Smart Assistance in ECU Design and Integration

Störmer, Christoph, Verhoef, Chris
ETAS GmbH, Germany, Vrije Universiteit Amsterdam, The Netherlands

KEYWORDS – design assistance, integration, semi-automation, timing, tools

ABSTRACT - Over the last years, significant effort has been spend to introduce model based software development methods and supporting tools in the development of electronic control units (ECU). One important future trend will be to complement model based development with smart assistance technologies. In the future, tools will support developers by offering assistance how to improve designs and calculate the impact of their design decisions.

For example, mapping functional building blocks on ECUs requires a sound understanding of multiple criteria, such as resource, performance, dependencies, cost, safety, and electrical power consumption. Providing assistance by presenting developers alternative choices and their respective impact on system performance, power consumption and cost will significantly support the decision making process. Mapping functionality to ECUs without having smart assistance in analyzing the impact of the mapping only addresses constructional aspects but lacks an analysis about the correctness of the execution behavior. Another example is the translation of requirements into design models. Assisting the developer in the creation of the design model and, at the same time, providing early feedback on the effect of design decisions on the implementation of initial requirements will significantly improve the quality of the design. In addition, through the early detection of design mistakes overall development time will be reduced.

How can such assistance be achieved? To illustrate the answer we sketch the assistance for optimizing the runtime performance of a system. Today, commercial tools exist to determine worst-case execution times through precise calculations for ECU software. Based on this data a system-level timing analysis, e.g., through holistic schedulability analysis, can be carried out for the SW on a single ECU or in a network of ECUs. The tools use analytical models to calculate the timing properties and the timing behavior. Tomorrow, this knowledge will be combined with suggestions on how to improve the timing behavior of ECU software. These will be design suggestions for architecture models as well as implementation suggestions for rather detailed models. Furthermore, the effects of alternative suggestions, such as using an event triggered approach instead of a polling approach, can be analyzed precisely and presented to the developer. For optimizing runtime performance, assistance will be achieved by providing an analytical model and effective patterns to improve timing behavior. Both parts constitute the timing assistance.

Future assistance technologies are “smart”, because the assistance will be carried out semi-automated by offering developers choices within their concrete development task. There are many more fields of application for assistance, which include on/off-board partitioning, diagnostics, refactoring, variability, and safety. The major driver to complement tools with smart assistance is to erase design mistakes early, and to guide the developer to successfully master difficult tasks in system design and integration.
Imagine your software development methods, tools, and processes will hardly master the complexity required by automotive systems in the near future. There are too many variation points, the behavior in your network of ECU is not predictable, the overall test coverage hardly exceeds 20%, the tools you use are getting increasingly complex, your best functional designer left the organization for other career opportunities, and the productivity of your team is still not there where you would like to see it.

If parts of this scenario sound familiar, then this paper can help you to identify some strategies and ideas that will help you to attack your future challenges. The core statement is that future improvements in software quality and productivity will be significantly determined by assisting developers with semi-automated techniques across the development process. The goal is not to replace developers with automation but rather support them in making design decisions that are predictable and repeatable. The assistance will reveal blind spots and direct developers to improved solutions. They are in the loop rather than an unnecessary obstacle. The developer has in-context assistance to validate the consequences of design decisions at each step of the development process. The availability of those techniques will significantly determine the complexity and business demands of next generation development approaches.

The other core statement in this paper is that those techniques will not lead to increasingly complex tools but rather simplified tools, reduced in their scope to the particular task that they have to fulfill. It seems that the future does not belong to monolithic tools that are hard to comprehend by their users and difficult to maintain by their manufacturers. There could be rather many tools or tool components, each specialized for a particular task, but interoperating as a whole, and assembled to the particular needs of the development process.

The underlying idea for both statements is a clear distinction between the construction of a design and the design decisions that are embodied in the construction. It is a distinction between a blueprint and the reasoning that leads to a blueprint. This distinction is outlined in Section 1 “Concept”.

Making design decisions and their consequences explicit rather than leaving them implicit in the minds of a developer will help to improve designs and the productivity in many ways. But the necessary requirement in order to do so is the availability of models that capture the knowledge that design decisions are built on. For example, if timing is a concern that a design decision has to be built on then the assistance should know analytical timing models and strategies how to improve the timing behavior of a design. Providing assistance by presenting developers with alternative choices and their respective impact on system performance, power consumption, or cost will significantly support the decision making process. But assistance does not only support the decomposition of a system into parts, it will also support the composition of parts to a system. For example, mapping functionality to ECUs without analyzing the impact of the mapping only addresses constructional aspects but lacks an analysis about the correctness of the execution behavior. Assistance will also help to validate and optimize assemblies, which will be further outlined in Section 2 “Assistance”. In Section 3 “Example” we illustrate the approach using a timing example. Finally, we conclude the paper with a “Summary” in Section 4.
The underlying idea that this paper is based on is a clear distinction between the construction and the reasoning that leads to the construction, or in other words, that is embodied in the construction. The distinction is illustrated in Figure 1, which was adapted from some earlier work of G. Övergaard (1) and F. Bachmann et al. (2). Three types of participants are illustrated in Figure 1: Developers, Experts, and Modeling Tools. Developers construct, for example, a functional design with Modeling Tools. The process is monitored by Experts who are able to evaluate the design and inform the Developers about the results of the evaluation. Additionally, Experts could suggest constructional changes that could lead to better evaluation results.

