An Overall System Design Approach Doing Object-Oriented Modeling to Code-Generation for Embedded Electronic Systems

Clemens Reichmann[1], Markus Kühl[2], and Klaus D. Müller-Glaser[1]

[1] Laboratory for Information Processing Technology (ITIV)
University of Karlsruhe,
Engesserstr.5, 76128 Karlsruhe, Germany
{Reichmann,Mueller-Glaser}@itiv.uni-karlsruhe.de
http://www.itiv.uni-karlsruhe.de
[2] Research Center for Information Technology,
Haid-und-Neu-Str. 10-14, 76128 Karlsruhe, Germany
Kuehl@FZI.de
http://www.fzi.de

Abstract. In this paper a new approach for an overall system design is presented. It supports object-oriented system modeling for software components in embedded systems in addition to time-discrete and time-continuous modeling concepts. Our approach provides structural and behavioral modeling with front-end tools and simulation/emulation with back-end tools. The UML metamodel is used for storing CASE data in a MOF object repository and XMI (XML Metadata Interchange format) is used to interchange this data with UML-CASE-tools. The CASE tool chain we present in this paper supports concurrent engineering including versioning and configuration management. It provides adaptors for the tools MATLAB/Simulink/Stateflow[1] and ARTiSAN Real-Time Studio[2] as well as an importer/exporter of UML/XMI. Utilizing the Unified Modeling Language notation for an overall system design cycle, the focus of this paper lies on the subsystem coupling of heterogeneous systems and on a new code generation approach.

1 Introduction

The design of embedded electronic systems has changed due to a broad introduction of the object-orientation paradigm. It is also characterized by rapidly increasing complexity and shorter product cycles today.

[1] MATLAB/Simulink/Stateflow is registered trademark of Mathworks, Inc.
[2] Real-Time Studio is registered trademark of ARTiSAN Software Tools, Inc.

M. Pezzè (Ed.): FASE 2003, LNCS 2621, pp. 52–66, 2003.

With the introduction of the Unified Modeling Language (UML) as a standard [6] and well-accepted notation for software design, the object-orientation paradigm is more and more accepted and supported in commercial CASE tools.

These facts are results of a trend towards increasing use of more software components in typical embedded system applications. Systems in automotive applications tend to have numerous software subsystems, event-driven subsystems and often a control subsystem in the time-continuous domain. These systems usually perform tasks like diagnosis, self-calibration, encrypted data-communication, and even database-oriented system processing.

A commonly used approach to handle system complexity in the system design process is called concept-oriented rapid prototyping[3] [1,2]. Concept-oriented rapid prototyping defines a way for the fast conversion of an executable specification, which captures the requirements by using modeling techniques for state-event, time-continuous, and software modeling, into a functional prototype. This prototype mainly serves to clarify system goals and must support a widespread range of hardware interfaces. It uses automatic code generation based on CASE tools like MATLAB/Simulink, ASCET-SD[4] or Statemate[5], and powerful, general-purpose, extensible hardware. The cost of rapid prototyping hardware is not critical, as the hardware is not specific and can be reused for different prototypes.

Up to now concept-oriented rapid prototyping was influenced by tools for Computer-Aided Systems Engineering (CASE) using the state chart theory (modeling of hierarchical finite-state-machines) or control systems engineering using block diagrams. In the past much effort was spent to improve design tools for a better integration of state charts and block diagrams. Current requirements in embedded systems design shift the attention from a pure state chart and block diagram approach to a more software-oriented view, which is captured by modern CASE tools in Software Engineering domain. A major problem, common to all commercial software CASE tools, is a lack of support for control systems engineering whereas modeling of time-discrete or state-oriented systems is well supported. Therefore, development of mixed-domain systems with both time-discrete and time-continuous subsystems and additional software components is not a continuous design process today. Thus there is a need to unify the description of the model to a single notation based on one metamodel to enable an overall design technique. Furthermore the integration of the domain-specific parts in terms of code generation, data and message exchange, task distribution, model versioning, user management and automated transformation between the domains is important. In our work the designer has the possibility to model in different modeling domains (time-discrete, time-continuous, software) and notations, which are transferred to one metamodel. This is done to achieve an executable specification running on a rapid prototyping platform.

