Using Domain-Specific Languages in the Design of HMIs: Experiences and Lessons Learned

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

Daniel Görlich
Center for
Human-Machine-Interaction
University of Kaiserslautern
P.O. Box 3049, 67653
Kaiserslautern, Germany
+49(631)205-3706
goerlich@mv.uni-kl.de

Detlef Zühlke
German Research Center for
Artificial Intelligence (DFKI)
Center for
Human-Machine-Interaction
P.O. Box 3049, 67653
Kaiserslautern, Germany
zuehlke@mv.uni-kl.de

Keywords

Model-driven useware engineering, user-centered HMI development, visual domain-specific language, HMI tool chain

ABSTRACT

Usability has become a decisive factor for successful product development and operation of human-machine systems. In order to meet the customers requirements, interdisciplinary development teams have to work together in a systematic and iterative useware engineering process. Furthermore, early integration of users and customers becomes indispensable, which implies the rapid and on-demand creation of demonstrative prototypes even in early phases of the development process. This paper presents the procedure for developing a domain-specific language (DSL) intuitively understandable to all members of the interdisciplinary teams, and the continued reutilization of this DSL in meta-CASE tools and GUI-builders able to generate code fragments and HMI simulation prototypes on the push of a button.

1. USEWARE ENGINEERING – INTERDISCIPLINARY AND ITERATIVE

Nowadays manufacturers usually have to face ambitious development tasks when successfully developing useware[1]. So as to meet these challenges the expertise of numerous developers from different disciplines is required just as are effective and efficient development processes. Furthermore, early user integration is particularly important to avoid that products fail to meet customers' requirements. Thus evolutionary development processes are necessary supporting iterated user integration during product development.

[1]Useware includes all hard- and software components of a technical system, which are required for its use. The expression *useware* has been created to demonstrate the equal importance of human-machine-systems compared to hard- and software [9].

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MDDAUI '06 Genova, Italy
Copyright 2006 ACM 0-89791-88-6/97/05 ...$5.00.

Our model-driven and user-centered HMI development process comprises five partially interacting and overlapping phases (Fig. 1):

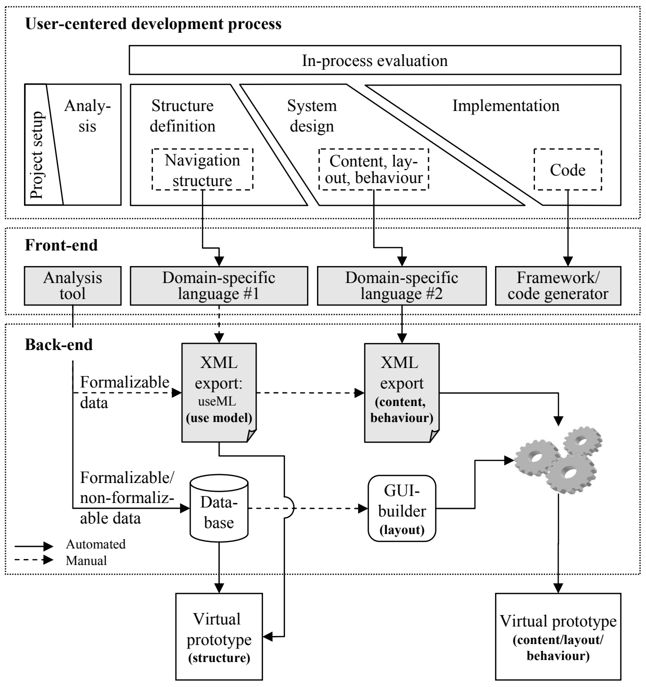

User-centered development process
In-process evaluation
Project setup
Analy-sis
Structure definition
Navigation structure
System design
Content, lay-out, behaviour
Implementation
Code
Front-end
Analysis tool
Domain-specific language #1
Domain-specific language #2
Framework/ code generator
Back-end
Formalizable data
XML export: useML (use model)
XML export (content, behaviour)
Formalizable/ non-formaliz-able data
Data-base
GUI-builder (layout)
Automated
Manual
Virtual prototype (structure)
Virtual prototype (content/layout/ behaviour)

Figure 1: User-centered HMI development process with specification front-ends and back-end architecture

Analysis: The analysis of users and their behaviour as well as the context of use can be carried out with different methods [10]. Thereby developers shall get an overall idea of tasks, user groups and possible working environments. To ensure an accurate documentation of all available information, results are stored in a structured database reflecting user requirements and environmental conditions.

