Model Based HMI Specification in an Automotive Context

Thomas Fleischmann

Elektrobit, Frauenweiherstr. 14,
91058 Erlangen, Germany
Phone: +49 (0) 9131 7701 288
Thomas.Fleischmann@elektrobit.com

Abstract. An overview of how a model based specification approach can be used in the domain of automotive human machine interface (HMI) development is presented. The common paper based specification approach is compared to a model based, tool supported process. Requirements from different stakeholders for such an approach are outlined. Intended audiences are all stakeholders involved in the creation of graphical user interfaces ranging from design, usability engineering, and prototyping to specification and final product realization.

Keywords: Model based, HMI Specification, Code generation, Domain Specific Language.

1 Introduction

The functionality contained within modern cars has increased drastically, and the car manufacturers (OEMs) face challenges in presenting the assistive systems and infotainment solutions to the driver in a usable way. Park- or lane assist and adaptive cruise control are just some of the examples. The following figure shows some more:

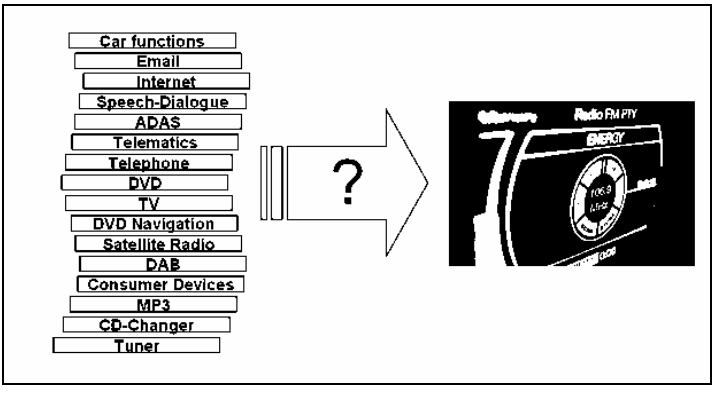

Fig. 1. Handling the complexity

M.J. Smith, G. Salvendy (Eds.): Human Interface, Part I, HCII 2007, LNCS 4557, pp. 31–39, 2007.

The number of features required forced the OEMs to shift away from the simple tuner and toward an integrated solution. The HMI became a brand identifier. Car manufacturers started to specify the look & feel of the HMI in order to get the supplier to build a custom system that fits into their brand image.

The common paper-based specification approach and its difficulties in the working process with the supplier will be described first to define the problem scope. Additional new social trends and as well technical features in the next automotive HMIs demand a change in the way these specifications are created.

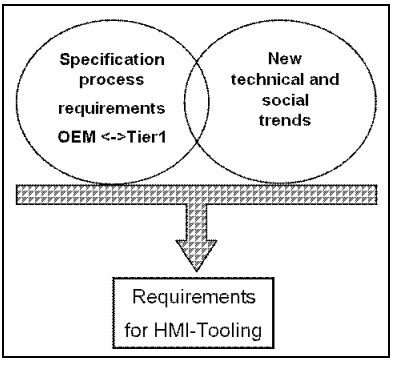

Fig. 2. Requirement input sources towards a solution

The solution for the problem lies in a domain specific and model based specification tool. Such a tool has to provide on one hand the technical features to allow specification of the latest multimodal user interface and on the other, to support the working process between OEM and Tier 1 supplier.

2 Paper Based Specification

It is still a common and widely used approach to specify an HMI on paper. Fig. 3 shows an example of what a common specification could look like. The specification typically consists of the following parts:

- Menu flow layout for graphics and speech dialog, specified in flow- or state charts
- Screen design, templates and complete screens defined in a drawing tool
- Texts for internationalization, held in a database or spreadsheet based
- Description of the software implementation details

This works well if the whole system is completely specified in a formal way and frozen at a certain deadline before handed over to the supplier. Real life experience has shown that the specification changes constantly during development and document versions become inconsistent. This typically results in a high number of very expensive change requests during these projects.

Another drawback is the missing simulation – final results can't be verified in a user clinic or by the top management before the system is a nearly finished.