In the following section we will summarize the related work for concept-oriented rapid prototyping (see Sect. 2). We will introduce the metamodel used in our approach

[3] Not captured in this paper: architecture and implementation rapid prototyping (RP)

[4] ASCED-SD is a registered trademark of ETAS GmbH

[5] Statemate is a registered trademark of i-Logix, Inc.

in Sect. 3. Section 4 will present our universal object-oriented modeling approach for embedded electronic systems. Then, in Sect. 5 our current work to provide a subsystem coupling technique on model-layer combined with a new automatic code generation approach (see Sect. 6) will be introduced. Afterwards, *GeneralStore*, our client/server CASE tool integration platform for mixed-domains and concurrent engineering, is described. Finally, Sect. 8 discusses results and offers a conclusion of the topic.

2 Related Work

Commercial solutions that support object-oriented modeling of embedded electronic systems are rare. Today, a wide range of CASE tools for object-oriented analysis and design is available mainly supporting pure software modeling. Software CASE tools for embedded systems modeling are often new to market. Only three well known CASE tools support object-oriented analysis and design using UML notation in addition to time-discrete modeling (e.g. state charts): ARTiSAN Real-Time Studio, i-Logix Rhapsody, and Rational Rose Realtime are tools explicitly classified for this domain.

A major drawback of those CASE tools is the lack of modeling concepts for control system engineering. For an overall system design users will have to include source-code as external C-code into these CASE environments. This approach is expensive and very rigid. Communication between both modeling domains (software to time-discrete/ -continuous system parts) is done via source code coupling. In large-scale systems, changing the system architecture during the development process is very susceptible to errors. Therefore, model-based coupling of software components and time-discrete/continuous system parts is desirable.

3 Metamodel

In our approach the whole system is described as an instance of one particular metamodel in one notation. This model has to cover all three domains: time-discrete, time-continuous, and software. The Unified Modeling Language (UML) is an Object Management Group (OMG) standard [6] which we use as system notation and metamodel. It is a widely applied industry standard to model object-oriented software. The abstract syntax, well-formedness rules, Object Constraint Language (OCL), and informal semantic descriptions specify UML.

The UML specification provides XML Metadata Interchange format (XMI) [8] to enable easy interchange of metadata between modeling tools and metadata repositories in distributed heterogeneous environments. XMI integrates three key industry standards: the eXtensible Markup Language (XML), a W3C standard, the UML, and the Meta Object Facility (MOF) [7], an OMG metamodeling standard which is used to specify metamodels.

One key aspect of UML is the four layered metamodeling architecture for general purpose manipulation of metadata in distributed object repositories (see Fig. 1) which makes it suitable for our universal object-oriented modeling approach. Each layer is an abstraction of the underlying layer with the top layer (M3) at the highest abstraction level. On the M-1 layer, which is not part of the 4-layer architecture, there is the execution code of the program. The M0 layer is comprised of the information that we wish to describe (the data). This is the source code in different languages, e.g. JAVA or C++. On the model layer (M1) there is the meta-data of the M0 layer, the so-called model. Object-oriented software is typically described on the M1 layer as a UML model. The metamodel on the M2 layer consists of descriptions that define the structure and semantics of meta-data (e.g. the UML model). These are the metamodels, e.g. UML 1.3, UML 1.4, and define the language respectively notation for describing different kinds of data (M1). Finally on the M3 layer there is the meta-meta-model. MOF is used to describe meta-models and define their structure and semantic. It is an object-oriented language for defining meta-data. MOF is self-describing. In other words, MOF uses its own metamodeling constructs.

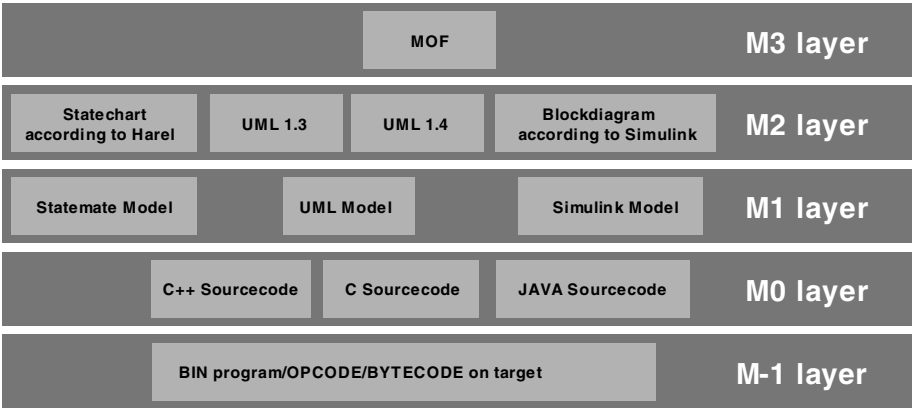

Fig. 1. 4-layer metamodeling architecture

XMI was partially influenced by the ideas for a tool-independent CASE data interchange format called CDIF [3], which was based on entity-relationship (ER) descriptions. CDIF addresses the problem of model data interchange between CASE tools.

