MODEL-DRIVEN HMI DEVELOPMENT – CAN META-CASE TOOLS RELIEVE THE PAIN?

Carsten Bock
Dr. Ing. h.c. F. Porsche AG
Stuttgart, Germany

Detlef Zuehlke
German Research Center for Artificial Intelligence DFKI GmbH
Center for Human Machine Interaction
PO Box 3049, Kaiserslautern 67653, Germany

Keywords: Model-driven useware engineering, electronic specification, model-driven HMI development.

Abstract: Today metamodeling and domain-specific languages represent many promising beginnings to create non-generic tool support for individual modelling tasks. Due to the inherent complexity and numerous variants of human-machine interfaces (HMIs) model-driven development becomes increasingly interesting for manufacturers and suppliers in the automtive industry. Particularly, the development of powerful user interfaces requires appropriate development processes as well as easy-to-use software tools. Since suitable tool kits are missing in the field of HMI development this paper describes the utilization of visual domain-specific languages for model-driven useware engineering in general and model-based specification of automotive HMIs in special. Moreover, results from a survey among developers are presented revealing the requirements for HMI specific tool support. Additionally, experiences with using current meta-CASE tools as well as standard office applications for creating a visual domain-specific language are presented. Based on these experiences requirements for future meta-CASE tools are derived.

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 (HMIs) coming along with the rapid development of computational power can be perfectly retraced by the example of the automotive industry. In the 1960s average passenger cars were equipped with an odometer while 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 systems including e.g. AM/FM tuners, compact disks, iPods, mobile phones, SMS message and e-mail support as well as navigation systems. Moreover, assistance 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 examples mentioned above: all functionality must be easily accessible for users by means of intuitive HMIs. Especially in the automotive industry the development of appropriate HMIs deserves special attention since road safety must never be endangered by using HMIs (Rudin-Brown, 2005). Therefore, within the scope of operating concept development both hard- and software engineering play a key role for user acceptance and market success of interactive dialog systems.

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 as well as the results of a survey among HMI developers 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.

Bock C. and Zuehlke D. (2006).
MODEL-DRIVEN HMI DEVELOPMENT – CAN META-CASE TOOLS RELIEVE THE PAIN?.
In *Proceedings of the First International Conference on Software and Data Technologies*, pages 312-319
DOI: 10.5220/0001322403120319

2 EFFICIENT DEVELOPMENT PROCESSES FOR POWERFUL HUMAN-MACHINE-INTERFACES

When successfully developing useware[1] (Oberquelle, 2002; Zuehlke, 2002a) 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. 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 is crucial for successful cooperation. This challenge is especially aggravated since developers from different fields of activity with different knowledge and experience typically use different methods and tools. Consequently, intuitive notations for all kinds of information must be established in order to achieve a common understanding among team members. 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 landscapes inevitably result in myriad media disruptions. These can only be overcome by transforming formats with the help of conversion tools.

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 specifications into virtual prototypes or even 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 this manual work ideas and concepts can only be evaluated with simulations or prototypes in relatively late project phases of the development process.

Experience shows that this is typically insufficient for an early comparison with customers' and decision makers' expectations (Fitton et al., 2005). Moreover, updating cycles of specifications take so much time that adjustments to rapidly changing requirements can only be accomplished with great efforts. Since changes can only be incorporated into the product in a rush without updating the specification paper-based documentation typically becomes less important in the final phases of development projects. 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 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 times.

3 TOOL SUPPORT FOR COMPLEX NETWORKED DEVELOPMENT PROCESSES

Although in the 1980s and 1990s 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 as well as the generic graphical representation of its modelling constructs (Schmidt, 2006). Especially in the field of HMI development CASE-tools could not establish their market position because the UML particularly lacks adequate instruments for specifying Graphical User Interface (GUI) (Blankenhorn and Jeckle, 2004). Obviously, sophisticated and established tools like Simulink or LabView are available in specific fields of application like electronic control unit (ECU) development.

However, appropriate tool support for HMI developers is still missing. This is due to the relatively small market volume of this business segment (Ledeczi et al., 2001). Hence, manufacturers are forced to develop non-generic computer-aided tools – and toolchains 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 spe-

[1] Useware 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 (Zuehlke, 2002b).