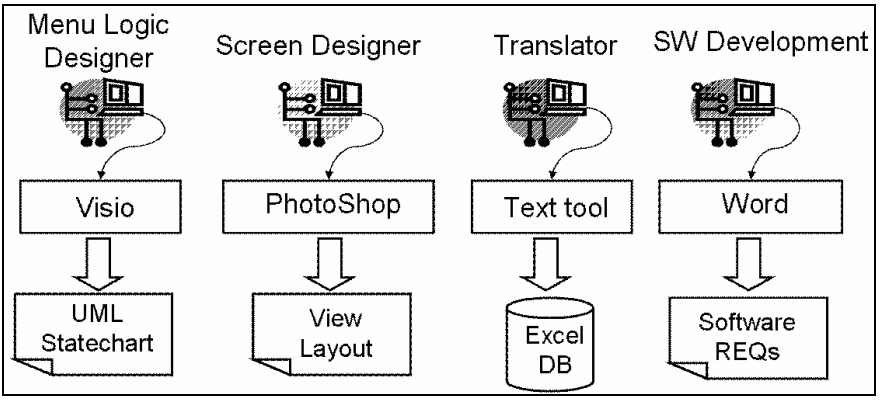

Fig. 3. Paper based specification

3 Solution - Model Based Specification

In order to create a consistent HMI system it is necessary to combine all involved specification elements into one model. Taking the speech dialog as example - the end user will notice that the graphic HMI part and the speech dialog are not synchronized or just loosely coupled if they are designed in two environments.

Fig. **4** shows the involved parties centered on the HMI model.

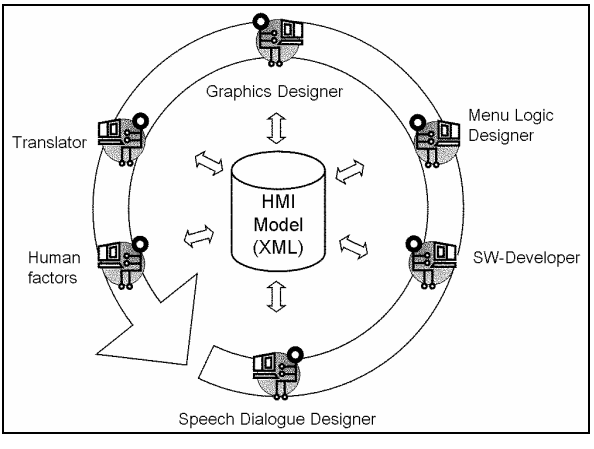

Fig. 4. Model based specification

The core parts of the model are:

- The event model to react to external triggers or events
- The data model to display and collect values from the applications
- Layout model for the screen definitions

- Menu flow model for the overall HMI structure
- Speech dialog model to enable the specification of multimodal HMIs

Event Model
The event model describes all events which affect the behaviour of the HMI. This comprises everything from a pressed button to an incoming phone call to every possible error that shall have a result in the HMI. Each event is identified by its name and can have a variable set of parameters to describe the event in more detail. All events are known to every part of the specification. The events may go into the HMI but the HMI can also send events to the outside world to trigger actions there.

This abstract view of the communication flow allows specifying different event sources and targets in one model without knowing every detail of the future system. A formal event model is also the first precondition for developing a simulation based on the specification.

Data Model
Specifying an HMI involves defining all data that the HMI requires from the system. If a value is displayed on the screen there has to be a provider for this data – the details of which software component this actually is does not matter while specifying the HMI. All data are kept inside the data pool component which is visible in the whole HMI and notifies interested components of data updates. To put it another way, this data abstraction follows the simple rule:

"I know the values exist;
I recognize when they change;
I don't care where the data comes from."

Following this rule, the data pool decouples the HMI from the rest of the system. Dynamic data which are subject to changes, like a tuner station list, must be distinguished from static data, like the coordinates of a title bar or the color model. All values are defined at one place and can be referenced throughout the whole system. This still makes changing a picture set for 500 screens an easier task.

The abstraction from the implementation specific data source is also one precondition to allow simulation during the specification phase without having a real tuner or telephone application available. The following figure shows how the data pool architecture separates the HMI and the application.

Layout Model
The layout model contains a formal description of each screen in the HMI. Each screen consists of multiple graphical control units called Widgets. The appearance of each Widget can be defined by a Widget-specific set of properties. Widgets should follow some basic rules. The first is that they should comply with the model-view-controller paradigm. This is nothing new to GUI developers. It simply means that graphical representation, behaviour and data must be kept separate to ensure flexibility. (e.g. A button can look completely different just by replacing its rendering component or work in a touch screen simply by replacing its controller part).

Secondly, the Widgets have to support inheritance to allow easy extension of the functionality of existing widgets.

The Widgets link different parts of the HMI model together. The most obvious link is a reference from for example, a label widget to a MP3-Text defined in the data pool. This slot would be provided by the MP3 application later in the target software. The Widget controller parts have to be able to act as event sinks or event sources into the system during a user's interaction with the Widget.

