Supporting system level design of distributed electronic control units for automotive applications

Klaus D. Müller-Glaser[1], Clemens Reichmann[2], Markus Kuehl[2],
[1] Universitaet Karlsruhe, ITIV, Engesserstrasse 5, 76128 Karlsruhe, Germany
kmg@itiv.uni-karlsruhe.de
[2] Aquintos GmbH, Lammstrass 21, 76133 Karlsruhe, Germany
reichmann@aquintos.com, kuehl@aquintos.com

Keywords: automotive control systems, heterogeneous models, CASE tool integration platform GeneralStore, model transformation, EE-Concept-Tool for design space exploration

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 fulfilling hard real time constraints. Some ECU's controlling on board entertainment/office systems are software intensive, incorporating millions of lines of code. The design of these distributed and networked control units is very complex, the development process is a concurrent engineering process and is distributed between the automotive manufacturer and several suppliers, this requires a strictly controlled design methodology and the intensive use of computer aided engineering tools. The CASE-tool integration platform "GeneralStore" and the "E/E-Concept Tool" for design space exploration supports the design of automotive ECU's, however, GeneralStore is also used for the design of industrial automation systems and biomedical systems.

1. INTRODUCTION

More than 60 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 on a standard hardware platform under a real time operating and network management system like OSEK/VDX [2]. 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 already 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]) 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-MD [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 inte-

gration platform “GeneralStore”, which is described in chapter 2 with its meta-modeling and integration aspects as well as simulation and automatic code generation. Chapter 3 discusses model to model transformation and chapter 4 describes the “E/E-Concept Tool” which supports design space exploration for the automotive domain.

2. 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 (see for Figure 1) 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 chapter further supports concurrent engineering including versioning and configuration management. Utilizing the UML notation for an overall model based system design, the focus of this chapter lies on the coupling of heterogeneous subsystem models and on a new code generation and coupling approach.

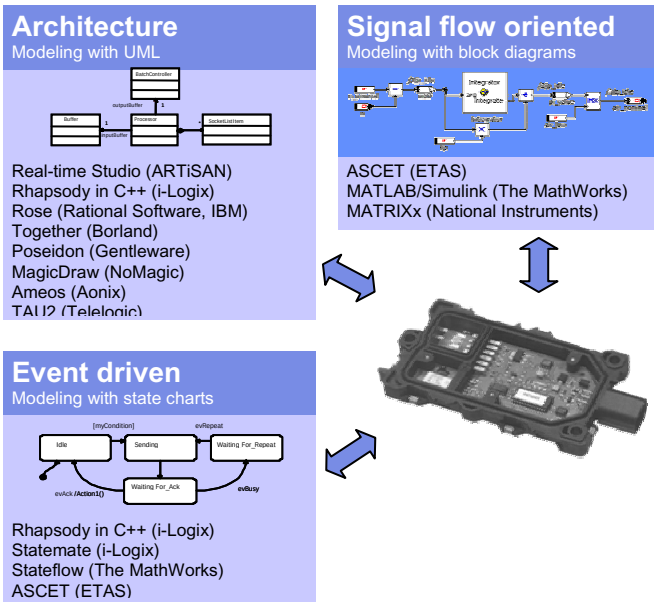

Figure 1. Tools used for heterogeneous modeling in automotive applications

2.1. 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.

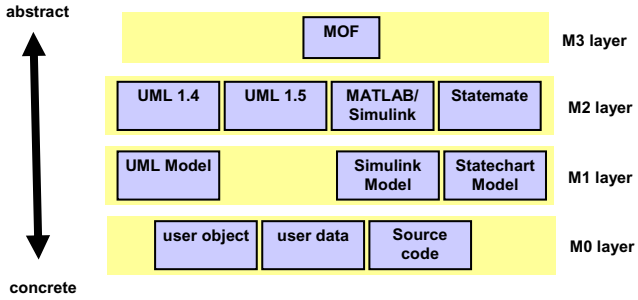

Figure 2. Four-layer meta-modeling architecture

One key aspect of UML is the four layered meta-modeling architecture (Figure 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