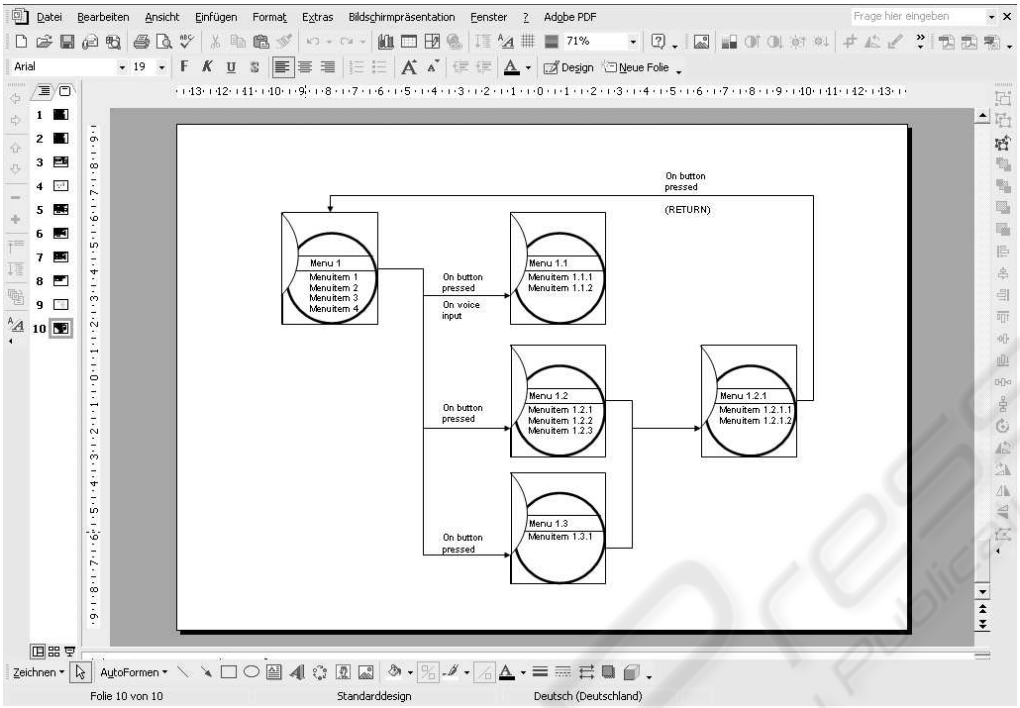

Figure 1: Illustrated state chart for HMI specification of an automotive instrument cluster in POWERPOINT.

cific methods and tools such as illustrated state charts (Hamberger et al., 2003) (Fig. 1).

Despite the apparent demand for domain-specific tool support and its benefits tenacious obstacles seem to exist. Especially the high cost of proprietary software development and a considerable development risk hinder the broad acceptance and employment of domain-specific CASE-tools (Spinellis, 2001). 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 (Myers, 1993) of user interface development. Furthermore, opportunities are provided to meet the requirements stemming from the multidisciplinarity and the close cooperation in networked development processes. Metamodelling offers promising possibilities for the development of non-generic modelling 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 (Bock and Zuehlke, 2006).

4 DEVELOPING A MODEL-DRIVEN TOOLCHAIN FOR HMI SPECIFICATION

In order to evaluate the applicability of metamodelling in HMI development this approach was used in a pilot project. The aim was to develop specific CASE-tools supporting developers in 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 abstraction level. This should hide implementation details from developers such as the hard- and software architecture of a target platform or the operating system.
- *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. These shall allow

for the automated generation of 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 communication with suppliers. The overall aim is to establish formal specifications as the central backbone of a model-driven development process.

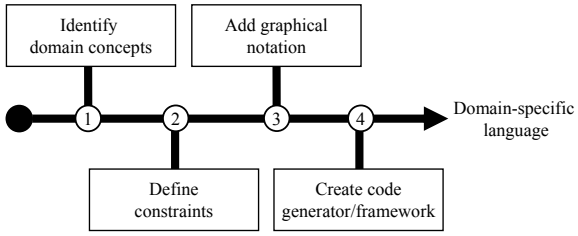

Figure 2: Development process for domain-specific languages.

At the beginning a small team of domain experts consisting of experienced HMI developers identified the essential concepts of the problem domain (Fig. 2). 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. Thereby all properties of every single domain concept were defined. Afterwards constraints were added to the metamodel 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 for 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. These pictograms shall enable developers to intuitively identify domain concepts when using the visual DSL at model time. The specification of textual content (e.g. menuitems) and behavior of a user interface for an instrument cluster with the DSL is illustrated in Fig. 3.

In the pilot project a simulation framework was implemented in JAVA containing a state machine and base widgets to cover 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 can be extracted from models and how domain concepts are mapped onto code.