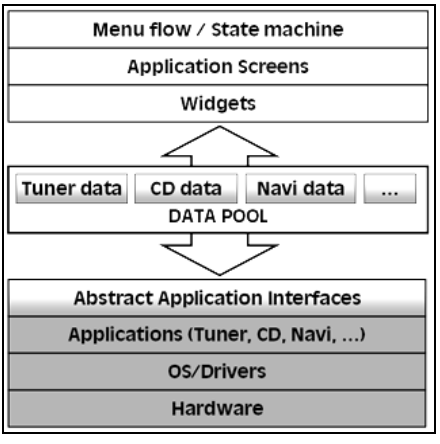

Fig. 5. HMI and applications separated by data pool

Menu Flow Model

The user's navigation through the whole HMI menu is specified in a menu flow model. A detailed description defines how each screen of the layout model can lead to the next screen. Illustrated UML state charts are used to define the menu logic. Each state without further child states can be linked to a screen from the layout model. A screen miniature is displayed in the diagram to allow a non expert in UML to easily follow the specified menu flow. The transitions between the states are triggered by events from the event model. Features provided by the UML state machine model like the history mechanism, conditional transitions or actions trigger when entering or leaving a state are supported as well.

Speech Dialog Model

The extension of the model with text-to-speech (TTS) prompts definitions and grammar rules will enable a full multimodal HMI specification and, if brought into a tool environment, an on-the fly simulation.

Currently many speech dialogs are created in independent tools decoupled from the graphical HMI model. Combining this in one model is a great step towards an integrated HMI specification. The resulting speech dialog model can be exported for different recognizer engines running in the device.

4 XML Export of a Domain Specific Language

Specifying an HMI model using these concepts will result in a domain specific language (DSL) which is very complex. Therefore a set of intelligent editors and tools is required to manipulate such a model. In order to allow others to setup automated processes using such a model, an export into a well documented data format is necessary. XML schemas fit this need very well. The figure below shows a part of the schema definition for a state definition from the menu flow model. Finally, the tool allows an export of the specified model into an XML file which is exactly compliant to the schema and therefore enables easy integration of the model data into a third party development process.

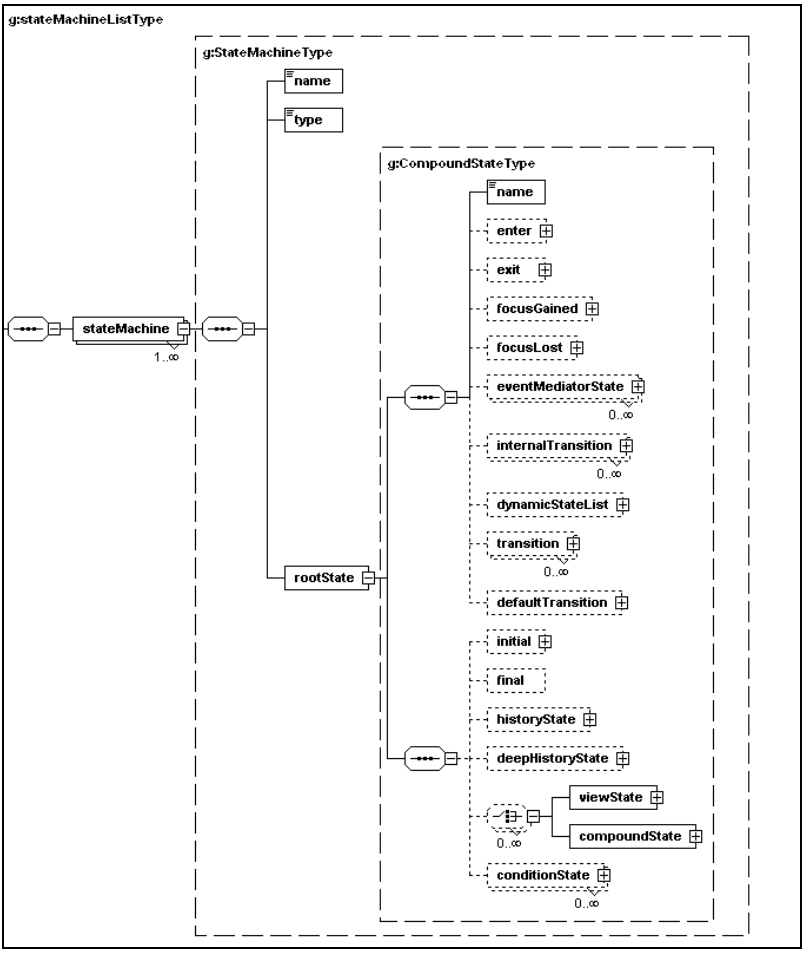

Fig. 6. XML Schema for a state definition

5 Tool Requirements

Some requirements for a tool, like simulation and automatic consistency checks, can already be derived from the above mentioned points. Some others are briefly outlined in the following section:

Rapid Prototyping
To try and test new HMI concepts, a rapid prototyping environment has to offer interfaces to integrate with new hardware and allow for a fast design of new HMIs. The reuse of elements from this prototyping into the actual specification is required to prevent "reinventing the wheel" during the final implementation phase.

