Meta-Modeling and Meta-CASE Tools – A Silver Bullet for Model-Driven HMI Development?

Carsten Bock
Dr. Ing. h.c. F. Porsche AG

 Detlef Zuehlke
German Research Center for Artificial Intelligence DFKI GmbH
Center for Human Machine Interaction
PO Box 3049, Kaiserslautern 67653
Email: zuehlke@dfki.de

Abstract: Due to the increasing complexity of automotive human-machine interfaces (HMI) the development of appropriate user interfaces requires powerful development processes as well as easy-to-use software tools. However, in comparison to domains like embedded system development suitable software tool kits are missing in the field of HMI development. Actually meta-modeling and domain-specific languages represent many promising beginnings to create non-generic tool support for individual modeling tasks. Therefore, this paper presents a model-driven HMI development process and describes the utilization of visual domain-specific languages in this process. Thereby experiences with using current meta-CASE tools as well as standard office applications for creating electronic specifications are presented. Based on these experiences requirements for future meta-CASE tools are derived. The suggested enhancements could pave the way to increased acceptance of model-driven approaches among HMI developers and could consequently allow for overcoming today’s urgent challenges in complex networked development processes.

1 Introduction

In the past decade the field of control systems rapidly changed with the continuous advancement of microprocessors and software technology. The increasing functionality of human-machine interfaces (HMI) coming along with the rapid development of computational power can be perfectly retraced by the example of the automotive industry. While average passenger cars in the 1960s were equipped with an odometer and upmarket cars provided a radio, soon tachometers, displays for cooling water temperature as well as warning messages and further information were established in instrument clusters and dashboards. Nowadays premium cars feature powerful driver information and assistance systems including e.g. AM/FM tuners, compact disks, iPods, mobile phones, SMS message and e-mail support as well as navigation systems. Beyond offering entertainment and information these systems can increase passenger safety with the aid of lane departure warning, blind spot detection and adaptive cruise control. The list of available features could be easily continued. However, the central challenge becomes apparent by the ex-
amples mentioned above: all functionality must be easily accessible for users by means of intuitive HMI. Especially in the automotive industry the development of appropriate HMI deserves special attention since road safety must never be endangered by using HMI [RB05]. Therefore, within the scope of interaction concept development software engineering plays a key role for user acceptance and market success of interactive dialog systems in addition to ergonomic design of hardware like rotary knobs, pushbuttons, levers and switches.

In the following section 2 the characteristics of interdisciplinary development processes are outlined. Moreover, the benefits of model-driven engineering for complex and networked development processes are revealed. In section 3 the demand for adequate computer-based tool support is derived. Furthermore, the essential requirements for tool support in the field of HMI development are described. Consequently, in section 4 the development of tailored tool support is illustrated. Experiences and lessons learned from a pilot project are unveiled in chapter 5. Additionally, the maturity of selected current meta-CASE tools is discussed. Finally, in section 6 conclusions are drawn from the outlined experiences.

2 Efficient development processes for powerful human-machine-interfaces

When successfully developing useware¹, manufacturers usually have to face the challenges of ambitious development tasks. So as to meet these challenges the expertise of on-site developers as well as the competence of experienced suppliers is essential for the success of the product under development. In such heterogeneous process environments all members of interdisciplinary development teams at manufacturers and suppliers must be able to communicate in an effective as much as efficient way. Therefore, the precise documentation of results – acquired by developers in different fields of activity with different knowledge and experience levels and moreover with different working methods and tools – is crucial for successful co-operation. Consequently, intuitive notations for all kinds of information must be established in order to achieve a common understanding among developers. Furthermore, to meet the requirements of interdisciplinary and networked development processes information must be easily accessible to all developers involved. As a result today’s development processes are predominantly paper-based. Moreover, the diversity of tools for creating requirement specifications corresponds to the number of methods and interfaces in development processes. Hence, besides requirement specifications regularly further documents need to be exchanged which are created with standard office applications or imaging software.

