Proceedings of the 2004 IEEE
Conference on Computer Aided Control Systems Design
Taipei, Taiwan, September 2-4, 2004

Heterogeneous Modeling for Automotive Electronic Control Units using a CASE-Tool Integration Platform

Klaus D. Müller-Glaser, Clemens Reichmann, Philipp Graf, Markus Kühl, Klaus Ritter

***Abstract*—Up to 70 electronic control units (ECU's) serve for safety and comfort functions in a car. Communicating over different bus systems most ECU's perform close loop control functions and reactive functions and have to fulfill hard real time constraints. Some ECU's controlling on board entertainment/office systems are software intensive, incorporating millions of lines of code. The challenge for the design of those distributed and networked control units is to define all requirements and constraints, understand and analyze those manifold interactions between the control units, the car and the environment (driver, road, weather) in normal as well as stress situations (crash). To design within a development process which is concurrent and distributed between the automotive manufacturer and several suppliers requires a well understood life-cycle model, a strictly controlled design methodology and using computer aided engineering tools to its largest extent. We have developed the CASE-tool integration platform "GeneralStore" to support the design of automotive ECU's, however, GeneralStore is also used for the design of industrial automation systems and biomedical systems.**

I. Introduction

MORE than 70 electronic control units (ECU's) serve for safety and comfort functions in a luxury car. Communicating over different bus systems (e.g. CAN class C and B, LIN, MOST, Bluetooth [1]) many ECU's are dealing with close loop control functions as well as reactive functions, they are interfacing to sensors and actuators and have to fulfill safety critical hard real time constraints. The embedded software in such ECU's is relatively small, counting up from a few thousand lines of code to several ten thousands lines of code. The software is running under a real time operating and network management system like OSEK/VDX [2] on a standard hardware platform. Other ECU's controlling the onboard infotainment system (video and audio-entertainment, office in the car with according internet and voice communication, navigation) are really software intensive incorporating millions of lines of code. All ECU's are connected to the different busses which in turn are connected through a central gateway to enable the communication of all ECU's.

As new functions in future cars require communication to traffic guidance systems, road condition information systems as well as car to car communication, the software intensive infotainment ECU's will be directly coupled to power train and body control ECU's, even forming closed loop control. Thus, the design of these future systems need to combine methodologies and computer aided design tools for reactive systems and closed loop control systems as well as software intensive systems.

The challenge for the design of those distributed and networked control units is to find and define all requirements and constraints, to understand and analyze those manifold interactions between the many control units, the car and the environment (road, weather etc.) in normal as well as stress situations (crash), within a development process which is concurrent and distributed between the automotive manufacturer and several suppliers. This requires a well understood life-cycle model (like the V-model [3], fig. 1) and a strictly controlled design methodology and using computer aided engineering and design tools to its largest extent.

For the development of closed loop control functions (e.g. power train control) ECU designers prefer graphical methods using data flow diagrams offered by tools like Mathworks Matlab/Simulink [11] or ETAS Ascet-SD [12]. For reactive functions (e.g. body control) designers prefer statechart desriptions offered by e.g. Matlab/Stateflow or I-Logix Statemate [13]. For the software intensive functions in car-infotainment designers prefer the Unified Modeling Language UML [14]. Therefore, the design of the new complex functions which are distributed over many ECU's will require heterogeneous modeling. To support an according model based design methodology we have developed the CASE (Computer Aided Software Engineering) tool integration platform "GeneralStore", which is described in chapter 2 with its meta-modeling and integration aspects as well as automatic code generation.

Manuscript received February 2nd, 2004.

Klaus. D. Müller-Glaser, Clemens Reichmann, and Philipp Graf are with the Institut für Technik der Informationsverarbeitung at the University of Karlsruhe (TH), Engesserstrasse 5, D-76128 Karlsruhe, Germany; e-mail: {Mueller-Glaser, Reichmann, Graf}@itiv.uni-karlsruhe.de.

Klaus. D. Müller-Glaser, Markus Kühl, and Klaus Ritter are with Forschungszentrum Informatik (FZI), Haid-und-Neustrasse 10-14, D-76128 Karlsruhe, Germany; e-mail: {kuehl, ritter}@fzi.de.