Without a standardized interchange format for integrating more than one CASE tool, proprietary import/export filters must support the exchange of model data. In addition, new interfaces have to be implemented for tool integration. XMI is supported in a wide range of industry applications. In the machine tool domain for example, STEP, a standard for the exchange of product definition/model data, will be compatible with XMI in the future.

In the current version 1.4 of the UML standard it is possible to completely interchange model information. Nevertheless, it is not yet possible to interchange the graphical views of the model in terms of diagrams, which will be supported in the forthcoming UML 2.0.

4 Integration on Model Level

When we capture current requirements in the design process of embedded real-time systems it is necessary to handle time-continuous, time-discrete, and software design techniques. Besides the problem of using a large number of description notations/methods, an enormous amount of design data has to be handled. Another problem in many commercial CASE tools is the lack of concurrent engineering support. Only a project file based datastore is offered that can be edited only by one user at a time.

To couple different notations of the domains, transformation rules are necessary. In our approach we support different notations used by various CASE tools to model in specialized domains. The designer will choose the best tool/notation for a sub-problem and integrate the solution into the UML top-level metamodel. The software domain is modeled in UML therefore no transformation is needed. In the time-discrete domain the UML provides a notation but with the lack of well defined semantics. Here we use the semantics of David Harels concurrent hierarchical state charts [12] implemented in the CASE tool Statemate. It has an XMI interface and can interchange the time-discrete model with our integration platform *GeneralStore* (see Sect. 7).

The main difficulty when using different description domains in a complex embedded systems design is the integration of control subsystems. There are two possible solutions to overcome this problem:

1. Integration of the time-continuous subsystem using the reverse engineering mechanism of modern CASE tools for software engineering. This case is called “subsystem coupling on source code level”. One major problem here is that the control subsystem is shown as a black box subsystem encapsulated inside a class with the loss of information for other designers (see top of Fig. 2).
2. A bidirectional transformation rule (see bottom of Fig. 2) enables the synchronization of the time-continuous subsystem to a representation in the UML notation that uses simple object- and class diagrams (see left of Fig. 3). With an automated CASE tool-coupling layer, this transformation is reversible. This technique is called “subsystem coupling on model-layer”.

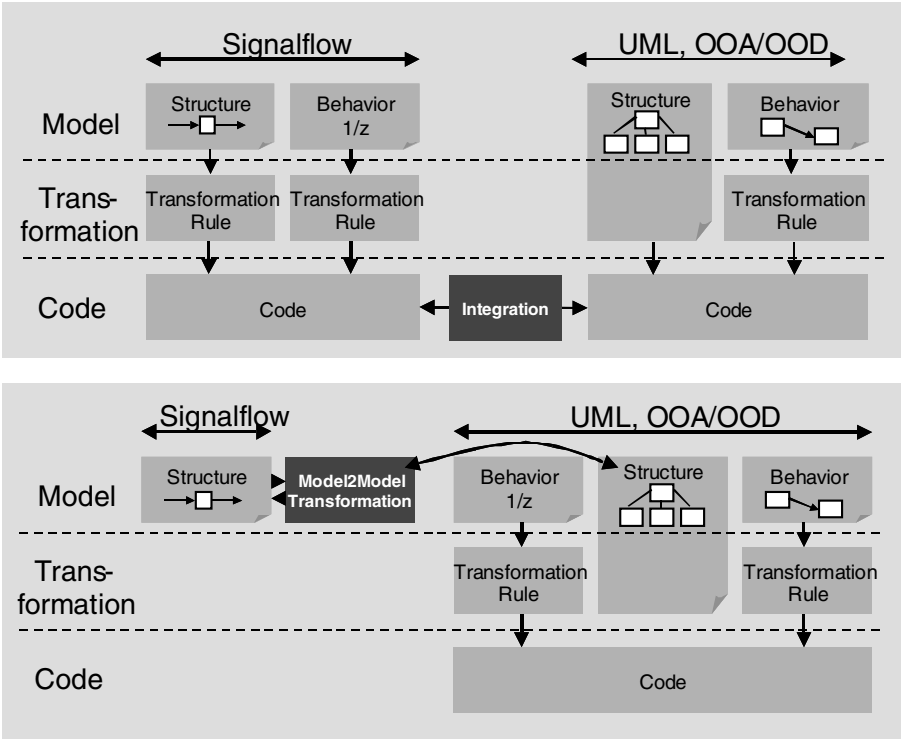

Fig. 2. Subsystem coupling on source-code level (top) and on model level (below)