Even though these tools can help to increase the productivity in many fields of application their usage is associated with significant problems. In particular such heterogeneous tool

¹Useware includes all hard- and software components of a technical system, which are required for its usage. The expression useware has been created to demonstrate the equal importance of human-machine-systems compared to hard- and software (Ober02, Zuo02).
landscapes inevitably result in myriad media disruptions which can only be overcome by transforming formats with the help of conversion tools – provided that a conversion is at all possible for oftentimes proprietary file formats. Besides transformation related difficulties the most important consequence is room for interpretations. Frequently, these interpretations lead to misunderstandings and cause unrecognized need for action in late project phases. Moreover, significant effort is required for transforming paper-based and thus non-machine readable specifications into virtual prototypes as well as the final product. For instance, the specification of a high-end driver information system easily comprises more than 1000 pages and has to be implemented manually. As a result of the required expenditure of time ideas and concepts can only be evaluated with simulations or prototypes in relatively late project phases of development processes. Experience shows that this is typically insufficient for an early comparison with customers’ and decision makers’ expectations [FCK+05]. Moreover, updating cycles between consecutive versions of an HMI specification take so much time that adjustments to occasionally rapidly changing requirements can only be accomplished with great efforts. Thus paper-based specifications become less important in the final phases of development projects since changes can only be incorporated into the product in a rush without updating the specification. Finally, this leads to product features whose origin can hardly be retraced after product launch.

One approach to overcome the above-mentioned obstacles is a consistently tool- and model-based development process. In the field of HMI software development such a process can bridge the gap between requirement specifications and virtual prototypes and the final product as well. Simultaneously, model-based concepts allow for increased flexibility, reduced reaction times and significantly shorter development cycles. Figure 1 illustrates the flow of information and work results between manufacturers and suppliers in a model-driven HMI development process. In this process the software for a complex HMI is developed by a close co-operation of manufacturers and suppliers.

First of all the manufacturer specifies the basic requirements for the interaction concept. Once all required information for an HMI simulation is available a simulation model is created. This formal model constitutes the central database for the whole specification process and enables developers to deduce all further artifacts on the basis of the electronic specification. Due to this machine-readable requirements document the specification can be maintained centrally and the difficulties of conventional paper-based specifications [Boc06] can be overcome. Already in very early stages of the development process the simulation model can be used for verifying the specified HMI. For that purpose a code generator creates an executable simulation from the simulation model at the push of a button. By means of this virtual prototype developers can verify the HMI and check whether the system meets the specified requirements. Among other things colors, layouts, menu items and dynamic behavior can be assessed very early without the need for hardware prototypes. Additionally, also the interaction concept can be evaluated by iterative in-process evaluations with representative users. Thereby developers can assess the system’s appropriateness for assisting users with specific tasks. Since necessary adjustments can be conducted quickly by changing the simulation model and generating a new HMI simulation, manufacturers are enabled to try out new ideas that would have not been considered in a paper-based development process due to the effort and costliness of change requests.
If incorrect specifications or inconsistencies will be revealed during such early evaluations the interaction concept can be iteratively improved without implementing any hardware prototype.

Simultaneously, the simulation model can be made available to the supplier by using a well-defined exchange format. For the definition of such an exchange format Extensible Markup Language (XML) dialects are recommended since such markup languages can easily be accessed by all parties. The supplier enhances the hitherto platform independent simulation model by appending additional information necessary for implementing the control system and the corresponding interaction concept in hardware on the supplier’s target platform. Since the platform independent simulation does not contain information about programming languages, operating systems and hard- or software architectures on
the target platform the supplier has to provide an implementation model holding the required information. This model will be changed (probably adapted and/or extended) if additional system features have to be implemented in hardware that cannot be realized on the basis of the current implementation model.