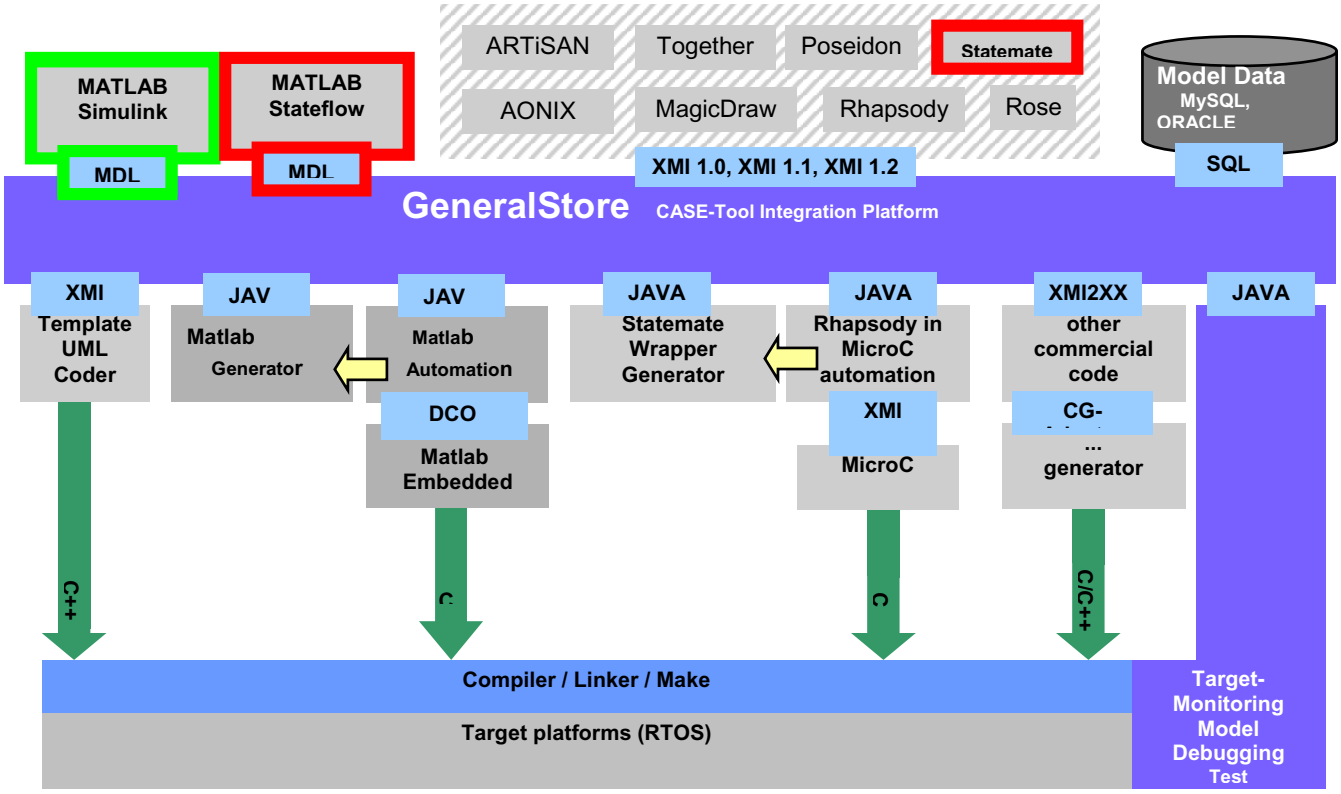

Figure 3. Integration Platform GeneralStore

semantics of meta-data (e.g., the UML model). These are the meta-models, e.g., UML 1.5 or UML 2.0, 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-modelling constructs.

2.2. Integration Platform

The integration platform 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.

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 Figure 3).

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 (Figure 3). 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, today's 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.

2.3. 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 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-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 Figure 3). 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.

2.4. Simulation and Emulation

For early verification on system level and subsystem level simulation as well as emulation is used.

The standard procedure in designing new functions for automotive applications is to build the model with the according tool and simulate that model within that tool (supporting either closed loop control systems, reactive systems or software intensive systems). This works well for the subsystem level where we may have only models of one kind.

