Proceedings of the First International Conference on Automotive User Interfaces and Interactive Vehicular Applications
(AutomotiveUI 2009), Sep 21-22 2009, Essen, Germany

Towards a Flexible UI Model for Automotive Human-Machine Interaction

Guido de Melo, Frank Honold, Michael Weber
University of Ulm
Institute of Media Informatics
89081 Ulm
{guido.de-melo, frank.honold, michael.weber}@uni-ulm.de

Mark Poguntke, André Berton
Daimler AG
Research and Development
Infotainment and Telematics
89081 Ulm
{mark.poguntke, andre.berton}@daimler.com

ABSTRACT
In this paper we present an approach for creating user interfaces from abstract representations for the automotive domain. The approach is based on transformations between different user interface abstraction levels. Existing user interface representation methods are presented and evaluated. The impact of specific requirements for automotive human-machine interaction is discussed. Considering these requirements a process based on transformation rules is outlined to allow for flexible integration of external infotainment applications coming from mobile devices or web sources into the in-car interaction environment.

Categories and Subject Descriptors
H.5.2 [**Information Interfaces and Presentation**]: User Interfaces

Keywords
human-machine interaction, user interface modeling, user interface generation, UML, Cameleon Reference Framework

1. INTRODUCTION
1.1 Overview
The increasing development and ubiquity of infotainment applications plays an important role for automotive manufacturers. Mobile devices, like smart phones or mp3-players, are widespread and increasingly used. In an in-car environment the use of external devices distracts the driver from the important task of driving. Also, there are legal regulations in many countries that prohibit the use of mobile devices while driving. The convenient integration of different external devices and services, such as smart phones and on-line services, in the in-car environment is desired. Since the development time and life cycle of automotive software is usually much longer than the life cycle of consumer electronics or web-based services, a flexible solution for the adaption of new applications to the existing automotive human-machine interface (HMI) is required. In order to integrate applications of different devices to the automotive HMI, it is necessary to map the device capabilities to the interaction devices within the automotive environment, e. g. central control unit and head-unit display. Additionally, voice control of external applications should be possible via the in-car speech dialog system.

Each automotive manufacturer provides their own HMI specified by colors, images, font styles, interaction concepts and flows. Since there are important automotive-specific requirements like font size settings for minimal driver distraction, the aim is to provide an appropriate HMI concept meeting these requirements, e. g. according to the European Statement of Principles on HMI for in-vehicle information and communication systems (ESoP). Also, for safety reasons, and as a distinctive feature, the control over the automotive HMI has to be completely handeled by the car software. The latter is hard to achieve if external applications are to be integrated which provide their own user interfaces. More abstract representations of user interface concepts build a basis for different concrete user interfaces. Devices should provide abstract descriptions of their functionality and capabilities. And the head-unit would transform these descriptions to the automotive HMI as needed. Thus, the integration of external devices would be possible after deployment of the car software, and the integrated system would still be controlled by the manufacturer.

1.2 Scenario
The simple example of integrating a portable music player into the head-unit HMI illustrates important issues to be solved. The driver or passenger intends to use any player in the car. This may be a very simple device providing basic audio player functionalities or a more complex device providing additional features like album cover presentations, different playlists, a dynamic play order based on similarity of songs and so on. Current car head-units provide their own audio player with mp3-support. Connecting an mp3-player to the car will initiate the car's audio player which is then used to play the songs contained on the device. However, device-specific interaction features are not integrated if they are unknown to the head-unit. Furthermore, completely unknown applications like a calendar cannot be integrated at all.

Copyright held by authors
AutomotiveUI'09, September 21-22, 2009, Essen, Germany
ACM 978-1-60558-571-0/09/009

2. ABSTRACT USER INTERFACE REPRESENTATIONS

In order to achieve a seamless integration, the device capabilities and interaction possibilities have to be transferred to the head-unit which then processes this information to map it to the car-specific interaction and presentation devices. Due to the diversity of interaction concepts, e. g. hard keys, touchscreens, speech interfaces or motion sensors, the representation of device capabilities has to be in an abstract manner. This ensures that the interaction possibilities can be transferred independent of specific user interface concepts or modalities. We use the term UI model for an abstract representation of a user interface which is independent of a certain implementation. Requirements for a flexible UI model are presented in the following. Based on these, existing UI representation techniques are presented and evaluated.