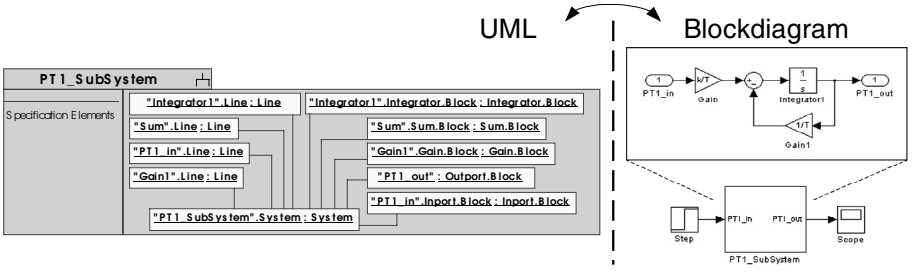

Fig. 3. Transformation of a block diagram to UML

Being more transparent the second approach is the more suitable way for a convenient integration of both modeling domains. Figure 3 shows a small footprint of the design process, which is formed by a so-called bidirectional transformation rule. Simplified, blocks in the block diagram are translated to objects in the UML metamodel. Connections represented by lines are also UML objects. The hierarchy in the block diagram is achieved by UML subsystems and links between objects. Parameters of the

blocks are slots on objects. The classifications of objects are classes whereas links are instances of associations. These elements are located in a separate UML package. This transformation rule is the result of a long-term scientific project at the Laboratory for Information Processing Technology (ITIV) at the University of Karlsruhe. We have currently developed a CASE tool integration environment (*GeneralStore*, see [4,5] and Sect. 7) that provides the necessary underlying design process support that will be introduced next.

5 Design Process

From our point of view the design process starts with object-oriented analysis using an UML CASE tool where we model use-cases to catalog the requirements. Then each requirement of the system specification is translated to a scenario modeled with a message sequence chart, state machine, or activity diagram. Non-functional requirements are modeled as OCL[6] constraints or added as comments. By doing so an initial class model is automatically introduced.

At the next step the developer arranges the class model in various class diagrams. These diagrams have to be refactored. Generalizations are identified and similar classes have to be combined. Dependencies and associations are revealed. With these steps the so-called analysis-model is achieved. This first analysis model is not in the main focus of our work but illustrate the overall process.

The model of the embedded system then has to be divided in parts, where each part is mentioned as a software component, a subsystem in the control-domain, or a state chart in the time-discrete domain (see Fig. 4).

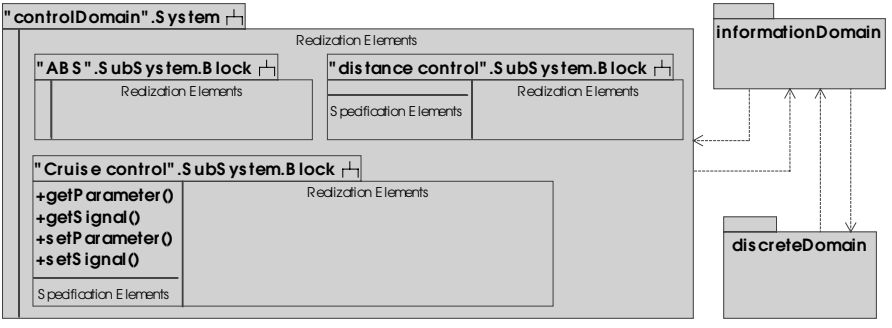

Fig. 4. Different domains as UML subsystems

[6] Object Constrain Language an OMG standard

Developers of different domains automatically select different notations and metamodels to describe the problem of the subsystem when using best fitting CASE tools. The integration platform *GeneralStore* applies automated transformations on these notations to convert them to the top-level UML metamodel (see Sect. 4).

Specialists in control system design use their notations (block diagrams) and tools (e.g. MATLAB/Simulink) to model in their domain. For integration with the overall system the control subsystem is transformed to the UML metamodel and vice versa. Thus we have a white box integration of the model in UML notation (compare Fig. 3 and Fig. 5 right).