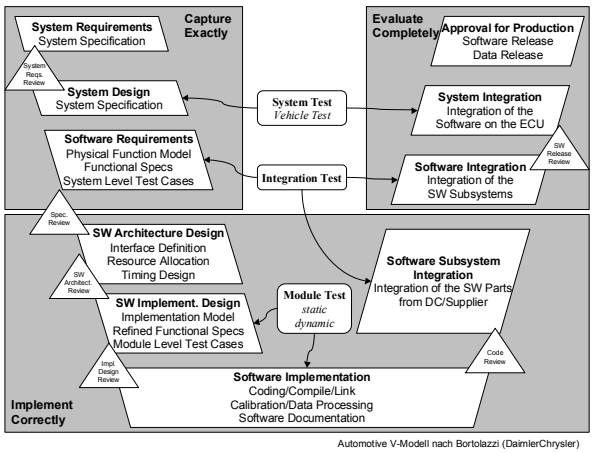

Fig. 1. The automotive V-Model with focus on software [3].

II. GeneralStore - a CASE tool integration platform

The integration platform GeneralStore is a tool that assists a seamless development process starting with a model and ending with executable code. The integration platform features coupling of subsystems from different modeling domains on model level. From the coupled model it generates a running prototype respectively system by code generation. In addition to object-oriented system modeling for software intensive components in embedded systems, it supports time-discrete and time-continuous modeling concepts. Our approach provides structural and behavioral modeling with front-end tools and simulation/emulation utilizing back-end tools. The CASE-tool chain we present in this paper further supports concurrent engineering including versioning and configuration management. Utilizing the UML notation for an overall model based system design, the focus of this paper lies on the coupling of heterogeneous subsystem models and on a new code generation and coupling approach.

A. Meta-modeling

In our approach the whole system is described as an instance of one particular meta-model in one notation. The related meta-model has to cover all three domains: time-discrete, time-continuous, and software. The Unified Modeling Language is an Object Management Group (OMG) standard [6] which we use as a system notation and meta-model. It is a widely applied industry standard to model object-oriented software. Abstract syntax, well-formed rules, the Object Constraint Language (OCL) and informal semantic descriptions specify UML. As we will point out later, we use this notation to store the overall model while ECU-designers still use those domain adequate modeling languages (e.g. signal flow diagram, state charts, UML, etc.), which fits best to her/his design problem.

The UML specification provides the XML Metadata Interchange format (XMI) [7] to enable easy interchange of meta-data between modeling tools and meta-data repositories in distributed heterogeneous design environments. XMI integrates three key industry standards: the Extensible Markup Language (XML) as a standard of the World Wide Web Consortium W3C [16], the UML, and the Meta Object Facility (MOF) [8], an OMG meta-modeling standard which is used to specify meta-models.

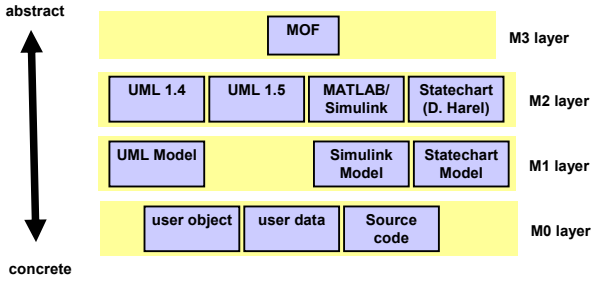

Fig. 2. Four-layer meta-modeling architecture

One key aspect of UML is the four layered meta-modeling architecture (Fig. 2) for general-purpose manipulation of meta-data in distributed object repositories. This makes it suitable for a universal object-oriented modeling approach. Each layer is an abstraction of the underlying layer with the top layer (M3) at the highest abstraction level. The bottom layer (M0) comprises the information that we wish to describe, e.g., the embedded system, the data, or the execution code of a program. For embedded systems the source code is given in different languages, e.g., JAVA or C++, which executes on the target machine. 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 an UML model. The meta-model on the M2 layer consists of descriptions that define the structure and semantics of meta-data (e.g., the UML model). These are the meta-models, e.g., UML 1.4 or UML 1.5, and define the language respectively notation for describing different kinds of data (M1). Finally, at the M3 layer there is the meta-meta-model MOF. It is used to describe meta-models and define their structure, syntax, and semantic. It is an object-oriented language for defining meta-data. MOF is self-describing. In other words, MOF uses its own meta-modeling constructs.