2.1 Requirements

A flexible UI model should fulfill a number of requirements. Van den Bergh and Coninx described some less formal requirements for the working environment [4]. The environment shall be expressive: The model shall be comprehensible and allow for complex relationships without becoming cumbersome. Tool support shall be possible since tools can ascertain that models are consistent. They also enable hiding of parts of the model during design. Other important requirements encompass internationalization. A UI shall be adaptable to different languages and cultures. According to Weld et al. units of measurement like speed, date, and time should be provided in a format matching the user's preferences [12].

A user interface shall also be consistent. User interaction shall run along the same lines each time, as Dix et al. describe [2, p. 584]. At the same time a UI has to comply to a set of ergonomic standards, like the ones set forth in ISO 9241-110. An automotive user interface furthermore has to follow certain automotive guidelines, e. g. the before mentioned ESoP guidelines.

The architecture needs to be extendable to achieve a consistent UI. It has to integrate hardware built into the car as well as additional external devices the user wants to employ within the car. Apart from devices, new functionality can also be provided by services from the web. The model itself needs to be extended at runtime to integrate the functionality of new applications.

The model also needs to be independent of the employed hardware. If the user connects their own mp3-player, the whole system shall be able to respond to commands issued via buttons as well as speech. Since the user's devices will typically not provide an own head-unit HMI or speech UI, the system will have to translate between these modalities and each device's service. Accordingly, a central control module is needed that is able to distribute respective presentation and interaction logic to the involved system components.

2.2 Model Components

Apart from requirements for the notation different aspects of the model need to be described. Our model uses a distinction between the application, tasks and the user interface. We employ the Cameleon Reference Framework which specifies four levels of abstraction [1]. The framework is illustrated in figure 1. The different abstraction levels are shown starting with *Tasks and Concepts* (T&C) at the topmost level. Tasks can be modeled using different notations which are evaluated below. Concepts are all domain objects, in our example the task *Play next mp3-track* invokes a method on a song object. The *abstract user interface* (AUI) is a modality-independent model of the UI. *Concrete user interface* (CUI) is the level at which widgets are employed and the *final user interface* (FUI) is the binary code or the UI in a markup language or hardware mapping. Since the task model concentrates on a high-level description of the user's actions, we use this location in the architecture to extend the functionality of the system.

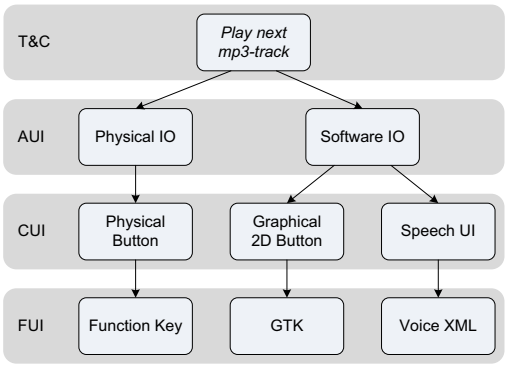

Figure 1: Different abstraction levels in the Cameleon Reference Framework starting with the task *Play next mp3-track*. Arrows denote transformations.

Transformations take place between the different models. The arrows in figure 1 indicate these transformations. The UI is generated from abstract representations by applying respective transformation rules for the HMI design, modality and capabilities of the target interaction devices. These rules have to be implemented on the car head-unit. Thus, the transformations are in control of the manufacturer while still giving them a flexible solution for integrating unknown external services. The approach is described in section 3. Any transformation entails some disadvantages, e. g. there is an additional overhead for maintaining all mapping rules between different models. In addition, to transform from one level of abstraction to another, an accurate mapping has to be found, so that no relevant information is lost in the process. The problem of determining these transformations is called the *mapping problem* and has been widely discussed by Puerta et al. [10].

2.3 Related Work

Recent work on approaches for abstract user interface representations are presented and evaluated in this section.

2.3.1 Existing approaches

Several proposals for abstract description standards exist which can be used to build upon. Concur Task Trees (CTT) [7] is one of the most widely discussed approaches to describe human machine interactions in an abstract manner.

In CTT a user interface model is specified by using several well-defined types of tasks and operators in a hierarchical top down description. Since there are no descriptions of any concrete UI elements, the task description remains platform and modality independent. A concrete UI is generated during the interpretation of a task tree either before executing the application or at runtime.

As demonstrated by Nóbrega et al. [5], the wide-spread modeling language UML [6] provides the same expressiveness as CTT. As established standard in software development, structured contents and problem descriptions can be illustrated in UML by a topology of class diagrams, activity diagrams, and statechart diagrams. Thus, an abstract and formal UI model can be specified in order to be concretized in further process steps. The description of user interactions is platform and modality independent and can be translated to different modalities.