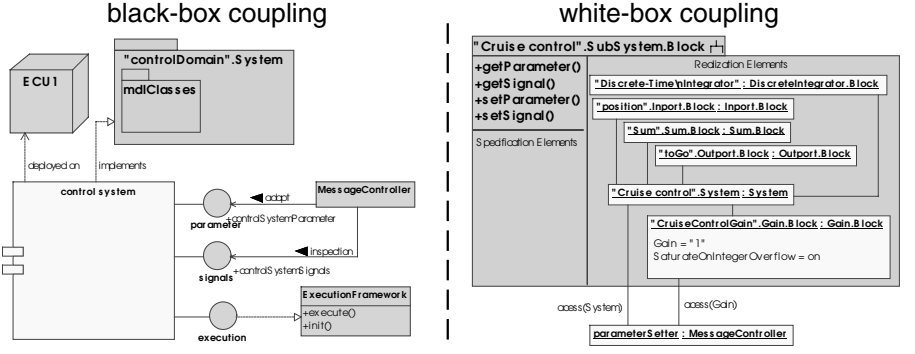

Fig. 5. Control system in UML view: black-box coupling (left) and white-box coupling (right)

Specialists in the software domain can model their subsystem and the interface to the control-domain in UML (see Fig. 5). In UML we can model the task distribution (see bottom part of Fig. 6), scheduling and message exchange of the heterogeneous system. In Fig. 5 there is the component *control system*, which is running on the electronic control unit *ECU1*. This component encapsulates the control domain, or more exactly its implementation. The *parameter* interface (see Fig. 5 left) allows manipulation to the parameters of the control system, whereas the *signals* interface provides access to the internal state of the control system, e.g., for diagnostic reasons. The *execution* interface is used for scheduling and task allocation of the control domain. The component is the black-box view of the control system.

In addition the designer can use the white box view (see Fig. 5 right) of the control system to model the access to the signals or parameters explicitly or in a generic way. On the right hand side of Fig. 5 the object *parameterSetter* has access to the parameter *Gain* of *CruiseControlGain*, which corresponds to the block *Gain* in the block diagram (see Fig. 3).

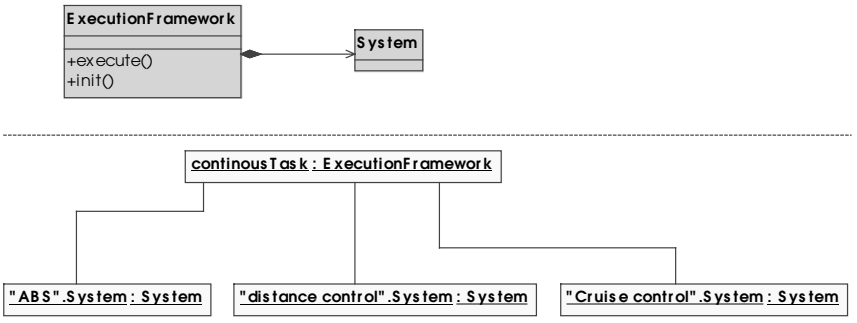

Fig. 6. Class diagram of execution modeling (top) and modeling task distribution (bottom)

The shown design pattern for domain coupling is based on an automatically generated wrapper of the control and time-discrete domain. This is done by inspection of the generated code of the supported commercial code generators. The wrapper is generated on UML model level and is transformed to code with our code generator (see Sect. 6) during code generation phase. The control domain wrapper provides access to parameters and signals. The other partitions are built with commercial generators and, as stated before, the whole system is generated to get an executable specification.

Although there are state machines in the UML metamodel, the semantic is not completely specified. But for an executable specification we absolutely need well-defined semantics of state charts. To solve this problem designers use the Harel state chart semantics [12] implemented in the CASE tool Statemate or Rhapsody in C to model the time-discrete subsystem. The coupling of the time-discrete model to the other domains is accomplished using a design pattern similar to the control-domain described above.

Now we can run the generated system on the RP[7] target. Usually, real systems will not fulfill the expected tasks completely at the first try. To find errors we use model monitoring. Test cases are modeled in UML, generated by our code generator, and run in a unit test mode on the target. After recording the whole scenario of the failed test case it can be shown in the UML model offline when the user is stepping through it.

6 Code Generation

There are highly efficient commercial code generators on the market. In safety critical systems certificated code generators have to be used to fulfill the requirements. The *GeneralStore (GS)* platform allows partitioning the whole system into subsystems to invoke different domain specific code generators.

[7] Rapid Prototyping

For control-systems there are commercial code generators like Target Link[8], Embedded Coder[9] or ECCO[10]. In the time-discrete domain we utilize the code generator of Statemate[11] (Rhapsody in MicroC). In software domain commercial code generators only generate the stubs of the static UML model while behavioral functionality has to be implemented by the software engineer.

