: SRBusProtocol, SROS, and SRSchedulingParameters, as shown in Figure 7.

![Figure 7: Operating Environment Modeling Aspect Extension](image)

**Stereotype SRBusProtocol.**

SRBus protocol: specifies the format for data exchange via the bus-system.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>UMLClass</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRBusProtocol&gt;&gt;</td>
<td>x</td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRBusProtocol can be attached to an UML class.

![SRBusProtocol](image)

**Stereotype SROS.**

SROperating system (OS): is a set of computer programs that manage the hardware and software resources of a computer. An operating system processes raw system and user input and responds by allocating and managing tasks and internal system resources as a service to users and programs of the system. At the foundation of all system software, an operating system performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating networking and managing file systems. Most operating systems come with an application that provides an interface to the OS managed resources. [12]

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>UMLClass</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SROperatingSystem&gt;&gt;</td>
<td>x</td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SROperatingSystem can be attached to an UML class.

![SROperatingSystem](image)

**Stereotype SRSchedulingParameters.**

SRScheduling Parameters: Priority, Deadline, Preemption, Time Slot, etc.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>UMLClass</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRSchedulingParameters&gt;&gt;</td>
<td>x</td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRSchedulingParameters can be attached to a UMLPartInstance.

![SRSchedulingParameters](image)

\(^1\)only available in UML 2.0
4.4 Computational Environment Modeling

The Computational Environment subprofile describes the actual hardware components used in the PSM. It is part of a possible deployment infrastructure of an ANRTS model. It primarily provides the hardware specific information needed to analyze the timing behavior of its model.

4.4.1 Stereotypes and Tagged Values

The following stereotypes are provided in the Computational Environment subprofile: SRProcessor, and SRNetworkSegment, shown in Figure 8.

![Figure 8: Computational Environment Modeling Aspect Extension](image)

Stereotype SRProcessor.

SRProcessor: is a component in a digital computer that interprets computer program instructions and processes data.

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>PSM</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRProcessor&gt;&gt;</td>
<td>x</td>
<td></td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRProcessor can be attached to an UML Class.

Stereotype SRNetworkSegment.

SRNetwork segment: is a bus???

<table>
<thead>
<tr>
<th>Stereotype</th>
<th>PIM</th>
<th>PSM</th>
<th>Base Class</th>
<th>Parent Class</th>
<th>Tags</th>
<th>Inherited Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;&lt;SRNetworkSegment&gt;&gt;</td>
<td>x</td>
<td></td>
<td>UMLClass</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The stereotype SRNetworkSegment can be attached to an UML Class.

4.5 Mappings

In order to use mappings in place of transformations for model refinement, several mappings are needed. Each mapping represents some step in the refinement process. In accordance with the MDA paradigm, though not a transformation, each refinement step fills the model with the details needed for that refinement step. Mappings are the essential facility for flexible design change in ANRTS models. Three types of mapping are necessary: Application Mapping, Scheduling Mapping, and Architecture Mapping.

4.5.1 Mapping Dependencies

Mappings enhance models by relating different modeling aspects. This means that the corresponding aspects must first exist. Each map requires its own model. The application map requires the design and topology model, the scheduling map requires the operating environment model as well, and the architecture map relies on the topology and computational environment model, as depicted in Table 2. Attention must be kept on later model changes after the completion of any mapping. In this case each mapping of the modified models needs to be refreshed.

<table>
<thead>
<tr>
<th>Map</th>
<th>From Aspect</th>
<th>To Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>design</td>
<td>topology</td>
</tr>
<tr>
<td>Scheduling/OS</td>
<td>design</td>
<td>operating environment</td>
</tr>
<tr>
<td>Architecture</td>
<td>topology</td>
<td>computational environment</td>
</tr>
</tbody>
</table>

Table 2: Map Dependencies

![Figure 9: Mapping Dependencies](image)

Application Mapping.

The application map bridges the gap between design and topology at the application level. It maps processes to nodes and messages to links. The process map deploys the processes on the nodes of a topology while the message map specifies the path of a message on a topology.

![Figure 10: Mapping Design to Topology](image)

Scheduling Map.

The scheduling map links topology elements to elements from the operating environment, which is one of the maps that includes a transformation from PIM to PSM. This way the topology model is enhanced with concrete information about scheduling, operating system, and bus protocol. In case of a one processor architecture, it is also possible to implement the scheduling map by mapping the design directly to the operating environment.
Architecture Map.

Finally, the architecture map is the other map that includes a PIM to PSM transformation. It associates topology elements with hardware elements. This map is the last part of a transitive mapping (design-topology-hardware). It maps I/O, nodes to processors, and links to network segments. The node map assigns nodes to concrete processors on the hardware. Together with the application map, it is defined which process will run on which processor. The link map associates links to network segments and thus, specifies the concrete path for every single message.

5. PLATFORM INDEPENDENT DESIGN

By separating design, topology, operating environment, and execution environment, the platform independence of Realtime Java or any other ANRTS language can be designed for at the modeling level.

The most common use case is to take a Java program that has been designed to run on one operating system or processor architecture and move it to another. The program will run, but the timing behavior may well be different. Since the operating environment and the computational environment are separate from the design and topology, the changes needed are limited to one of these domains respectively, thus limiting revalidation effort and cost. The developer needs only recreate one of these domains and the maps between it and the design and topology.

Similarly, a design can be moved from a single processor to a network of processors or vice versa. The designer creates a new topology and then remaps the processes to new nodes and messages to new links in the topology. Of course, new code will need to be generated at the node boundaries to support non-local access through some mechanism such as JNI or CORBA. Once this is done, the design can be revalidated in the new configuration. Reducing the number of nodes in the system can be done likewise.

Mappings enable managing and reusing each modeling aspect independently of the other aspects. The same design can be reused on different topologies. Each topology can support many different designs. The same applies to operating and computational environments.

6. CONCLUSION

Using Realtime Java Technology for building ANRTS requires more flexibility than current MDA approaches offer. Refinement by model transformation forces a linear discipline on model development. This makes changing one part of the model, e.g., the operating system, while leaving the rest unchanged, more difficult. By using modeling aspects bound together by maps, one can more easily exchange a given aspect without altering the rest of the model. This is particularly useful for ANRTS systems based on Java Technology, where code generation can be done directly from the PIM and is the crucial advantage of this profile.

7. FUTURE WORK

The profile defined here will be used in the context of the SuReal project to use realtime Java Technology to build robust ANRTS systems. The profile will be the starting point for a full development toolchain including code generation, model verification, timing and memory analysis, and code validation. The profile will be augmented with a library facility to further simplify redesign.

8. ACKNOWLEDGMENTS

Some of the funding for this work was provided through the SuReal project sponsored by the German Government (BMBF) for which the authors are grateful.

9. REFERENCES