2.3.2 Evaluation

Describing an automotive HMI on a task level with the widespread CTT-notation quickly resulted in easily understandable task trees. However, due to increased concurrency, e. g. destination entry while listening to music and accepting an incoming phone call, the tree structure quickly becomes very complex. Also, CTT does not provide a history concept which is often needed in the automotive domain for task switching. Usually, the applications open with the last active state. Thus, modeling with CTT may be useful for some interaction tasks but the approach lacks important aspects needed for more complex task descriptions.

As mentioned before UML provides the same expressiveness as CTT. Furthermore, the UML notation concepts exceed CTT, so that the emerged issues can be addressed by UML. Thus, we considered UML to be the appropriate approach to model the interactions on abstract level.

3. TRANSFORMATION-BASED UI

Since the car's HMI needs to integrate new devices during its lifetime, the preinstalled descriptions of tasks need to be updated from time to time. This can be done automatically by attaching an unknown device which provides its own task descriptions and abstract UI model or by hand through the user. The UI is then generated on demand. For this process transformations are needed between the different abstraction levels. Existing approaches are presented and evaluated in the following. Then, our transformation-based approach is outlined.

3.1 Related Work

Several approaches to generate UIs from abstract representations exist. UIML (User Interface Markup Language) is an XML-based Meta-Interface Model (MIM) [9]. Apart from the UI definition, the runtime behavior of an application can be exemplified. Interfaces for different modalities can be specified in UIML, however each modality-dependent specification is directly bound to the underlying abstract description and restricts the overall flexibility. While UIML is a widespread and advanced approach for modality and platform independent descriptions, it retains one important disadvantage. The described data and their presentations are administered in one document, thus modification at runtime is impossible. The model cannot be extended and modalities cannot be added later on.

UsiXML [11] is an XML-based User Interface Description Language (UIDL). It offers the possibility to describe a UI according to the Cameleon Reference Framework. Employing transformations between the four levels it is possible to transform the basic *Task & Concepts* (T&C) model into several adequate *Final UIs* (FUI) matching different platforms and modalities. Finding all the necessary transformations is a tedious process however, as the mapping problem illustrates.

Another interesting method of how to integrate several external services into the automotive HMI is described by Hildisch et al. [3], who propose to describe all possible abstract UI facets in a semantic ontology hierarchy. The HMI acts as an interpreter mapping the OWL-based interface description to given FUI-elements provided by the HMI system itself. This concept allows generation of UIs which are highly consistent but has the disadvantage of not being able to integrate previously unknown concepts at runtime.

3.2 General Approach

In order to build a system which can be extended at runtime we propose an approach that employs an extendable task model. The task model contains extension markers at which submodels of external devices can be attached.

We propose a transformation-based approach similar to the Cameleon Reference Framework to structure the different levels of abstraction. In contrast to the Cameleon approach, we propose to describe the tasks and concepts as well as the abstract user interface in one step using UML without the need of transformations between these levels. This alleviates the mapping problem since no transformations are necessary for the first level of the framework.

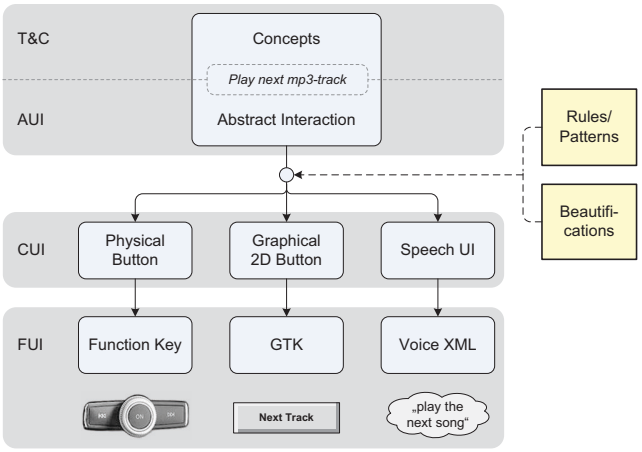

Figure 2: *Play next mp3-track* on different levels of abstraction using the adapted framework.

The adapted framework is shown in figure 2. Again the user wants to carry out the task of playing an mp3-track. The task can be modeled with activity charts and state machines, concepts are modeled as classes. Transformations take place

along the arrows, like in the original framework.

3.3 Rule-based Transformations

We aim at flexibility and extendability on the one side and controlled predictable interaction on the other side. The proposed system uses rule-based transformations in order to generate UIs. To this end it employs rules and beautifications to generate consistent UIs of a high standard.