Structure definition: The aim is to derive a platform-independent use model containing the navigation structure of the system under development. In order to get machine-readable information, use models are described with a markup language, namely *useML* [6].

System design: The purpose of this phase is a mapping of the use model onto a specific hardware platform. Besides hardware issues particularly software related topics such as graphical user interface (GUI) development deserve special attention. According to the Seeheim model three categories constitute integral parts of GUI development: (graphical) layout, content and behaviour. Thereby, layout relates to the ergonomic arrangement of dialog objects, content refers to the definition of information to be displayed. Finally, behaviour describes the dynamic parts of a GUI with respect to controls available on a specific system.

Implementation: In this phase the results of the previous conceptual phases are implemented by merging hardware and software specifications.

In-process evaluation: In addition to these sequential phases evaluations are conducted. In a strict HMI development process iterative successful evaluations are mandatory for passing project milestones.

Experience with this development process reveals that the earlier simulations are available, the more effective user integration will be. But, the demand for early (virtual) prototypes following from user-centered development approaches calls on manufacturers to acquire knowledge in the field of developing interactive systems. Since this is normally not among manufacturers' core competences, easy-to-use front-ends and a powerful back-end are necessary for creating machine-readable specifications [3]. By means of code generators and (meta-)modelling frameworks these specifications provide manufacturers the opportunity to create executable simulations at the push of a button. Thus manufacturers are enabled to use simulations for evaluations in early phases of product development processes.

2. TOOL SUPPORT FOR USER-CENTERED HMI DEVELOPMENT

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. Current meta-CASE tools try to bridge this gap by leveraging meta-modelling for tool creation. This can enable manufacturers to create specific tool support at manageable costs. For the development of appropriate tool support the following requirements are particularly important [1]:

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

High abstraction level: Non-generic 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.

Formal specification: Developers shall be enabled to create formal specifications. These shall allow for the au-

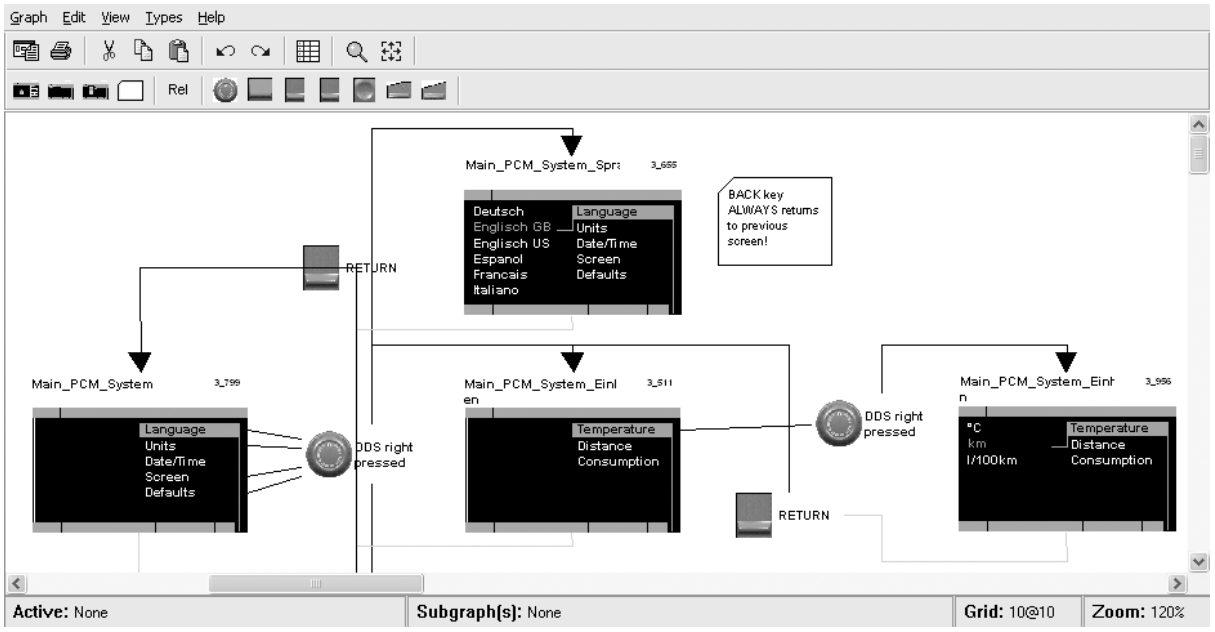


Figure 2: Domain specific language for specifying content and behaviour of HMIs in system design phase (implemented with meta-CASE tool MetaEdit+ 4.0)

tomated generation of simulations for in-process evaluation.

