UML Based Design of Time Triggered Systems

István Majzik, Gergely Pintér and Péter Tamás Kovács
Budapest University of Technology and Economics
Dept. of Measurement and Information Systems
Magyar Tudósok krt. 2., 1117 Budapest, Hungary
{majzik,pinterg}@mit.bme.hu, kovacsp@sch.bme.hu

Abstract

This paper presents how the platform-specific development environment of time-triggered (TT) systems can be integrated with a visual design toolkit based on UML. The built-in facilities of UML and the modeling extensions introduced by us enable the unification of the advantages provided by both the embedded development environment and the UML tools. UML offers visual design, automatic code and documentation generation, while the underlying TT development environment offers platform-specific task and communication scheduling and fault tolerance middleware construction. This results in an integrated system that is capable of supporting the entire development within the framework of the UML tool.

1. Introduction

Embedded systems are typically applied for performing tasks closely related to the hardware, i.e. they can not expect a programmer-friendly multi-layered virtual machine interface and operating system API. This direct connection to the hardware impacts the fundamental implementation aspects (e.g. the interface design), as well as the dependability and timing considerations [3]. Fault tolerance measures can involve redundant communication channels, modular hardware redundancy, software diversity, etc. As a special aspect, real-time operation can be required from the embedded system. In these cases sophisticated task and communication scheduling and special algorithm design are needed [5].

These demands are reflected in the development environments offered for these systems. The fine-tuned tools and techniques enable the development of highly efficient systems exploiting most of the platform-specific features.

Visual modeling languages receive more and more attention also in the area of embedded and real-time systems [2]. Their application is motivated by the increased productivity due to the easy-to-understand visual notation that helps in finding a common language between developers and users, provides a structured presentation of ideas and design decisions, and forms the basis of automated documentation and code generation. One of the best known (object oriented) visual modeling languages is the Unified Modeling Language (UML) [1] that is a standard notation for software visualization. Designers are becoming more and more familiar with UML and UML-based CASE tools.

In this paper we show how the platform-specific development tools of the Time Triggered Architecture (TTA), a fault-tolerant distributed real-time architecture [4], can be interfaced to a UML-based visual design toolkit. The design of the embedded applications is supported by UML language extensions and a transformation that provides the specific input format of the embedded development environment. Our approach promises this way the unification of the advantages of these different development methods. Namely, UML offers a widely known visual design language, external model checkers, fault modeling [10], automatic documentation and code generation [11], while the TTA development environment offers real-time scheduling, support of the platform-specific operating system and fault tolerance middleware.

2. UML and domain-specific languages

UML offers extensions that support the creation of domain-specific modeling languages. The extensions customize the generic elements available in UML by using stereotypes, constraints, and tagged values. A coherent set of such extensions, defined for a specific purpose, constitutes a UML profile [6]. By using these standard extensions, the basic concepts of the language, the well-defined syntax and the semantics are kept and thus the existing tool support can be directly utilized.

\* This work was supported by the Bolyai Fellowship of the Hungarian Academy of Sciences and the European IST project NEXT TTA.
An example of a standard profile is the UML Profile for Schedulability, Performance and Time Specification [9]. This specifies the domain viewpoint (the structural and behavioral concepts) of real-time systems and their analysis methods, and specifies how the domain concepts are realized in UML. Most of the domain concepts are abstract and are intended to be refined in sub-profiles. Our proposed TTA Profile can be considered as such a refinement. We capture the domain concepts of TT systems and introduce the extensions that represent these concepts. By using the profile we get platform-specific analysis models that can be transformed to the input format of the existing development tools like the TTA message and task scheduler, the middleware configurator and the download manager.

In the following, first the domain concepts, i.e. the elements of the design space of the target platform are enumerated, together with their relations and usage constraints. They become the elements of the metamodel of the platform-specific visual language. This metamodel is considered as an extension of the standard metamodel, and the profile is constructed accordingly. On the basis of the profile, UML tools are integrated with the TTA tool set.

3. The Time Triggered Architecture

The Time Triggered Architecture provides a computing infrastructure for the design and implementation of distributed real-time embedded systems [4]. It defines hardware and software architecture and a corresponding design methodology. The most important characteristic of a time triggered system is that all activities (including communication and task execution) are initiated at predefined points in time. This approach determines the communication scenario and the software architecture of the TTA.

From the hardware point of view, a TTA cluster consists of individual hosts interconnected by a redundant bus or a redundant star topology network. Each host is a standalone computer that has a specific communication controller (CC) attached.

The communication in a TTA cluster is based on the time triggered protocol TTP/C. The CC decides autonomously when a message is transmitted to other hosts. The timeliness of message transmissions is guaranteed by the design-time allocation of communication slots for each host (time division multiple access to the physical media). The schedules belonging to different cluster modes (operational modes) are stored in the Message Description List (MEDL), which is loaded into each node’s CC before application start. The CC is also responsible for maintaining a common time base and membership information inside the cluster.

The software architecture of a TTA cluster consists of system and application tasks and subsystems. Application tasks are responsible for processing the messages arriving on the TTP bus as well as I/O messages originating from other sources. Messages have specific types. System tasks perform system-level duties. A subsystem consists of a set of tasks that work on the same mission, i.e. take some input and produce output in a given application mode. The scheduling of tasks is determined at design time (static scheduling). The architecture supports redundancy as tasks can be replicated on different hosts in the cluster. The management of the replication, e.g. error detection and agreement among replicated messages is solved by the system tasks that form a fault tolerant communication layer (FT-COM).