B. Integration Platform

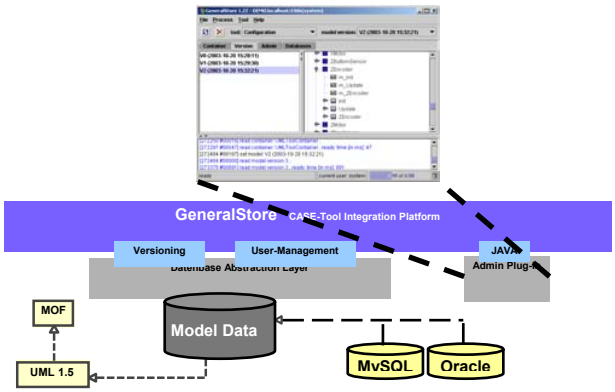

Fig. 3. CASE tool integration platform GeneralStore: model database handling view

Fig. 3 and Fig. 4 show the integration platform GeneralStore. The setup of GeneralStore follows a 3-tier architecture. On the lowest layer (database layer) the commercial object-relational database management system ORACLE respectively MySQL was selected. On the business layer we provide user authentication, transaction management, object versioning and configuration management. GeneralStore uses MOF as its database schema for storing UML artifacts. Inside the database layer an abstraction interface keeps GeneralStore independent from the given database.

While interim objects on the business layer enclose MOF elements, the CASE adaptor stays thin and highly reusable. These interim objects are used to enable object identity to handle object versioning and configuration management. On the business layer of GeneralStore the mediator pattern [9] is used to keep the CASE-tool integration simple and its adaptor uniform. The transformations for specific notations supported by CASE-tools are implemented using plug-ins (see top of Fig. 4).

On the presentation layer GeneralStore provides three principal CASE-tool adapters:

1. MATLAB/Simulink/Stateflow was selected for the control system design domain and the integration is done using the proprietary model file (MDL).

2. Generic and specialized XMI importer/exporter filters of *.xmi files: Here we use XSLT transformations [10] to adopt the tool specific interpretation of the UML and XMI standard. The UML CASE-tools we have chosen are Together (Borland), Poseidon (Gentleware), MagicDraw (No Magic, Inc.), Rhapsody in C++/JAVA (i-Logix), and Rational Rose (IBM). Statemate (i-Logix) was chosen in the time-discrete domain.

3. COM based integration of ARTiSAN Real-Time Studio: This UML CASE-tool was selected because of its focus on embedded real time systems.

All tools, except Statemate, which allows only export of XMI files, are linked to the GeneralStore architecture in a bidirectional way.

The code generation plug-ins (Template Coder, Embedded Coder, and Rhapsody in MicroC) controls the transformation to the source code (Fig.4). Their wrapper generators are automatically building the interface to the UML model.

For model management and CASE-tool control, GeneralStore offers a system hierarchy browser. Since the internal data-model representation of GeneralStore is UML, GeneralStore offers a system browser for all UML artifacts of the current design. A large amount of MOF objects are generated, e.g. an empty model transformed out of Matlab/Simulink already generates 3783 XMI entities because of the many tool internal parameters normally not visible to the user. A simple integrator block needs 405 entities, a basic PID-block can count up to 2786 entities. However, describing many instantiated blocks of the same kind, the XMI entities increase is linear and scales well even to very large designs. The description of an autosampler (robot arm) with closed loop and reactive functions for a chemical analysis system, for example showed a total of 44239 XMI entities. The description of a complete passenger car for hardware-in-the-loop tests in Matlab/Simulink (>8 Megabyte .mdl file) generated more than 4 million entities. However, todays powerful database systems still perform very well with that amount of data items. For speed up in navigating through a complex design GeneralStore 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.

C. Code Generation and Coupling

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 platform allows partitioning of the whole system into subsystems. Thus we enable the usage of different domain specific code generators. Each code generator has benefits in specialized application fields.

We follow the Model Driven Architecture (MDA) [14] approach: transforming a Platform Independent Model (PIM) to the Platform Specific Model (PSM).

For control-systems there are commercial code generators like TargetLink (from dSPACE GmbH [17]), Embedded Coder (from Mathworks, Inc.) or ECCO (from ETAS GmbH). In the time-discrete domain we utilize the code generator of Statemate (Rhapsody in MicroC from I-