Provided that the manufacturer approved a specific simulation model this is handed over to the supplier for prototyping. The supplier merges the specification with the implementation model and creates a platform dependent model. Afterwards the supplier produces executable code by means of code generators and frameworks specific to the target platform. With this prototype the manufacturer can assess the control system which before was only partly evaluated by means of simulations. Although many questions can be answered with the help of simulations the use of (hardware) prototypes is still indispensable. For instance color themes for day and night design or a system’s response time can only be assessed in a reasonable way when using the final target platform. After successful verification and validation the control system is finally approved by the manufacturer. For start of production the supplier only has to incorporate the change requests from the manufacturer’s final evaluation into the code for the final prototype.

With this model-based development process control systems can be developed in an evolutionary and iterative way. In contrast to conventional development processes the model-driven development allows for separating specification from implementation. Moreover, the increased formalization of electronic specifications can help overcoming the barriers in paper-based processes and can thereby improve the communication between all participants in interdisciplinary development teams.

3 Tool Support for Complex Networked Development Processes

Although powerful Computer Aided Software Engineering (CASE-) tools were developed for the field of software development their use is still limited to few application areas. This is mainly a consequence of the UML’s ’one-size-fits-all’ approach [Sch06] as well as the generic and (in current high-end tools hardly) customizable graphical representation of its modeling constructs. Especially in the field of HMI development CASE-tools could oftentimes not become accepted since the UML particularly lacks adequate constructs for Graphical User Interface (GUI) specification [BJ04]. Whereas sophisticated tools like SIMULINK or LABVIEW are available for limited fields of application such as the development of electronic control units (ECUs), however appropriate tool support for HMI developers is still missing. This is due to the relatively small market volume of this business segment [LBM+01]. Hence, manufacturers are forced to develop non-generic computer-aided tools and tool-chains respectively on their own. These software tools shall provide support for specific domains such as HMI development. By means of non-generic CASE-tools developers shall preferably be assisted in all phases of a model-driven and user-centered development process with using specific methods and tools such as an advancement of Harel’s state charts [Har87] called illustrated state charts [HDB03] (Fig. 2).
Figure 2: Illustrated state chart for HMI specification in PowerPoint

Even though demand for domain-specific tool support and its benefits are apparent, the high cost of proprietary software development and a considerable development risk hinder the broad acceptance and employment of domain-specific CASE-tools [Spi01]. Therefore, in the following an approach for implementing a CASE-tool for model-driven HMI development is presented. This approach shall allow for overcoming the problem of missing tool support while simultaneously enabling manufacturers to face the challenges of the inherent complexity [Mye93] of user interface development. Furthermore, opportunities are provided to meet the requirements stemming from multidisciplinary work and the close co-operation in networked development processes. Meta-modeling offers promising possibilities for the development of non-generic modeling languages which are tailored for the individual requirements of a specific problem domain. In this way the typical problems of misusing tools in application areas significantly differing from the originally intended domain can be avoided: huge effort for adaptation and missing support for familiar development methods including the corresponding notations [BZ06].

4 Developing a Model-Driven Tool-chain for HMI Specification

In order to evaluate the applicability of meta-modeling in complex development processes this approach was used in a pilot project. The aim was to develop specific CASE-tools supporting developers during the HMI development process.
For the development of an appropriate tool support the following requirements were particularly important:

**High problem orientation:** Tool support must be extremely problem-oriented so that even non-experts can read specifications and work with the specific CASE-tools.

**High abstraction level:** The specific CASE-tools shall make system specification possible on an appropriate level of abstraction so that concrete implementation details for example the hard- and software architecture of a target platform or the operating system are hidden from developers.

**Intuitive notation:** The use of graphical tools shall allow developers to specify a system by means of a familiar graphical representation such as illustrated state charts.

**Formal specification:** Developers shall be enabled to create formal specifications which consequently shall allow for the automated generation of executable simulations for in-process evaluation.

Finally, electronic specifications shall be used as a central communication instrument for manufacturers’ in-house developers and moreover for the information exchange with suppliers. The overall aim is to establish formal specifications as the central backbone of a model-driven development process.