Due to the strict time triggered approach the TT communication system is composable with respect to temporal control. Accordingly, the system-level perspective of the cluster design and the implementation-centric level of node development are clearly separated. The two-layered design method is supported by the following development tools:

- The cluster design tool TTPplan [12] is used to define the cluster, namely the hosts, subsystems, messages, message types and cluster modes. Numerical attributes to be specified include the length of a round, the transmission period of messages etc. According to these information, the MEDL can be constructed.
- The node design tool TTPbuild [12] is used to construct the internal behavior of a host according to the MEDL computed at cluster level. Tasks building up subsystems are identified and the operating system configuration and FT-COM layer are generated. The produced sources are available as program code skeletons that are to be completed by the programmer with application specific routines.

4. The UML profile for the TTA

The domain specific model of the TTA contains the platform elements identified above. According to our methodology, the necessary metamodel elements are introduced straightway as extensions (descendants) of standard UML metamodel elements, like Classes and AssociationClasses, further customized in [9] as SchedulableResource, Processor etc. Two types of extensions are introduced here intuitively as follows:

- Classes are used when a concept within the system is being modeled. In this way clusters, hosts etc. are to be represented in the visual model as instances of a corresponding customized UML Class. The properties are carried by the attributes and tagged values of these classes. Based on this extension, for example, a host is modeled as an instance of Host (that is a descendant of Class) in the visual model.
Association classes are used when an association itself has class properties. For example a host in a cluster can be synchronized to the other hosts or not. This distinction is represented by a Boolean attribute of the association class Host in ClusterLink connecting the cluster and the host.

The standard extensions of UML can be used for representing the domain concepts in the following way:

- Stereotypes are used to virtually define the new modeling elements (i.e. virtual subclasses of UML metaclasses) by assigning different intentions, or usage distinction, to existing ones. For example, a stereotype Cluster is defined that can be attached to an instance of Class in the model. The stereotyped element is considered as an instance of the metamodel element Cluster. The new semantics of these model elements are clarified by tagged values and additional constraints.
- Tagged values are keyword-value pairs that can be used to attach arbitrary properties to the model elements.
- Constraints are used to specify those conditions and propositions that must be maintained when the model is constructed.

In our case, the set of stereotypes is defined according to the metamodel extensions demonstrated above. The constraints attached to the stereotypes specify well-formedness rules that must be observed by all model elements branded by that stereotype. Basically, two categories of constraints are used:

- The associations between stereotyped model elements are prescribed.
- The required attributes or tagged values of model elements are specified by invariants formalized in the standard Object Constraint Language (OCL) of UML.

For example the rule “classes stereotyped with Cluster must have a feature named transmission_speed” (which defines the speed of the TT bus) is formalized as follows:

```
Context Class inv:
self.stereotype.name = "Cluster" implies
self.feature-> select (name = "transmission_speed") ->notEmpty()
```

By using the stereotyped model elements, platform-specific models can be built in commercial UML tools. OCL constraints interpreted by advanced modeler tools can be used to force designers to satisfy the rules when the profile is applied.

The separation of the concerns of cluster design and node design are maintained by the following conventions: The cluster design is the root package of the UML model, while the node designs are placed into separate packages. It is also allowed that the node designs are constructed in separate UML models that import the model elements of the cluster design, in this way cluster level elements can be referenced.

5. Integration of the tools

The UML visual modeling toolkit and the embedded development environment are integrated by using XMI (XML Metadata Interchange [7]), the standard interchange format of UML models. Based on the TTA profile, a transformation is defined and implemented that processes the XMI output of the UML tool, extracts the relevant information and feeds the model repository of TTPplan and TTPbuild by using their respective programming interfaces. On the basis of the repository imported from the visual model, TTPplan can produce the communications schedule (MEDL) while TTPbuild is used for generating the OS configuration and the FT-COM layer (Figure 1). The resulting code fragments, together with the program code of the task functions, can be compiled, linked and downloaded to the hosts of the cluster.

![Figure 1. Visual cluster and node design.](image)

In our prototype implementation, Rational Rose was used as the UML CASE tool. The limitations of Rose, namely that it cannot interpret the constraints formalized in OCL, were resolved by a set of scripts (written in RoseScript, the internal scripting language of the tool) that help
the designer to apply the conventions and satisfy the well-formedness rules.

Using the toolkit of UML dynamic behavior can be specified by assigning statecharts and activity diagrams to classes. Since the Task metaclass in our profile is derived from Class, this control-flow description facility is inherited enabling the developer to attach these diagrams to tasks as visual representation of their internal behavior. Since application tasks in TTA are single threaded, time triggered (i.e. not event-driven), non-preemptable, statically scheduled functions started by the operating system, activity diagrams are more suitable for specifying their operation. We support the UML based design of TT tasks by automatic source code generation, i.e. programming language level instantiation of activity diagram elements by a code generator that is capable of producing ANSI C source containing TTP-OS system calls.

The visual modeling is demonstrated in Figure 2 by the UML based cluster design of the sample break-by-wire application provided with TTPplan.

![Figure 2. Cluster design of the example application.](image)

6. Conclusions

The profile introduced in this paper allows the designer to assemble TTA cluster and node designs in the form of UML class diagrams and specify task behavior by using UML activity diagrams. On the basis of the structural models the platform-specific development environment is able to construct the OS configuration and the middleware layers, while the behavioral diagrams can be used by a customized code generator to construct program code.

The recent initiative of the OMG is the Model Driven Architecture (MDA) [8] that aims at the partitioning of the design process into a specification of logic functionality resulting in a Platform Independent Model (PIM) of the target application, followed by a mapping to a Platform Specific Model (PSM) and subsequently to the target code. Our approach fits to MDA by defining the format of the PSM in the case of the TTA. It is a task of future research to examine the feasibility of a potential mapping from a PIM to this PSM.

References