Subsequently, the procedure for developing a visual domain-specific language (DSL) for an automotive driver information system is described. Since the procedure [2] is generic for all problem domains, the description only refers to the development of a DSL for the system design phase of the previously outlined HMI development process.

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 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 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 metamodel allowing for intuitive identification by developers. The specification of textual content (e.g. menuitems) and behaviour of a user interface for driver information systems with the DSL is illustrated in Fig. 2.

Moreover, meaningful pictograms are defined for domain concepts in the meta-model allowing for intuitive identification by developers when using the DSL for creating specifications. The resulting DSL for the system design phase (Fig. 1, front-end: DSL #2) is shown in Fig. 2.

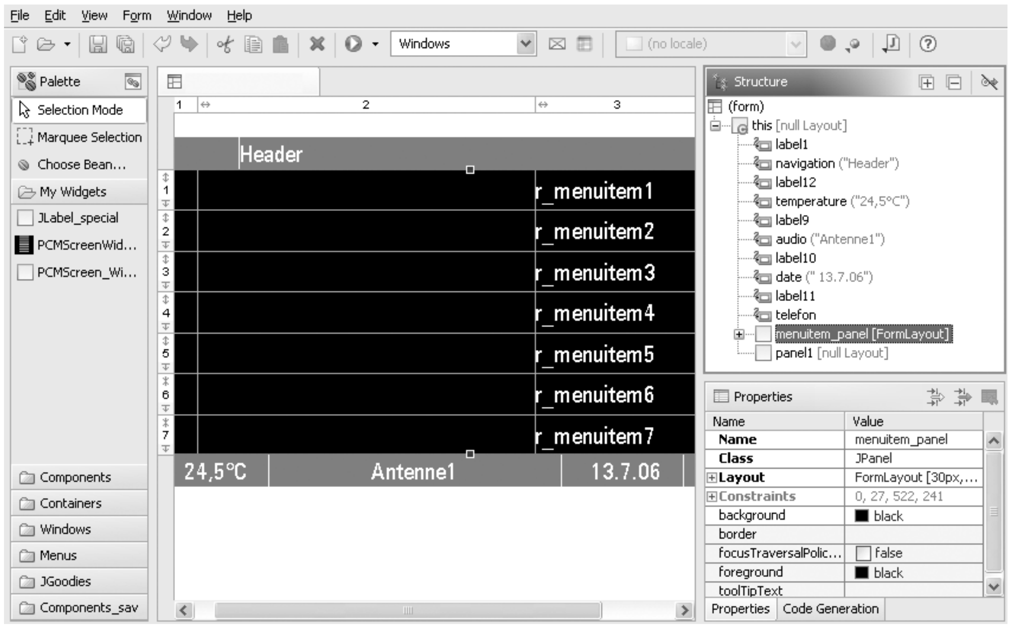
For these tasks meta-CASE tools such as MetaEdit+ [7], Generic Modelling Environment (GME) [4] or Microsoft DSL Tools [5] provide each a graphical user interface which can subsequently also be used as a modelling environment and thus as a specification tool for product development.

Additionally, 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 behaviour, 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, i.e. source code, a code generator was built. 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, carrying out the metamodeling 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.

3. FUTURE WORK

Although first experiences with the presented development process and suitable tool-chains were promising, several issues remain for further refinement. Thus endeavours have begun to support the pictured HMI development



```
(form)
this [null Layout]
    label1
    navigation ("Header")
    label12
    temperature ("24,5°C")
    label9
    audio ("Antenne1")
    label10
    date (" 13.7.06")
    label11
    telefon
    menuitem_panel [FormLayout]
    panel1 [null Layout]
```

Name	Value
Name	menuitem_panel
Class	JPanel
Layout	FormLayout [30px,...
Constraints	0, 27, 522, 241
background	■ black
border	
focusTraversalPolic...	☐ false
foreground	■ black
toolTipText	

Figure 3: GUI-builder for layout specification

process with a dedicated tool-chain removing the restrictions of partially incompatible third-party tools requiring repeated manual re-entering of data and development of converters. Instead, all data gathered from the earliest analyses to the final prototype shall be collected in a project database that establishes a sound basis for three individual tools supporting the analysis, structure definition, and system design phases severally. Split up into several dedicated tools, the tool-chain will be easier to use both for IT-experts and other domain experts in interdisciplinary development teams; yet, the project database ensures that media breaches between the processes phases are eliminated without losing data.