In the following the procedure for developing a visual domain-specific language (DSL) is described (Fig. 3). The creation of a specific meta-model by the mapping of domain concepts constituted the starting point for developing a specific CASE-tool for the HMI development process. Beyond the identification and abstraction of domain concepts the development of a visual DSL comprises the definition of notations for the domain concepts’ graphical representation and the definition of constraints for the modeling process. For these tasks meta-CASE tools provide a meta-modeling environment which can subsequently also be used as a modeling environment and therewith as a specification instrument for product development in a specific problem domain.

---

**Figure 3: Development process for domain-specific languages**
At the beginning a small team of domain experts consisting of experienced HMI developers identified the essential concepts of the problem domain. In this step existing requirements documents such as specifications and style guides and particularly the terminology used in daily project work were analyzed. Thus in the case of driver information systems single menu screens of the GUI and controls like rotary knobs and pushbuttons represent main concepts of the problem domain. These concepts could quickly be identified by the domain experts since they are frequently used for product specification. Additionally, the events a system should react to were included, e.g. turning and pressing a rotary knob or pressing and holding a pushbutton respectively. Thereby all properties of every single domain concept necessary for specifying driver information systems were defined. Afterwards constraints were added to the meta-model in order to restrict the degrees of freedom for developers in a reasonable way. Amongst others, the use of some controls was limited to special conditions. For instance, constraints were defined limiting the number of subsequent menu screens after selecting a menu item to at most one. Additional constraints prescribe a fixed pushbutton for return actions. In a final step meaningful pictograms were defined for domain concepts in the meta-model allowing for intuitive identification by developers. The specification of textual content (e.g. menu items) and behavior of a user interface for driver information systems with the DSL is illustrated in Fig. 4.

![Visual domain-specific language for GUI content and behavior](image)

In the pilot project a simulation framework was implemented in Java containing a state machine and base widgets thus covering the static parts of simulations. Consequently, only the dynamic parts, i.e. textual content and behavior, must be linked to the framework to bring automatically generated simulations to life.

For transforming the platform independent models created with the visual DSL into platform specific models or source code a code generator was built. Thereby the gap between domain models and source code necessary for simulations or the final product can be bridged. The main challenge when building a code generator is to define how information

148
can be extracted from models and how domain concepts are mapped onto code. Consequently, carrying out the meta-modeling of domain concepts carefully allows for full code generation of simulation code’s dynamic parts. Although thereby error-prone manual programming cannot be completely eliminated in any case at least a significant reduction is likely to be accomplished. Upon completion of code generation the compilation of dynamically generated code is triggered. Finally, by calling functions provided by a static domain framework, executable simulations can be created without any further activities of developers or programmers. The architecture of the tool-chain is outlined in Fig. 5.

Subsequently, the efficiency of meta-modeling and available meta-CASE-tools to build specific tool support for the domain of HMI development is discussed.
5 Lessons Learned

In the pilot project the development of non-generic CASE-tools and domain-specific languages has proven that meta-modeling must be considered as an interesting approach offering manufacturers promising possibilities to build tailor-made tool support for model-driven development processes at affordable cost. The following discussion on meta-modeling tools, namely the meta-CASE tools METAEDIT+ 4.0 (MetaCase) and GENERIC MODELING ENVIRONMENT (GME) 5 (Vanderbilt University) and in addition the standard office application VISIO 2003 (Microsoft), is based on experience with the establishment of model-driven engineering in an HMI development process in general and the development of visual DSLs for the HMI specification domain in particular. Thereby the discussion is split into the dimensions expressiveness and usability. Whereas the first dimension refers to the amount of problem knowledge embodied in a domain model the latter incorporates the effectiveness, efficiency, and satisfaction [ISO98] with which users can fulfill a task. The evaluation of the selected meta-CASE tools shows that in general designing visual DSLs and thus developing specific tool support is reasonably straightforward in comparison to building a CASE-tool from scratch. At the beginning of this process experienced domain experts together with a programmer have to identify and abstract the essential domain concepts resulting in a domain meta-model. This process is supported by the meta-CASE tools METAEDIT+ and GME respectively.