However, the model of the complete system very often consists of models of different kinds. Simulation of heterogeneous models for closed loop control together with reactive control is supported within several tools e.g. Matlab/Simulink/Stateflow or ETAS Ascet MD). These tools offer synchronized simulation kernels, one for closed loop control to numerically solve systems of ordinary differential equations (calculating physical quantities continuous in value and time), the other kernel to solve boolean equations (quantities discrete in value and time) for reactive systems. Combining these models with UML models is not supported in a single simulation tool. A possible solution is to use and link different tools to carry out co-simulation [20], to use a simulation-backplane [21, 22], or to use GeneralStore to link the code fragments.

In using automatically generated code running on a rapid prototyping hardware platform, emulation complements simulation for verification purposes. Running generated C-code on a target microprocessor platform or even generating VHDL code and synthesizing a hardware implementation of an algorithm to be mapped into an FPGA speeds up execution time by two orders of magnitude compared to simulation.

A very common problem for simulation of automotive applications is due to the use of model libraries of one domain (e.g. closed loop control) but of different tools; most prominent is the use of libraries in Matlab/Simulink and ETAS Ascet MD. Whereas Matlab/Simulink models are used in early system design phases, ASCET MD models may be used later on for models closer to product implementation. So very often a simulation using models out of both tools is required. Here, a tool integration platform like ETAS INTECRIO gives support [23]. INTECRIO is a rapid prototyping environment for electronic control units supporting the specific needs of electronics and software development in the automotive industry. The possible integration of different Behavior Modeling Tools (BMT) allows working with the tool of choice in each project and in each specialized team. Open interfaces and the available BMT Connectors for MATLAB®/Simulink® and ASCET enable a smooth integration of different software functions generated with different BMTs. The resulting software prototype can be tested on a rapid prototyping hardware.

Another possible solution is to do a model transformation from one tool to the other tool, e.g. Matlab/Simulink into ASCET MD. The very same problem exists when using UML models built with different UML tools. Thus model to model transformation is another important issue.

3. MODEL TO MODEL TRANSFORMATION

The above mentioned reasons ask for model to model transformations, but there are other reasons, too. One of the major problems in the ECU software development process using heterogeneous modelling is an obvious lack in design synchronisation, which means that model data has to be created multiple times to cover the needs of the different design phases (e.g. concept design versus detailed design). Moreover the phase transition between software architecture design (UML notation) and software development (block diagram notation, state-charts) is not continuous as a result of different description languages. This often leads to a design gap between the mentioned design phases. Overall the design gap between different CASE tools or even between different design phases has to be closed to accelerate future developments. A technique to solve this productivity issue is model to model transformation. Established model to model transformations (e.g. model importers to migrate models from different vendor tools) are hard coded today and therefore hardly to maintain as the model often is very complex. We have developed an innovative solution for model-to-model (M2M) transformation to bridge the obvious design gap between different CASE tools.

Our model-to-model transformation uses a model-based design approach for the design and construction of transformation rules. This technique to annotate the transformation rules is called M²TOS. It uses the graphical Unified Modeling Language combined with a special action language as a specification language for transformation rules between a source model and a target model. The advantages in using UML for the specification of transformation rules result in a very good readability and maintainability of the transformation rules. An optimising code-generator is used in the M²TOS design process to produce executable and efficient transformation rules without hand-coding at all.

In Figure 4 one specific transformation rule is shown. It is separated into a sub-package LHS (left hand side) and RHS (right hand side). LHS is implemented as a search pattern and RHS as a creation pattern that simultaneously parameterises the merge process. The structure of a transformation rule is graphically modelled as a UML Object diagram. The search pattern supports the AND, OR and NOT structures. The attribute values of an object are specified formally in text form by means of the Method Definition Language (MeDeLa). Therefore the expressions are declarative in the search pattern and imperative in the creation pattern. On one hand the transformation rules are grouped in transformation concepts and on the other hand, the rules are organised based on the source meta-model. With both these mechanisms, a clear and easily understand-