As we focus on a completely generated executable specification it is necessary to generate code of the overall model. Therefore we provide a code generator as a *GS* plug-in to enable structural and behavioral code generation directly from a UML model. The body specification is done formally in the Method Definition Language (MeDeLa). Which is an abstract action language based on Java syntax. Our template code generator is using the Velocity engine to generate Java or C++ source code. Velocity is an Apache open source project focused on HTML code generation. It provides macros, loops, conditions, and callbacks to the data model business layer.

The different domains have interactions, e.g., signal inspection, adoption of control system parameters at runtime or sending messages between time-discrete and software artifacts. There should be one scheduler on each ECU[12] as many commercial code generators provide a specific scheduling mechanism. Integration of these different frameworks on one ECU is error prone. In our approach these design tasks could be modeled in UML with assistance of MeDeLa as outlined in Sect. 5. Afterwards our code generator renders the specified glue (interactions between domains). We provide a highly flexible modeling process, which is supported by an integration platform described in the following section.

7 Client/Server Architecture for CASE

To keep the system model manageable for designers, CASE tool integration is necessary. In [1,2] an open design environment based on CDIF for the development of mechatronic systems was presented. Many of the experiences from this project influence our current work.

After the design of prototypes during the last two years the focus of our current work is on a so-called *Integration Platform GeneralStore (GS)*. The setup of the *GS* follows a 3-tier architecture well known in the software engineering domain. On the lowest layer (database layer) a commercial object-relational database management system ORACLE[13] respectively MySQL[14] was selected. On the business-layer we

[8] From dSPACE GmbH
[9] From Mathworks, Inc.
[10] From ETAS GmbH
[11] From i-Logix, Inc.
[12] Electronic Control Unit
[13] ORACLE is registered trademark of ORACLE Corporation
[14] MySQL is an open source project: see www.msql.com

provide user authentication, transaction management, object versioning, and configuration management.

For managing CASE data, *GS* supports UML as its metamodel (see Sect. 2). *GS* uses MOF as its database scheme for storing UML artifacts. Inside the database layer an abstraction interface keeps *GS* independent from the given database.

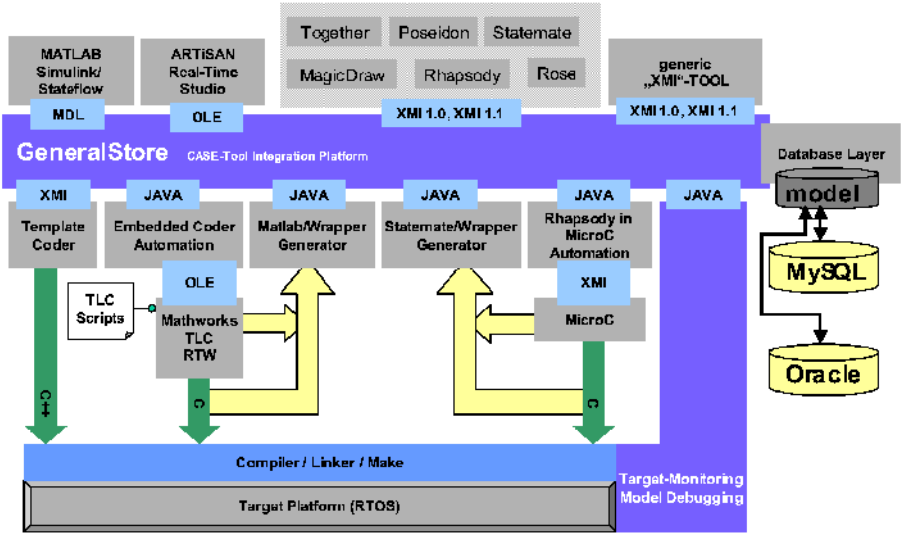

Fig. 7. Architecture of integration platform *(GeneralStore)*

On the business layer of *GS* the mediator pattern [11] is used to keep the CASE tool integration simple and its adaptor uniform. The transformations for specific notations supported by CASE tools are implemented in plug-ins (see top of Fig. 7). The code generation plug-ins (Embedded Coder and Rhapsody in MicroC) controls the transformation to the source code. Their wrapper generators are automatically building the interface in the UML model (see Fig. 7 in the middle).

While interim objects enclose MOF elements, the CASE adaptor (see class *GSTool* in Fig. 8) stays thin and highly reusable. Another reason for using interim objects on the business layer is because of object identity to handle object versioning and configuration management. To visualize these circumstances, an example is taken where a unique identification number is added to a subsystem block from the time-continuous system domain. After the block is checked out from the repository, a designer moves this subsystem to another hierarchy layer. Despite the fact that it is the same compound object, the system controller in the time-discrete system part keeps track of this link because of the subsystem identification number.