5.1 Expressiveness

With respect to the expressiveness dimension both tools were comparably powerful within the scope of the pilot project. Since their meta-metamodeling languages MetaGME [Eme04] (GME) and GOPRR [SLTM91] (METAEDIT+) provide mechanisms for describing entities, attributes and relationships as well as concepts for information reuse (e.g. modularization and inheritance) both meta-CASE tools were capable of creating meta-models completely reflecting the identified domain concepts. Nevertheless, there is evidence that further refinement could be advisable. For instance, in METAEDIT+ the possibilities for defining complex constraints are limited. Thus it was impossible to define a minimum of connections between entities ensuring that a certain object is at least connected with a minimal number of corresponding instances. Furthermore, it was not possible to define constraints prescribing that relationships between entities should depend on specific values of particular object properties. In this respect GME offers more flexibility due to the implementation of the UML’s object constraint language (OCL) [ALP05].

5.2 Usability

Regarding usability aspects the evaluated meta-CASE tools reveal potentials for further enhancements both on the meta-modeling and the modeling level. Thus in METAEDIT+
meta-modeling must be carried out textually by means of numerous dialogs. In contrast
GME allows for meta-modeling via direct manipulation of graphical objects using the
drag-and-drop paradigm. Nevertheless, constraints also have to be defined textually by
specifying OCL expressions. Therefore, integrated graphical support for building OCL
expressions as proposed in the V1SUALOCL project [FHTW05, TFS04] would be ben-
eficial. Concerning modeling issues – that is to say the instantiation of meta-models –
the evaluation in daily project work has proven that the graphical representation of do-
main concepts was of major impact for alleviating HMI developers’ reservation against
the unfamiliar paradigms of DSLs and model-driven specifications. In particular, the di-
rect manipulation of objects relevant to their current task context gives developers the
impression of operating with real objects. Moreover, by means of visual DSLs developers
are enabled to create specifications in an intuitive way by using objects corresponding to
their anticipated mental model [Jac86]. Consequently, due to the close semantic distance
between domain concepts and their graphical representation in the DSL the acceptance of
this new approach was significantly increased.

Moreover, support for established and familiar workflow was essentially important for ac-
ceptance among developers. Thus it soon became apparent that developers were hardly
willing to resign functionality of traditional specification tools. For instance, a frequently
used specification instrument is the insertion of comments and memos. While on the meta-
modeling level it is obviously possible to provide a DSL with textual notes and an appro-
priate graphical representation, the evaluated meta-CASE tools did not possess means for
adding graphical comments such as rough sketches which do not possess any semantic
meaning. Unfortunately, developers make heavy use of this feature in current projects.
Experience has shown that in daily work such flexibility is so vitally important for quickly
deciding on design alternatives that – if at all – it can only very slowly be offset by the
indisputable advantages of more formal specifications if these have to be created with less
easy-to-use tools.

Furthermore, developers strongly demanded the implementation of familiar interaction
patterns of standard office applications and current Integrated Development Environments
(IDEs). These provide e.g., discretely colored grids for aligning objects, tree views for
exploring object hierarchies and inspectors for object properties. Moreover also powerful
auto layout for complex diagrams would be desirable. While GME seems to be superior
with respect to the mentioned requirements an editor for the definition of domain con-
cepts’ graphical representation is missing. Although GME supports the assignment of
bitmap files to abstracted domain concepts, METAEDIT+ with its integrated symbol editor
was considered superior in this respect. But, the features of this rudimental symbol editor
are limited to only elementary geometric and free-form shapes. Unfortunately, this imple-
mentation appears to be too cumbersome at first glance. Even at closer inspection support
for complex constraints and dependencies of domain concepts’ representation is missing.
5.3 Enhanced features