Consequently, careful metamodelling of domain concepts 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. Finally, executable simulations can be created without any further activities of developers or programmers by calling functions provided by a static domain framework.

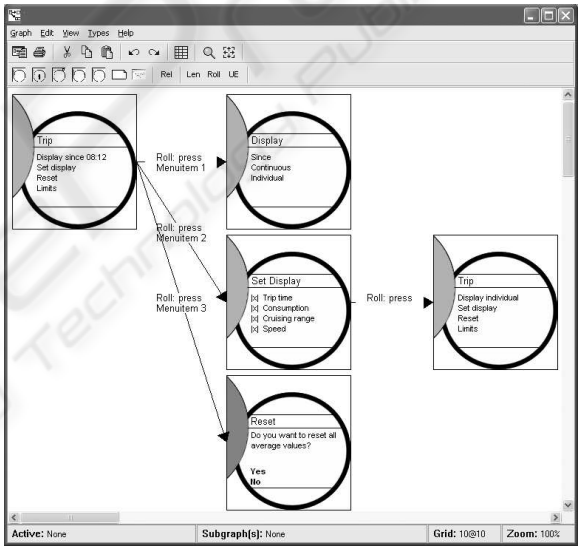

Figure 3: Visual domain-specific language for GUI content and behavior.

Subsequently, the potentials of metamodelling and the efficiency of current meta-CASE tools are discussed. Furthermore, experiences with building a toolchain for model-driven HMI development are presented.

5 META-CASE TOOLS VERSUS STANDARD OFFICE APPLICATIONS

In the pilot project the development of non-generic CASE-tools has proven that metamodelling must be

Table 1: Structure of questionnaire for HMI toolchain requirements.

Category	Criterion	Description
Functionality	Flexibility	Easy adaptation for other development projects
	Modelling course of actions	Specification of course of action with illustrated state charts
	Familiar notation	Developers can specify HMIs with well-known graphical symbols
	Usability	Usability of modelling tool(s)
	Model reuse	Possibility to use models in different HMI projects
	Model validation	Check for deadlocks, basic rules from HMI style guide
	Collaboration	Developers can work simultaneously on a common database
	Role concept	Different view on specification data dependent on developer's role
	Layout menu screens	Possibility to chance GUI details
Automatic processing	Specification transformation	Transform HMI specification into supplier specific, machine-readable format
	Export standard formats	Export HMI specification to WORD, POWERPOINT, PDF etc.
	Test automation	Automated model-based HMI testing
	Usability tests	Support for usability tests (e.g. usage frequency analysis)
	Simulation	Easy creation of virtual prototypes
Infrastructure	Process stability	Maturity of tools used in toolchain
	Use of standards	Non-proprietary programming/scripting languages, machine-readable export formats
	Interfaces	Possibility to exchange specification data in toolchain via API
	Documentation & Support	Availability of documentation and support for tools used in toolchain

considered as a promising approach for building tailormade tool support for model-driven development processes at affordable cost. Therefore, HMI developers' requirements for a toolchain were in the focus of the pilot project. The following discussion on metamodelling tools – namely the meta-CASE tools METAEDIT+ 4.0 (MetaCase) and GENERAL MODELLING ENVIRONMENT (GME) 5 (Vanderbilt University) and in addition the standard office application VISIO 2003 (Microsoft) – is based on the general results of a survey among HMI developers at Porsche. Moreover, experiences with the development of visual DSLs for automotive HMI development are presented.

5.1 Survey Results

The examination of developers' requirements was the starting point for building a toolchain which is capable of supporting HMI developers in all phases of the development process. In order to develop a deep understanding of developers' demands structured interviews were conducted by means of a questionnaire (Tab. 1). Developers had to assess 18 criteria on an ordinal scale with the attribute values "very important", "important" and "less important". Table 2 shows the overall result and a prioritization of the criteria.

Most notably the criterion usability was not among developers' top priorities. At first glance this result was surprising, since the development of usable automotive HMIs is the daily business for developers. Hence, a strong demand for usable tool support was expected. Nevertheless, this antilogy can be clarified by stating that developers intuitively assumed POWERPOINT and VISIO as a benchmark. During the interviews developers pointed out repeatedly that every tool with POWERPOINT's or VISIO's look and feel and interaction patterns would be suitable to be used in a toolchain. Thus, it has to be concluded that usability is not explicitly claimed as a desired attribute, but nonetheless stipulated in advance as an essential and inherent feature of appropriate tool support.