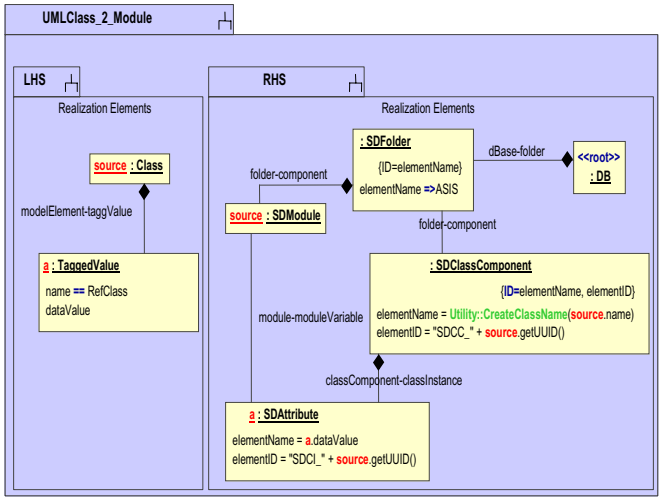

Figure 4. Transformation Rule specified in UML

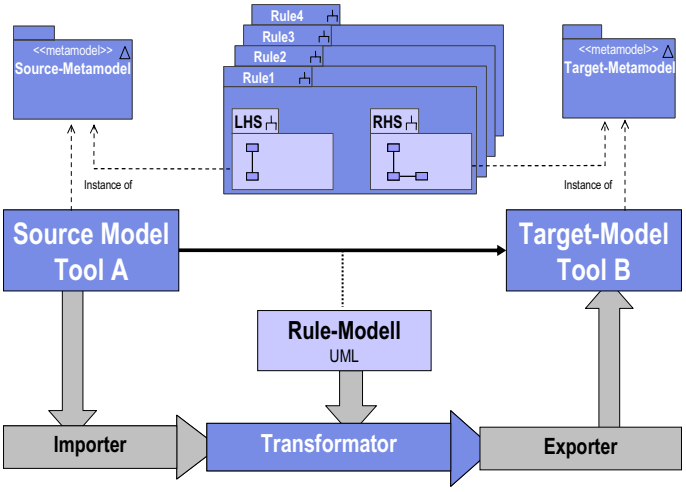

Figure 5. Structure of the M2M transformation

able transformation model can be created. Redundant transformation rules can be avoided by reuse-mechanisms.

In Figure 5 the resulting model-to-model transformation sequence is presented. In the given scenario Mathworks Simulink® is used as a source tool to provide rapid prototyping models to feed an ASCET® function development for detailed ECU software design. The Simulink models are parsed by an importer of the transformation engine. The transformation engine with the help of the M2M rule-set scans the source model and generates target model artefacts to export to ASCET. The speed of the transformation is of very high performance. Depending on the size of the model and the given CASE tool interface even big models can be transformed under one minute on a standard PC. In a comparable sequence, a different rule-set implements a M2M transformation between ARTiSAN Studio UML and ASCET.

The available transformation rules for ASCET and Simulink are implemented bi-directional to synchronize models between both CASE tools. This synchronisation technology is very important to ensure that further modifications of transformed models have to be bridged back into a master model or to synchronize development teams using different CASE tools without the need for time-consuming hand-based transformations. This model synchronisation (which also covers the "merge" use case) is implemented with a model versioning technique based on a relational database, to compare an updated model with a previous version of the model. This synchronisation technique ensures that changes in the model will not be lost after repetitive transformation. In any case, a transformation report will be generated.

So far we looked at modeling functionality using heterogeneous models and using different behavior modeling tools. However, this is not enough to evaluate different architectures of distributed ECU's in a car. Design space exploration in early system design phases needs domain specific descriptions of different views onto the system, taking into account not only ECU's but also the mounting spaces and the wiring harness which interconnects them. These aspects build up the electrical/electronic architecture (e/e-architecture) of a car, a large set of metrics is used to analyze alternative architectures.

4. DESIGN SPACE EXPLORATION

The early concept phases in the development process do have a deep impact on the final e/e-architecture of the vehicle. This impact can be measured in resulting costs for the mass production of e/e components. The goal of the concept phase therefore is to make the right decision for a specific e/e-architecture under different weighted metrics (costs, quality, reliability etc.) or constraints.