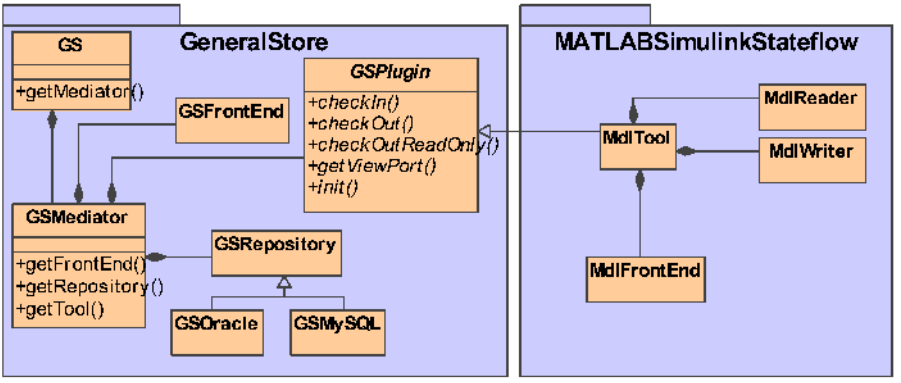

Fig. 8. Plug-in architecture of *GeneralStore*

On the presentation layer *GS* provides three principal CASE tool adapters:

- MATLAB/Simulink/Stateflow was selected for the control system design domain and the integration is done using the proprietary model file.
- Generic and specialized XMI importer/exporter filter of **.xmi*[15] files.
- And the COM[16] based integration of ARTiSAN Real-Time Studio. The tool was selected because of its focus on embedded real time systems.

Both CASE tools are bidirectional linked to the *GS* architecture. For model management and CASE tool control, *GS* offers a system hierarchy browser (compare Fig. 9). Since the internal datamodel representation of *GS* is UML, *GS* offers a system browser for all UML artifacts of the current design. Due to the large amount of MOF objects (for example: the transformed PT1 subsystem needs about 10,000 XMI entities), *GS* offers design domain specific hierarchy browsers, e.g., a system/subsystem hierarchy view for structural or time-continuous design or a package hierarchy view for software design.

8 Conclusion

The introduced universal object-oriented modeling approach supports the concurrent development of electronic systems in all design phases. We showed how heterogeneous system descriptions could work as integrated parts of an object repository based-Client/Server CASE tool environment. Based on an object diagram representation,