Moreover, the development teams' work shall be supported by a knowledge base comprising rules, guidelines, style guides, (e.g. graphical) component libraries, and more. While in its earlier stages, this knowledge base will only provide human-readable text documents with indices and cross-references, semantically formalizable content shall later enable software agents and assistants to monitor the development teams' work and hint to potential problems, offences against rules and style guides et cetera.

While the development of this knowledge base has just begun, a tool supporting the analysis of users, their tasks, etc., is already nearing its completion. This tool is able to export results from analyses into rudimental use models employing the Useware Markup Language (useML). Already providing the navigation structure of the HMI system to be built, down to elementary use objects, together with the DSL #2 (see Fig. 2), a first simulation prototype can be rendered automatically based on the layout, content and basic behaviour, when appropriately defined in the system design.

Currently, the JAVA GUI-builder JFORMDESIGNER [8] is used for specifying static GUI components, especially the overall layout of menu screens (Fig. 3). Mainly, this includes the pixelwise specification of the position and the dimension of headers and footers or information and warning areas. Also colors are specified with this GUI-builder. In our present tool-chain this graphical work has to be done twice: first with the GUI-builder in order to use the specification for creating virtual prototypes with the help of the framework and the code generator. Second, the (rough) layout of the menu screens has to be created in the meta-CASE tool again for building the graphical notation of DSL #2. In order to avoid this extra work we actually investigate possibilities for a closer integration of the GUI-builder and the meta-CASE tool that should allow for automatic synchronization of both components.

Finally, for a seamless integration of all back-end components (analysis tool, DSLs, GUI-builder, simulation framework) an application bus has to be created. This bus shall supersede the manual data transfer in our current back-end architecture. Moreover, possibilities for an integration of third-party tools such as XAML GUI-builders shall be investigated. Thereby the automatic derivation of the meta-model of DSL #2 (system design) becomes possible. In general the automatic synchronization of models will be an important final step towards full code generation in complex HMI development processes.

4. CONCLUSION

Challenged by the need for usable HMI systems in classic and upcoming fields of application, these systems have to be developed quickly and efficiently by interdisciplinary teams comprising IT-experts and other domain experts. They are in need of computer tools supporting their work by adapting to their project domains' specific requirements, language, and thinking. The presented tool-chain supports such a systematic useware development process based on domain specific languages and meta-models appropriate to the respective project. It bridges media breaches in the development process by using a project database, thereby superseding the currently error-prone manual transmission of data and specifications.

Though much work has still to be done, the fundament for the tool-chain has been laid and a proof of concept is currently being conducted. The development of the main parts of the outlined tool-chain shall be finished by the next two years, but will be improved subsequently e.g. by formalizing more knowledge semantically to semi-automate the development process and let software agents assist human developers with their work.

5. REFERENCES

[1] C. Bock and D. Zühlke. Meta-modelling – A Silver Bullet for Model-Driven HMI Development? In *WoMM: 2nd International Workshop on Meta-modelling and Ontologies, October 12-13, Karlsruhe, Germany*, 2006.

[2] C. Bock and D. Zühlke. Model-Driven HMI Development – Can Meta-CASE Tools Relieve the Pain? In J. Filipe, B. Shishkov, and M. Helfert, editors, *ICSOFT 2006: 1st International Conference on Software and Data Technologies, September 11-14, Setúbal, Portugal*, volume 2, pages 312–319, 2006.

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

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

[5] Microsoft. Domain-Specific Language Tools [online]. 2006 [cited 15.9.2006]. Available from: `http://msdn.microsoft.com/vstudio/DSLTools/`.

[6] K. Mukasa and A. Reuther. The Useware Markup Language USEML – Development of user-centered interfaces using XML. In *9th IFAC Symposium on Analysis, Design, and Evaluation of Human-Machine Systems, September 7-9, Atlanta*, 2004.

[7] K. Smolander, K. Lyytinen, V.-P. Tahvanainen, and P. Marttiin. MetaEdit: A flexible graphical environment for methodology modelling. In *CAiSE '91: Proceedings of the 3rd International Conference on Advanced Information Systems Engineering*, pages 168–193, New York, NY, USA, 1991. Springer.

[8] K. Tauber. JFormDesigner documentation [online]. 2005 [cited 29.3.2006]. Available from: `http://www.jformdesigner.com/download/JFormDesignerDoc-2.0.1.pdf`.

[9] D. Zühlke. Useware forum [online]. 2002 [cited 13.6.2006]. Available from: `http://www.useware-forum.de`.

[10] D. Zühlke. *Useware-Engineering für technische Systeme*. Springer, Berlin, 2004.