Table 2: Prioritized requirements for HMI toolchain.

Priority	Feature
Very important	Model reuse
	Modelling course of actions
	Familiar notation
	Usability tests
	Simulation
	Collaboration
	Process stability
Important	Flexibility
	Usability
	Model validation
	Use of standards
	Interfaces
	Export standard formats
Less important	Specification transformation
	Documentation & Support
	Layout menu screens
	Test automation
	Role concept

Therefore, usability was explicitely chosen to derive the taxonomy presented in the following section.

5.2 Experiences

The following 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 (International Organization for Standardization, 1998) with which users can fulfil a task. The evaluation of the selected meta-CASE tools shows that in general designing graphical 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 metamodel. This process is supported by the meta-CASE tools METAEDIT+ and GME respectively. With respect to the expressiveness dimension both tools were comparably powerful within the scope of the pilot project. The tools' meta-metamodelling languages MetaGME (Emerson, 2004) (GME) and GOPRR (Smolander et al., 1991) (METAEDIT+) provide mechanisms for describing entities, attributes and relationships. Moreover, concepts for information reuse (e.g. modularization and inheritance) are provided. With these mechanisms and concepts both meta-CASE tools were capable to create metamodels that completely reflect 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) (Alanen et al., 2005).

Regarding usability aspects the evaluated meta-CASE tools reveal potentials for further enhancements both on the metamodelling and the modelling level. Thus in METAEDIT+ metamodelling must be carried out textually by means of numerous dialogs. In contrast GME allows for metamodelling 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 VISUALOCL project (Fish et al., 2005), (TFS, 2004) would be beneficial.

Concerning modelling issues – that is to say the instantiation of metamodels – the evaluation in daily project work has proven that the graphical representation of domain concepts was of major impact to the decrease of HMI developers' reservation against the unfamiliar paradigms of DSLs and model-driven specifications. In particular, the direct 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 (Jacob, 1986). Consequently, due to the close semantic distance between domain concepts and their graphical representation in the DSL the acceptance of this new approach could significantly be increased.

Besides the graphical representation of a DSL the support for established and familiar workflow was important for acceptance 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 metamodelling level it is obviously possible to provide a DSL with textual notes and an appropriate 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 tool flexibility is vitally important. Therefore, tool flexibility can hardly 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 the widespread operating philosophy of standard office applications. Among other things this includes object inspectors, tree views for object hierarchy or discretely colored grids for alignment. Moreover also powerful auto layout for complex diagrams would be desirable. While GME meets the two last-mentioned requirements an editor for the definition of domain concepts' 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 freeform shapes. Unfortunately, this implementation appears to be too cumbersome at first glance. Even at closer inspection support for complex constraints and dependencies or nested graphical objects is missing. Finally, besides GUI related topics also enhanced features for model

checking could be desirable. While METAEDIT+ validates at modelling 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.

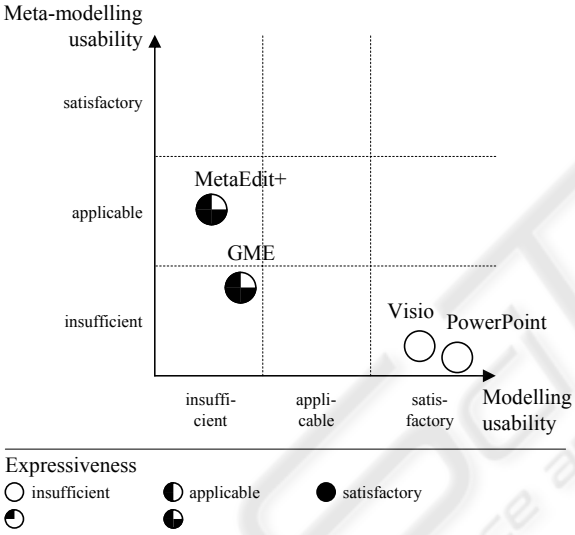

Figure 4: Evaluation of metamodelling tools by means of expressiveness and usability.

Due to the usability restrictions mentioned above VISIO was considered as an alternative to specialized meta-CASE tools. This standard office application was chosen because its operating philosophy comes 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 modelling environment. Although VISIO's limited expressiveness began to show almost immediately on the metamodelling level developers nevertheless got obviously used to this modelling tool much faster than to one of the alternative meta-CASE tools. Figure 4 shows the results of a survey among developers and programmers on the basis of the above-mentioned dimensions.