The introduction of methods to improve and formalize early concept phases is a must. The requirements for methods in early concept phases are: a complete formalised description of the e/e architecture; automated rapid evaluation of the various possible e/e-architectures against defined cost and/or quality metrics. Both requirements can be captured by the introduction of a complete data-model for concept evaluation of e/e-architectures including design domains like Function/Feature Networks (top level description), Function Networks, components networks and finally topology (e.g. wiring harness, installation place for ECU

components). Combined with a set of properties for each data-model artefact, different e/e-architectures can then be evaluated and compared.

When looking into concepts, the evaluation of different e/e-architectures is nearly as important as their formal description. The assessments to find the "right" architecture are based on numerous metrics that give key figures and qualitative facts for measuring purposes and comparisons between various conceptual alternatives.

Currently, most of these metrics are collected and set together by hand. Often, some data is held in a spread sheet tool like Excel which then allows for calculating some metrics, but still necessary data is to be collected and prepared. This means a time consuming process, which limits today's evaluation to only looking into a few conceptual alternatives focusing on one or two possible vehicle variants. Other metrics can only be estimated by experts, a rather non-objective and non-repeatable basis for drawing decisions. Regarding the actual situation with constraints on time and costs while running into an ever increasing complexity, such manual evaluation is not sufficient anymore.

A prototype for a tool has been developed by Daimler-Chrysler (Mercedes-Benz Passenger Car Development and DaimlerChrysler Research & Technology) and by FZI Forschungszentrum Informatik (Research Center for Information Technology, Department Electronic Systems and Microsystems) in the last two years. After a successful evaluation of the tool prototype at DaimlerChrysler, the tool is now being transferred into a commercial product by Aquintos GmbH [24].

The importance of a formalised meta model for the concept evaluation phase of e/e-architectures is resulting from a need to structure model data (so called concepts), graphics (graphical notation), metrics (cost and quality estimation etc.) and also evaluation results. Benchmarking (e.g. comparison of different e/e concepts) can only be implemented with an e/e model. The meta model is an automotive domain specific data model to handle the overall concept model of a complex e/e-architecture in the concept phase. The focus of the meta model therefore lies on interface and structure data information (e.g. functional architecture). Algorithmic parts are intentionally excluded from the meta model since the algorithmic or state event behaviour of the ECU functions are part of the following detailed (fine) development phase. In Figure 6 the different e/e-architecture layers of the meta model are named, Figure 7 shows the abstraction layers, Figure 8 shows a schematic of ECU mounting locations and the wiring harness of a car.