Besides GUI related topics also enhanced features for model checking could be desirable. While METAEDIT+ validates at modeling time (online) this has to be explicitly triggered in GME. None the less, further features such as deadlock verification for state machines would provide additional benefits. Although such functionality could be implemented by leveraging scripting languages or application programming interfaces integrated solutions would be more comfortable. Moreover, the assessed tools offered no support for round-trip engineering from models to source and vice versa. The selected tools also provide only very limited support for debugging at the model level. This is a straightforward requirement when aiming for increasing the abstraction level of product development processes. Finally, for a seamless integration of model-based approaches in real-life projects solutions for version control at the model level is badly needed.

![Figure 6: Evaluation of meta-modeling tools and standard office applications by means of expressiveness and usability](image)

Especially due to the usability restrictions mentioned above VISIO was considered as an alternative to specialized meta-CASE tools. This standard office application was chosen due to its interaction philosophy coming very close to POWERPOINT’s – with its outstanding usability as stated by HMI developers – while at the same time offering the possibility to save diagrams in a XML format. Since serialization is crucial for the use of machine-readable specifications along a model-driven development process it is also an important criterion for the selection of a suitable modeling environment. Although VISIO’s limited expressiveness began to show almost immediately on the meta-modeling level developers nevertheless got obviously used to this modeling tool much faster than to one of the alternative meta-CASE tools. Figure 6 shows the results of a survey among developers and programmers on the basis of the above-mentioned dimensions².

²Since POWERPOINT 2003 cannot fulfill the rudimentary requirements on the expressiveness dimension it is only included for comparison.
Accordingly, it must be stated that in the pilot project none of the evaluated tools could fully meet the requirements for tool support in our model-driven HMI development process regarding expressiveness and usability. While on the one hand METAEDIT+ and GME have proven to be sufficiently powerful for mapping domain concepts of complex problem domains limitations on the definition of constraints and graphical representations as well as numerous usability issues still remain. On the other hand the standard office application VISIO fails to meet the expressiveness demands while in contrast clearly satisfying developers’ usability requirements. Although these findings must be treated carefully – since particularly VISIO’s strong acceptance might be superposed by its familiarity in consequence of frequent use in daily work – the overall conclusion is on the horizon: for meta-modeling to gain broader acceptance beside sufficient expressiveness meta-CASE tools need to provide satisfying easy-of-use.

6 Conclusion

The aim of the outlined pilot project was to evaluate the applicability of model-based approaches for a model-driven HMI development process. Experience shows that meta-modeling can provide valuable benefit for improving processes and tool support. The key factors for possible process improvements are:

**Abstraction:** The development problem has to be solved only once on a high level of abstraction.

**Focus on problem domain:** Developers can work on the development task with the concepts of the problem domain. Thereby, complexity can be hidden – although not necessarily reduced – and developers can use their familiar terminology.

**Transparency:** The knowledge of domain experts is explicitly kept in the domain-specific language. Thus, the DSL enables new members of a development team to become acquainted with the concepts and constraints of a specific domain more easily.

In addition to these process-oriented improvements meta-modeling provides manufactures opportunities for building individual tool support at arguable cost. Particularly, when building CASE-tools for well-defined problem domains productivity gains can be achieved compared to conventional programming.

Despite these promising beginnings the evaluation of the selected meta-CASE tools reveals that further enhancements are possible. While the assessed tools meet the expressiveness requirements of the HMI problem domains, better usability for meta-modelers and modelers in particular is inevitable. Since – apart from effectiveness and efficiency – the ease-of-use is most crucial for the acceptance of tools among developers usability issues deserve special attention. Hence, the additional incorporation of carefully selected interaction patterns and the look-and-feel of standard office applications and/or IDEs could be an important step for achieving the right balance between expressive power and usability. With the help of such tool kits the odds are good that manufacturers are enabled to lever-
age the obvious advantages of model-driven approaches for overcoming today’s urgent problems in complex and networked development processes.

References