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 metamodelling to gain broader acceptance meta-CASE tools need to provide sufficient expressiveness while simultaneously offering 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 metamodelling 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 metamodelling provides manufacturers 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 meta-CASE tools meet the expressiveness requirements of

the HMI problem domains better usability for metamodellers and modelers in particular is inevitable. Apart from effectiveness and efficiency the ease-of-use is most crucial for the acceptance of tools among developers. Consequently, usability issues deserve special attention. Hence, the additional incorporation of carefully selected interaction patterns and the look-and-feel of standard office applications could be an important step for achieving the right balance between expressive power and usability.

Finally, such tool kits would allow developers to utilize the obvious advantages of model-driven approaches in their daily development work. This would enable manufacturers and suppliers to overcome today's urgent problems with complex and networked development processes.

REFERENCES

Alanen, M., Lundkvist, T., and Porres, I. (2005). GXL and MOF: A Comparison of XML Applications for Information Interchange. Available from: `http://www.tucs.fi/publications/attachment.php?fname=inpAlLuPo05a.pdf` [cited 11.7.2006].

Blankenhorn, K. and Jeckle, M. (2004). A UML profile for GUI layout. In Weske, M. and Liggesmeyer, P., editors, *Object-Oriented and Internet-Based Technologies, Net.ObjectDays 2004, Erfurt, Germany, September 27-30, 2004*, pages 110–121. Springer.

Bock, C. and Zuehlke, D. (2006). Non-generic tools support for model-driven product development. *atp – Automatisierungstechnische Praxis*, 48(7):42–48.

Emerson, M. (2004). GME-MOF: The MOF-based GME Metamodeling Environment. In *Model-Integrated Computing Workshop*. Available from: `http://www.omg.org/news/meetings/workshops/MIC_2004_Manual/03-1_Emerson_etal.pdf` [cited 13.5.2006].

Fish, A., Howse, J., Taentzer, G., and Winkelmann, J. (2005). Two visualizations of OCL: A comparison. Available from: `http://www.cmis.brighton.ac.uk/research/vmg/VOCLTR.html` [cited 27.6.2006].

Fitton, D., Cheverst, K., Kray, C., Dix, A., Rouncefield, M., and Saslis-Lagoudakis, G. (2005). Rapid prototyping and user-centered design of interactive display-based systems. *IEEE Pervasive Computing*, 4(4):58–66.

Hamberger, W., Deutler, P., and Bouaziz, T. (2003). *Audi Multi Media Interface (MMI) – Von der Idee zum Produkt: Interdisziplinär – Prozessorientiert – Modellreihenübergreifend*, volume 1789 of *VDI-Berichte*, pages 1175–1191. VDI.

International Organization for Standardization (1998). ISO 9241-11:1998 : Ergonomic requirements for office work with visual display terminals (VDTs) – Part 11: Guidance on usability.

Jacob, R. J. K. (1986). A specification language for direct-manipulation user interfaces. *ACM Trans. Graph.*, 5(4):283–317.

Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., and Karsai, G. (2001). Composing domain-specific design environments. *Computer, IEEE*, 34(11):44–51.

Myers, B. A. (1993). Why are human-computer interfaces difficult to design and implement? Technical Report CMU-CS-93-183, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, USA.

Oberquelle, H. (2002). *Useware Design and Evolution: Bridging Social Thinking and Software Construction*, pages 391–408. MIT-Press, Cambridge, London.

Rudin-Brown, C. (2005). Strategies for reducing driver distraction from in-vehicle telematics devices: Report on industry and public consultations. Research Report TP 14409 E, Transport Canada, Road Safety and Motor Vehicle Regulation Directorate.

Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31.

Smolander, K., Lyytinen, K., Tahvanainen, V.-P., and Marttiin, P. (1991). MetaEdit: A flexible graphical environment for methodology modelling. In *CAiSE '91: Proceedings of the third international conference on Advanced information systems engineering*, pages 168–193, New York, NY, USA. Springer.

Spinellis, D. (2001). Notable design patterns for domain-specific languages. *J. Syst. Softw.*, 56(1):91–99.

TFS (2004). VisualOCL – Editor plugin for Eclipse. Available from: `http://tfs.cs.tu-berlin.de/vocl/` [cited 25.7.2006].

Zuehlke, D. (2002a). USEWARE – Herausforderung der Zukunft. *atp – Automatisierungstechnische Praxis*, 44(9):73–77.

Zuehlke, D. (2002b). Useware forum. Available from: `http://www.useware-forum.de` [cited 13.6.2006].