Rules are divided into different categories. They adjust the modality of human-machine interaction to match several situational conditions i. e. provide automated switching between screen-based and voice interfaces. We propose to use patterns for well-known situations and heuristics to decide the modality otherwise. Rules also cover layout and design aspects as well as user customizations. Especially in the layout process patterns can be employed. Additionally, rules can be used to meet special personal requirements (e. g. a larger font size for the elderly).

Another important aspect are beautifications, as described by Pederavia et al. [8]. These are additional rules which are created by a designer who wants to adapt the automatically generated UI of a specific device. The application of these rules is repeated on subsequent UI generations each time the device type is connected to the car. By using beautifications designers can ensure a corporate design or adapt UIs to their preferences.

3.4 Contributions

Our proposed architecture is based on the Cameleon Reference Framework but adapts it for use with UML. This leads to less transformations and thereby alleviates the mapping problem. By employing UML the approach can leverage existing tool support and know-how, thus allowing easier participation in the design process.

The approach allows for extendability of UIs on an abstract level. External devices can be fully integrated into the system at runtime. The added task descriptions are integrated into the UI in a way to make the UI appear to come from a single source. Thus, we achieve a flexible interaction environment to support different capabilities of current and future devices.

4. CONCLUSION AND FUTURE WORK

We presented automotive-specific requirements for a flexible solution to integrate external services into the car. The possibility to completely control the integration of external devices and their user interface into the car was highlighted. Methods for abstract user interface representations were evaluated and our transformation-based approach for building user interfaces from abstract representations was motivated.

Our proposed approach can be employed for all kinds of devices. An already deployed system remains extendable independant of the car's life cycle. This flexible integration of external applications into the automotive interaction environment pushes the development of modern applications for in-car infotainment and their safe use.

The envisioned overall architecture was presented based on well-considered requirements and the evaluation of existing approaches. Further research for a detailed proof of concept is needed and scheduled for the near future.

Acknowledgements

This work is originated in the Transregional Collaborative Research Centre SFB/TRR 62 "Companion-Technology for Cognitive Technical Systems" funded by the German Research Foundation (DFG) in co-operation with Daimler AG.

5. REFERENCES

[1] G. Calvary, J. Coutaz, D. Thevenin, Q. Limbourg, L. Bouillon, and J. Vanderdonckt. A unifying reference framework for multi-target user interfaces. *Interacting with Computers*, 15:289–308, 2003.

[2] A. Dix, J. Finlay, G. D. Abowd, and R. Beale. *Human-computer interaction*. Pearson, 2004.

[3] A. Hildisch, J. Steurer, and R. Stolle. Hmi generation for plug-in services from semantic descriptions. In *ICSE Workshops SEAS '07.*, page 7, May 2007.

[4] Jan Van den Bergh and K. Coninx. Model-based design of context-sensitive interactive applications: a discussion of notations. In *Proc. TAMODIA '04*, pages 43–50, New York, NY, USA, 2004. ACM Press.

[5] L. Nóbrega, N. J. Nunes, and H. Coelho. Mapping concurtasktrees into uml 2.0. In *Proc. DSVIS'05*, pages 237–248, Berlin, 2005. Springer.

[6] Object Management Group. *UML 2.0 Superstructure Specification*, 2005.

[7] F. Paternò, C. Mancini, and S. Meniconi. Concurtasktrees: A diagrammatic notation for specifying task models. In *Proc. INTERACT*, pages 362–369, 1997.

[8] I. Pederiva, J. Vanderdonckt, S. E. na, I. Panach, and O. Pastor. The beautification process in model-driven engineering of user interfaces. In C. Baranauskas, P. Palanque, J. Abascal, and S. D. J. Barbosa, editors, *Proc. INTERACT*, volume 4662/2008 of *LNCS*, pages 411–425, 2008.

[9] C. Phanouriou. *UIML: A Device-Independent User Interface Markup Language*. PhD thesis, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, September 2000.

[10] A. Puerta and J. Eisenstein. Towards a general computational framework for model-based interface development systems. In *Proc. IUI*, pages 171–178, 1999.

[11] J. Vanderdonckt, Q. Limbourg, B. Michotte, L. Bouillon, D. Trevisan, and M. Florins. Usixml: a user interface description language for specifying multimodal user interfaces. In *Proc. Workshop on Multimodal Interaction WMI'2004*, pages 1–7, 2004.

[12] D. Weld, C. Anderson, P. Domingos, O. Etzioni, T. Lau, K. Gajos, and S. Wolfman. Automatically personalizing user interfaces. In *Proc. IJCAI-03*, page 7, 2003.