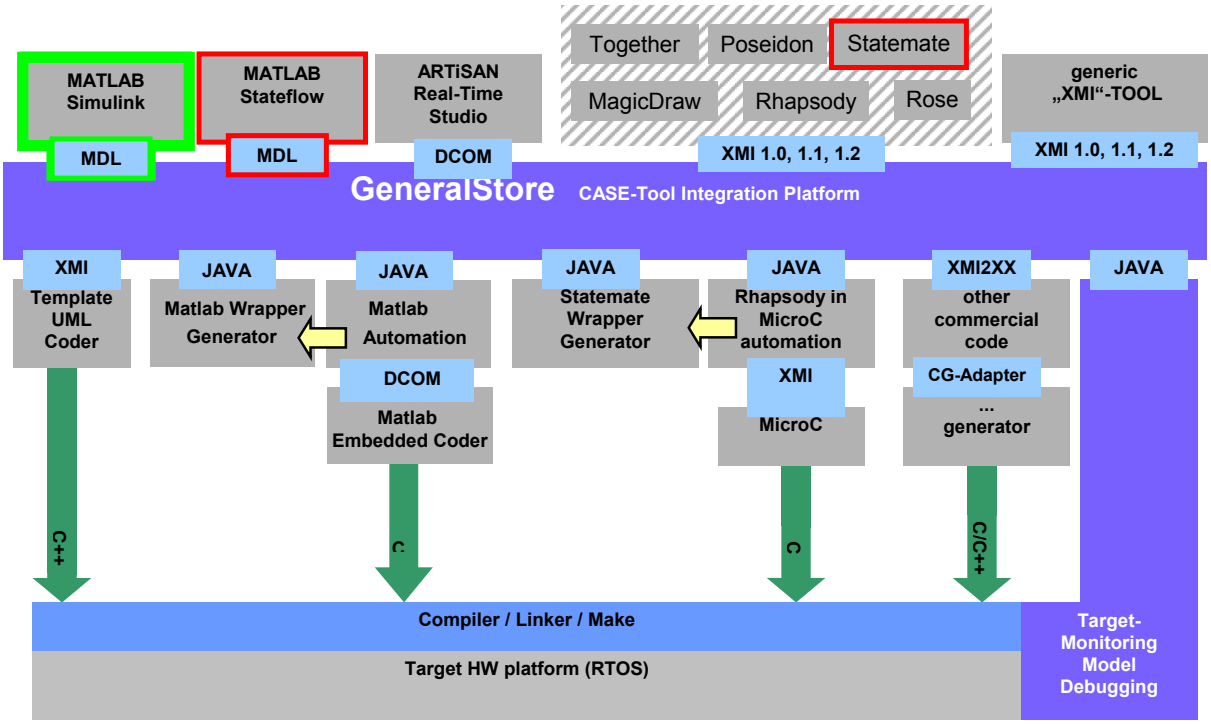

Fig. 4. CASE tool integration platform GeneralStore: code generation view

Logix). In the 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 out of the overall model. Therefore we provide a code generator as a GeneralStore 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 a high level action language based on Java syntax. It suits the action semantics defined by the OMG since UML version 1.4 as the concrete syntax.

Currently GeneralStore supports Embedded Coder for closed-loop models, Rhapsody in MicroC for state charts, and a template coder for the UML domain (see Fig. 4). Our template code generator is using the Velocity engine to generate Java or C++ source code. Velocity is an Apache open source project [18] focused on HTML code generation. It provides macros, loops, conditions, and callbacks to the data model's business layer. One of its strengths is the speed when rendering.

Using the templates, the structure of the UML model is rendered to code. The behavioral specification is done with MeDeLa. It is transformed to the according Abstract Syntax Tree (AST). Then the AST is traversed as the Velocity template renders each statement or expression. It is possible to access the whole UML model from the template. Up to now, we use class diagrams and state diagrams.

Different domains have interactions, e.g., signal inspection, adoption of control system parameters at runtime, or sending messages between time-discrete and software artifacts. Those interfaces are defined in the different models and the coupling definitions are done in the UML model. The developer of such a subsystem is able to explicitly define which events, signals, data, and parameters can be accessed from the outside (the UML model). After the definition in the specific domain (e.g., closed-loop control system design) is finished the notation is automatically transformed to the UML. For the discrete subsystem domain this works analogously.

The wrapper generator collects all the information about the interface in this model and generates artifacts, which represent the interface in UML. This includes the behavioral part of the access, which is done with MeDeLa. The developer uses the UML template mechanism to specify the general access mechanism for a specific type of interface. This is highly customizable. Thus the code generation provides a highly flexible mechanism for modeling domain interfaces.

GeneralStore is currently in beta-site evaluation in the automotive industry. Results so far are very promising,

however, thorough investigations on performance limits and other limits are still under investigation. Also we started investigating other applications domains like industrial automation systems and biomedical engineering systems.