Developers are able to construct a model consisting of elements and their relations in the Modeling Tool. They will further name those elements and relations in order to provide a meaning, understandable for them and to others. The Modeling Tool offers element and relation types to the Developers and is able to ensure a syntactically correct result before results would be further processed in other development steps. The Experts have a particular knowledge with respect to concerns that Developers have. Examples of typical concerns relate to safety, timing, variability, on/off-board partitioning, memory footprint, diagnostics, and reliability aspects.

There is not a single Expert but many experts. Each Expert has specialized knowledge about an area of expertise. An Expert is able to analyze a constructed model with respect to the particular expertise. For example, a timing expert could calculate the worst case execution times for elements in the constructed model, and inform the Developers based on a selected scheduling approach whether the model has the correct timing behavior. Experts understand the element and relation types in a model from their particular perspective. A timing expert will be interested in the type task but will probably not care about a type device. Experts will assist the Developers in the development process from their particular perspective.

The roles are clearly distinguished in the illustration of Figure 1. Dissolving the distinction and combining the Modeling Tool with many Experts in one tool will lead to monolithic tools that ...
- …are hard to comprehend for Developers
- …organizations do not want to pay for because they have more features than required
- …will produce confusion on the Developer side about automatic improvements carried out by the Modeling Tool with a high potential to keep developers not in the loop or that are even hidden from them

A clean distinction between the constructional aspects in a Modeling Tool and the required Experts is decisive for future improvement potentials in design quality. But combining the Experts with the Developers provides some traps as well:

- The complexity of models will further increase
- It will be difficult to find developers that capture all expertise
- Expert knowledge will increasingly be captured and automated
- Experts will capture knowledge beyond the industry domain that the Developers are in
- Changes in the model require a high discipline by the Developers to apply the expertise with the same scrutiny. Automated Experts have potential for repeatable and objective results

Despite the advantages of distinguishing the Expert role from the Developer role, a lot of expertise will remain with the developer, especially expertise that is difficult to capture in form of quantitative and qualitative models that Experts are built on. Experts will also not take over the role of Developers. The Developers will stay in charge with the design choices. One of the reasons is the mitigation (trade-off) between solution alternatives offered by several Experts.

It is not difficult to imagine that the Developer role is a placeholder for other roles as well, such as integrator, calibrator, and even service technician, each one with a different perspective on the subject of interest, e.g. models, components, and parts.

2 ASSISTANCE

The previous section distinguished between different roles and how they relate to each other. A good understanding of the Expert is important to develop tool concepts that are scalable, interoperable, and evolvable. Accepting the role of Experts will help to prevent monolithic tool approaches and will allow development organizations access to knowledge in particular areas of expertise. This knowledge is either obtained in the organization itself or obtained from captured knowledge across industries and academia. The Experts assist the Developers in achieving quality designs.

There are different objectives that Developers would like to achieve from Expert assistance. A few examples are listed in the following:

- Calculations. A quantitative analysis of a model is carried out. Examples are the calculation of memory footprint, worst-case execution times, or failure rates.
- Optimization. An analysis about substitutions of model parts with more effective constructs. For example, substitution of static parts in a model with fewer and less expensive model elements.
- Parallelization. Dependencies in functional models are analyzed. Independent model parts are presented to the Developer.
- Testing. A software should be tested, for example with respect to a safety
- Guidance. A service technician is guided with operational sequences to find the root cause of a symptom. A calibrator obtains feedback via a heads-up display about operational sequences to optimize emissions.

The Guidance example shows that expertise could be authored by others in earlier development steps. For example, a diagnostic tree-builder as illustrated in Figure 2 could be authored during the development of a functional design, enriched with further knowledge in later development steps until a service technician uses this expert knowledge. Consequently, one way to collect expertise is to add step-by-step knowledge across the lifecycle and offer the result in a language understandable and executable by others.

Figure 2: Sequence Editor by ETAS

Other expertise is built on algorithmic know-how, such as model-checking. Those algorithms will not be changed or adapted during the development process. Other objectives comprise the collection of re-occurring patterns that are recognized to successfully solve known problems. For example, the substitution of complex interpolation routines with maps to lower the computational footprint.