[15] XML file based
[16] Microsoft Component Object Model

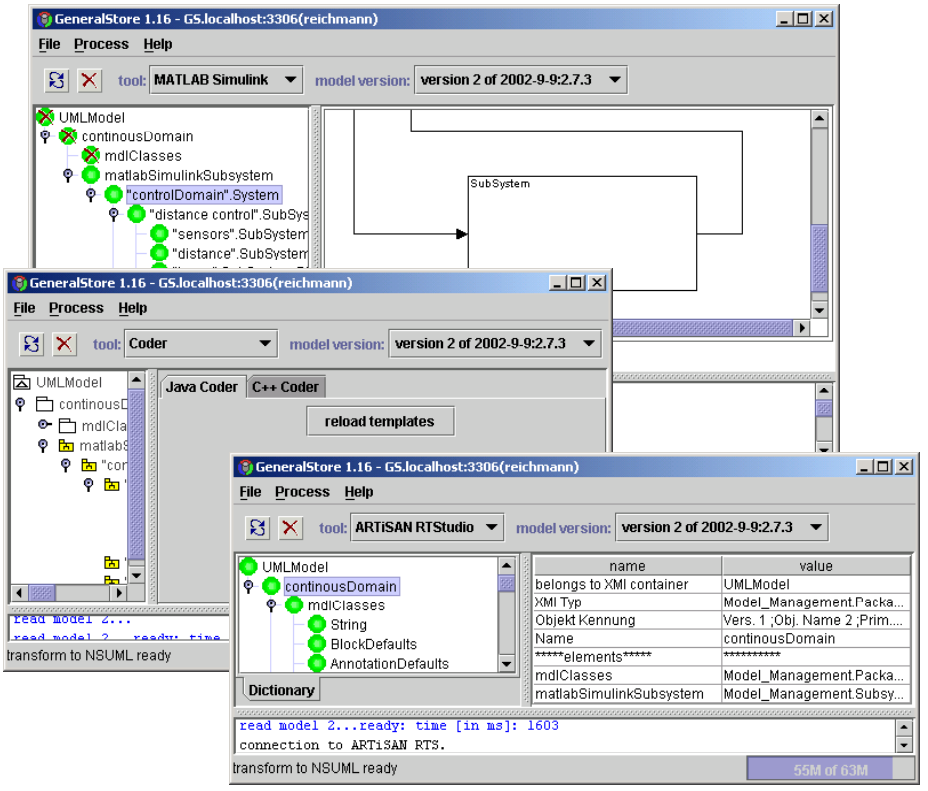

Fig. 9. *GeneralStore's* MATLAB, Coder, and ARTiSAN UML plug-in

time-continuous subsystems and software components can be modeled using one single description style. A direct linkage between different description domains is possible on an abstract model level. While transforming all subsystem parts to an uniform object notation, adding additional model information to time-continuous blocks will enable system designers to start with system simulation early in the design process.

Embedded electronic systems can be subdivided in time-discrete, time-continuous and software domain. Each domain uses its specific notation. A highly flexible design process was described to integrate those notations, which are supported by CASE tools, to one UML model. The glue between the domains is modeled in UML. Finally the overall system is transformed to source code with the assistance of commercial code generators in addition to our UML code generator.

An often-needed feature and simultaneously a drawback of project file-based CASE tools (e.g. Rhapsody, Simulink/Stateflow) is the lack of CASE tool assisted concurrent engineering. Using the presented CASE tool backend *GeneralStore* together with MATLAB/Simulink, an interim project file is created each time a designer

checks out a part of the model. This is possible at any specific model hierarchy point. The checked out subsystem hierarchy becomes protected. Other designers still have read access to the last version of this subsystem and can obtain read/write access to other subsystems in the time-continuous hierarchy.

One major drawback of using UML/MOF from XMI as a metamodel for system description is the deficiency of a standardized graphical representation for class and object diagrams. Up to now, this is one of the most requested topics for UML 2.0 and the next XMI generation. On the other side, using XMI and the UML metamodel for the description of embedded systems enables model exchange with other CASE tools. Today, at least 10 software CASE tools on the market can handle XMI descriptions from the information point of view (import of a model without graphical description).

9 Outlook

Future work has to focus on the definition of a tailored design process and the integration of CASE tools for requirements management (e.g. DOORS[17]). On the back-end side of our development environment simulation and emulation of system models is going to be supported.

We will examine the integration of further modeling tools to estimate the saved integration effort using UML/MOF. Especially for large systems of different design domains, our universal object-oriented modeling approach for the design of embedded electronic systems based on MOF will show its advantages.

Furthermore it is planned to work on enhanced modeling of operations in UML for code generation, e.g., collecting information from activity charts, object collaboration diagrams, and state charts.

It is also considered to enable modeling for reconfigureable hardware components, e.g. FPGA[18]s that means creating VHDL code from an UML model.

References

1. A. Burst; M. Wolff; M. Kühl; K.D. Müller-Glaser: A Rapid Prototyping Environment for the Concurrent Development of Mechatronic Systems, ECEC, Erlangen, Germany, 1998.
2. A. Burst; M. Wolff; M. Kühl; K.D. Müller-Glaser: Using CDIF for Concept-Oriented Rapid Prototyping of Electronic Systems, RSP, Leuven, Belgium, 1998.
3. EIA / CDIF Technical Committee: CDIF / CASE Data Interchange Format. EIA Interim Std. EIS / IS-106–112, 1994.
4. M. Kühl; C. Reichmann; B. Spitzer; K.D. Müller-Glaser: Universal Object-Oriented Modeling for Rapid Prototyping of Embedded Electronic Systems, RSP, Monterey, USA, 2001.

[17] DOORS is registered trademark of Quality System & Software, Inc.
[18] FPGA: field programmable logic array

5. M. Kühl; C. Reichmann; K.D. Müller-Glaser: Universal Object-Oriented Modeling with ARTiSAN Rts and MATLAB/Simulink, ARTiSAN User Conference 2001, London, UK, 2001.
6. Object Management Group: OMG / Unified Modeling Language (UML) V1.4, 2001.
7. Object Management Group: OMG / Meta Object Facility (MOF) V1.4, 2001.
8. Object Management Group: OMG / XML Metadata Inter-change (XMI) V1.0, 2000.
9. B.P. Douglass: Doing Hard Time – Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns. Addison-Wesley, 1999.
10. M. Fowler: Refactoring - Improving the Design of Existing Code. Addison-Wesley, 1999.
11. E. Gamma et al.: Design Patterns – elements of reusable object-oriented software; Addison-Wesley, 1994.
12. David Harel: Statecharts: a visual formalism for complex systems. Science of Computer Programming 8 (1987,3), 231–274