III. IP-Software Module Certification

Automotive ECU-software development is mainly done by ECU suppliers, whereas original equipment manufacturers (OEM's) usually develop requirements and later on do the integration and test. More and more libraries of functions are built up implementing software modules portable to several standard hardware platforms. However, these function libraries must be quality assured. To assist future IP-software module exchange between OEM and tier 1 and tier 2 suppliers, we are currently working on the development of a software-module certification procedure incorporating tools for rule checking (e.g. for programming guidelines as MISRA-C), static and dynamic software tests as well as incorporating model checking and abstract interpretation in a seamless way into the software development flow.

Model based ECU-software development is still not standard. Also, a lot of legacy code exists. As writing static and dynamic software-unit-tests or module-tests may take even more time than implementing the function itself, and as test coverage is critical for safety related functions we are interested in incorporating new test methods. Abstract interpretation is such a method which, as commercial tools now become available, should be used already in early development phases to help discover runtime errors without the need to manually writing test cases. We have integrated the abstract interpretation tool Polyspace C-Verifier [4] within three established automotive tool chains (fig. 5) in such a way that a seamless automatic design and verification process flow is possible. First results on abstract interpretation of new software modules proved the value. Even when testing ECU legacy code a remarkable number of runtime faults has been detected.

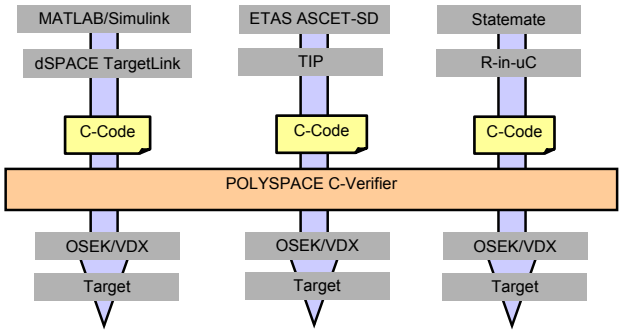

Fig. 5. Abstract interpretation verifier within established automotive tool chains

IV. Conclusion

The CASE-Tool integration platform *GeneralStore* combined with a universal object-oriented modeling approach supports the concurrent development of embedded electronic systems in all design phases. We showed how heterogeneous system descriptions in different notations could work as integrated parts of an object repository based client/server CASE-tool environment. Based on an object diagram representation, time-continuous subsystems and software components can be modelled using one single meta-model. 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 domains. Each domain uses its specific notation. A highly flexible design process was described to integrate those notations, which are supported by different CASE-tools. The glue between the domains is modelled in UML. Finally the overall system is transformed to source code with the assistance of commercial code generators in addition to our own UML template based 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 point in a specific model hierarchy. 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 project hierarchy.

One major drawback of using UML/MOF from XMI as a meta-model 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 generation XMI. On the other hand, using XMI and the UML meta-model for the description of embedded systems enables model exchange with other CASE-tools. Today, at least 10 software CASE-tools are on the market which can handle XMI descriptions to import a model but without graphical description.

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 from Quality System and Software Inc. [19]) and especially integrating seamlessly a design process for safety-relevant ECU's.

REFERENCES

[1] Automotive Busses: http://www.interfacebus.com/Design_Connector_Automotive.html

[2] OSEK/VDX homepage: http://www.osek-vdx.org

[3] V-Model home-page: http://www.v-modell.iabg.de/vm97.htm#ENGL

[4] Polyspace home-page: http://www.polyspace.com

[5] Bortolazzi, J.: Systems Engineering for Automotive Electronics. Lecture Notes, Dep. of EEIT, University of Karlsruhe, Germany, 2003

[6] Object Management Group: OMG / Unified Modeling Language (UML) V1.5, 2003

[7] Object Management Group: OMG / XML Metadata Inter-change (XMI) V1.2, 2002

[8] Object Management Group: OMG / Meta Object Facility (MOF) V1.4, 2001

[9] E. Gamma et al.: Design Patterns - Elements of Reusable Object-Oriented Software; Addison-Wesley, 1994

[10] David Sussman, Michael Kay: XSLT Programmer's Reference, WROX, 2001.

[11] The Mathworks homepage http://mathworks.com

[12] ETAS homepage: http://en.etasgroup.com

[13] I-Logix homepage: http://www.ilogix.com

[14] UML homepage: http://www.omg.org/gettingstarted/what_is_uml.htm

[15] Artisan homepage: http://www.artisansw.com

[16] World Wide Web Consortium (W3C): homepage:http://www.w3.com/Consortium

[17] Dspace Inc. homepage: http://www.dspaceinc.com/ww/en/inc/home.htm

[18] Java based template engine: http://jakarta.apache.org/velocity/

[19] Telelogic Inc. homepage: http://www.telelogic.com/