Formal Specification
A model based tool must offer a way to create a formal specification – therefore it requires a formal element for every type of problem domain object. Multi-user access to this specification is a must in bigger projects.

Version Management
The specification will be given to the supplier at a certain point in time. If multiple parties start to change the model, an intelligent HMI tool has to provide version management, change logs and merge capabilities to support this as
Fig. **7** illustrates. The simulation model *V1.0* is handed over to a supplier who takes care of the realization. The supplier will extend the model into an implementation model *V1.2* which is used for code generation. The extensions made to create *V1.2* need to be merged with the next simulation model *V2.0*. Even though XML is used to store the model information, a simple text compare would not be of any help. In order to achieve a comparison on a logical level and to allow a conflict free merge a tool support for model comparison and merging is required.

Automatic Code Generation
If the whole system is specified in a formal model and this model can be exported in a machine readable format; the specification can be directly used to automatically generate the code for the embedded target system out of the specification. For speech dialogs the tool would directly provide the grammar, commands and prompts. For the graphic HMI it would provide all screen definitions and an executable state machine.

Test Support
Automated HMI testing as well as export functions for manual HMI test instructions and validation of specified behavior could be easily extracted from a machine readable HMI model and used by both OEM and Tier 1 developers. This covers the whole lifecycle from early usability testing to playback of regression tests in the maintenance phase of a project.

Independence
The OEMs have the need to stay independent from the supplier to a certain extent. Therefore Tier 1 suppliers can not create an HMI tool and offer it to an OEM. Only an

independent vendor can provide such a tool which provides open interfaces and support for standards to both the OEMs and Tier 1 developers.

Future Requirements

Future systems will include full graphics based cluster instruments, head-up displays and monitors for rear seat entertainment – all of this probably combined with speech dialogue systems with natural speech input. 3D graphics capabilities will have to be supported by an HMI tool as well as new input concepts like touch screens which are especially useful for Asian character input and handwriting recognition.

HMIs are getting increasingly complex while the society is getting older and the number of elderly people driving will increase. Future automotive HMIs will have to pay respect to this fact and HMIs will have to adapt to this user group.

Another challenge for the whole automotive industry lies in the seamless integration of various portable devices or portable navigation solutions. Initial attempts have been made to connect iPods and USB memory sticks but the different lifetimes of the car and the consumer industry will demand a flexible software architecture in the automobile and the tools to handle it.

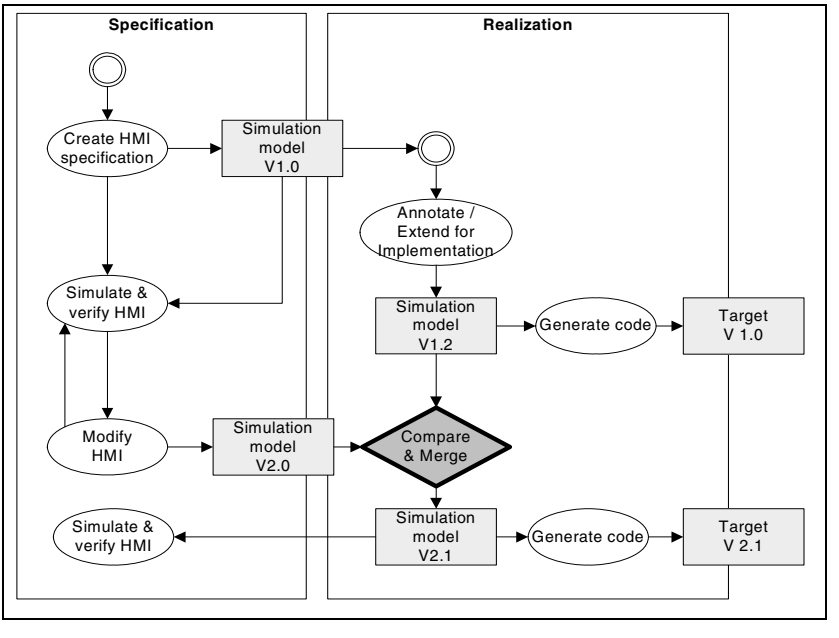

Fig. 7. Compare and merge of HMI versions

6 Status of the Tool

The basic HMI of the current Audi A6 model was successfully built by Elektrobit with a tool which implemented the techniques for model based HMI development

described in this paper. Production start for the A6 was in early 2004. The automatic code generation supported by the tool was widely used during this project. The Audi HMI comprises roughly 500 screens to control tuner, CD, navigation and car setup functions.

Since 2005, Elektrobit has been providing the HMI tool **tresos® GUIDE** and its associated runtime software environment to the automotive market.

Reference

1. Dr. Holve, R.: A model-based Approach towards Human-Machine-Interfaces. ITS Telecommunication 2005, Brest, FRANCE (June 27-29, 2005)