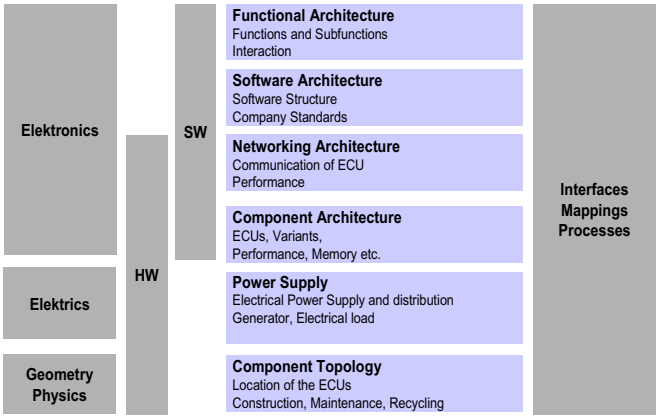

Figure 6. Architecture Layers available within the E/E meta model

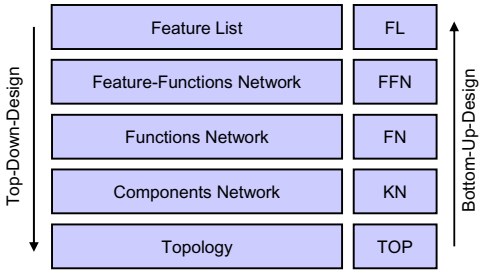

Figure 7. Abstraction Layers of the Concept Tool Notation

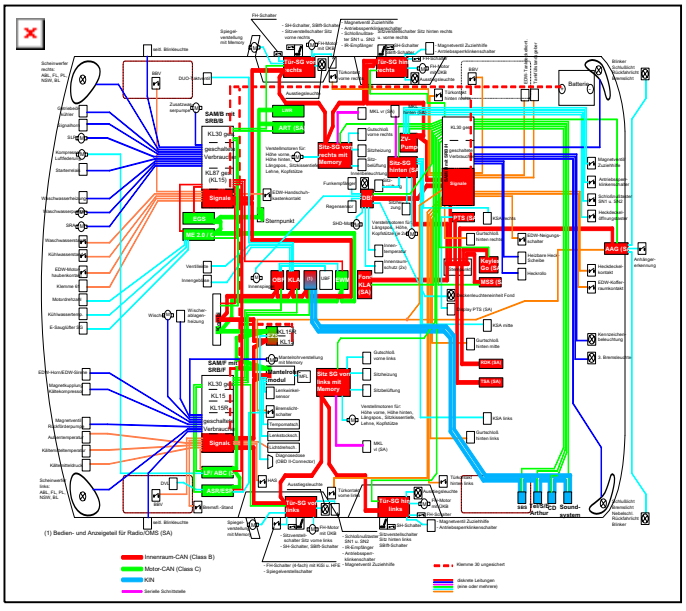

Figure 8. Electrics/electronics in a car – schematic drawing

Figure 8 visualizes the need to also model the electrical part of a distributed ECU network. So, the formal description of e/e-architectures in a database now allows for a widely automated evaluation. Once set up, the tool is capable of going repeatedly through various different architecture concepts and producing objective metrics for their comparison. This doesn't only save time, but allows for further investigation of factors that couldn't be taken into count so far.

In an atomised evaluation process, which is supported by a formal database description, it is important to distinguish the two main categories of such metrics: quantitative (e.g. cost, weight, length, number of wires) and qualitative (e.g. reusability, reliability, level of complexity, safety integrity level). Obviously, the quantitative side consists basically of database queries and counting tasks. This remains sufficient as long as all relevant information is already at hand and therefore is included in the databases. However, in the early conceptual stages of a project, several values such as cost and weight or even whole parts have to be estimated. Such estimations can be done by users or automatically derived by more complex algorithms. In addition, the qualitative side needs more sophisticated algorithms. And, as soon as one requires automated evaluations and comparisons, that summarize the results and use thresholds for flagging, a simple querying is not sufficient any more. With Jython, a standardized scripting language is available to the user for independent implementation of any evaluation algorithm. This makes it possible to both program database queries and investigate complex correlations in a similarly straightforward fashion to Excel macros. Furthermore, the results can be attached to the architecture description in the database and visualized in the editors as either information or colour changes. With seamlessly providing views and editors, that are necessary to support the implementation of algorithms and the back-annotation of the results, the evaluation component is not just a scripting interface but an integral part of the whole tool framework.

5. CONCLUSION

To cope with increasing complexity of future e/e-architectures in a car according and different best of point tools are used for modelling and simulation of closed loop control, reactive control and software intensive functions. Tool integration platforms and new tools for early design space explorations are required and are currently becoming available commercially. First applications of these tools for the design of next generation cars show promising results but also the need for further research and development of tools especially supporting early system design phases and supporting a seamless design flow from requirements specification to system level modelling, system level simulation, emulation, rapid prototyping, design, analysis, integration, test, application and maintenance. Also, a tight connection to life-cycle product data management systems (PDM) is required.

Domain specific tools will evolve quickly, as evaluating architectures requires taking into account domain specific metrics of high complexity.

List of 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/

[20] Krisp,H., Bruns, J., Eolers,S.: Multi-Domain Simulation for the incremental design of heterogeneous systems. European Simulation Multiconference ESM'2001, S. 381-386

[21] Tanurhan,Y:;schmerler,S.,Müller-Glaser,K.: A Backplane Aprroach for Co-Simulation in High Level System Specification Environments, Proceedings of EURODAC'95

[22] Schulz, H.M.: Description of EXITE and distributed simulation toolbox. Extessy AG, 2002

[23] ETAS INTECRIO Homepage http://en.etasgroup.com/products/intecrio/in_detail.shtml

[24] Auintos GmbH Homepage: http://www.aquintos.com/de/index.php