Expertise can also integrate the knowledge how to test ECU software in a HiL System as illustrated in Figure 3. For example, the Expert includes the knowledge, today in form of scripts, how to validate chassis systems with respect to safety.

The examples show that Experts have a few attributes that are listed in the following

1. They capture an area of expertise, such as safety, timing, diagnostics, etc.
2. They have quantitative and/or qualitative knowledge to verify and test models that are typically constructed in Modeling Tools (requirement models, functional models, code, etc.)
3. They know strategies, tactics (3), and patterns how to improve or change models with respect to their expertise
Assistance by those *Experts* will be *smart*, because the assistance will be carried out semi-automated by offering *Developers* choices within their concrete development task. There are many more fields of application for assistance, which include on/off-board partitioning, diagnostics, refactoring, and variability. The major driver to complement tools with smart assistance is to erase design mistakes early, and to guide the developer to successfully master difficult tasks in system design and integration.

3 EXAMPLE

Is such assistance a vision? No, it is not as we will explain in this section with respect to timing assistance.

Today, commercial tools exist to determine worst-case execution times through precise calculations for ECU software. Based on this data a system-level timing analysis, e.g., through holistic schedulability analysis, can be carried out for the SW on a single ECU or in a network of ECUs. The tools use analytical models to calculate the timing properties and the timing behavior. We integrated in the INTEREST project (4) a timing analysis, which we will briefly introduce in this section.

One of the objectives of the INTEREST project is “to integrate tools with respect to timing along the “V” development cycle, spanning all the analysis, system design, module design, implementation (including certified code generation), functional testing, module testing, system testing and requirements testing”. Note that the intention is not to invent a new timing analysis in each development step and associated tool, but to capture the timing analysis in an expert environment and let the different tools interact with the timing analysis expert. For the tools it is important to provide their model elements and relations in a way that the timing expert is able to carry out the analysis and provide the feedback.
Figure 4 provides an overview of the mentioned tool couplings. In a first step, the functional developer constructs a model according to a set of requirements. The model is exported into the partner tool *ait* from AbsInt (Step 2) that calculates the worst-case execution time for each functional block. The results are transferred in Step 3a and 3b to the schedulability analysis tool *symta/S* by SymtaVision. The result provides feedback if the behavioral correctness with respect to timing is guaranteed. All steps in the analysis are carried out without any hardware involved.

There are some additional remarks to this approach compared to the desired approach that was outlined earlier in Section 1 “Concept”.

- The timing expertise is captured by two tools, *ait* for the worst-case execution time analysis, and *symta/S* for the schedulability analysis. Both tools come from different vendors.
- Each tool provides its own user interface with its own philosophy, views, and interface standards. The user does not have a single interface with the construction tool but has to operate and understand several tools.
- Several steps are involved before the Developer obtains the analysis results (see (2) and (4) for further information).
- The feedback from the expert tools is not integrated in the construction tool but rather comes with a textual presentation. The Developers would prefer an in-context feedback in the views of the construction tool.
- The timing experts do not provide improvement suggestions in terms of elements and relations that are used in the construction tool. A feedback in terms of design alternatives would significantly improve to convert analysis results into a synthesis suggestion.
Despite these challenges we expect many improvements in the future that will have the developer assistance in the center. By assisting the developer in the creation of the design model and, at the same time, providing early feedback on the effect of design decisions on the timing behavior will significantly improve the quality of the design. In addition, through the early detection of design mistakes with respect to timing overall development time will be reduced.

The presented approach is usable during functional decomposition as well as during the composition of components into a system. The advantage is that the timing expert does not care in which development step the system is at the moment.

4 SUMMARY

Smart assistance is about the conscious introduction of automated Experts that will assist Developers to master complexity. It will be one decisive key to ensure higher quality standards and increase productivity. At the heart of the assistance are Experts that capture areas of expertise, such as timing, diagnostics, safety, etc. Those Experts can be applied across the lifecycle. Consequently, we expect that …

- …tools will not endlessly grow in features but will concentrate on their particular tasks and at the same time are enriched with interfaces to expert components that will fulfill specialized tasks
- …developers will get alternative choices with validation support in-context
- …designs will have a much better predictability in quality compared to today

The semi-automated approach to Experts will keep the Developers in the loop and not replace them. In other words, Experts will help to reduce unnecessary complexity in development and maintenance steps across the lifecycle.

5 ACKNOWLEDGEMENTS

This work has been partially supported by the INTEREST (INTEgrating euRopean Embedded Systems Tools) project. INTEREST is a Specific Targeted Research Project (STReP) of the 6th Framework Programme (6FP) of the European Union within the Network of Excellence on Embedded Systems Design ARTIST2.
REFERENCES


