Standardizing Model-Based IVI Development in the German Automotive Industry

Steffen Hess, Anne Gross, Andreas Maier
Fraunhofer IESE
67663 Kaiserslautern, Germany
{steffen.hess, anne.gross, andreas.maier}@iese.fraunhofer.de

ABSTRACT

Based on the analysis of existing HMI development processes in the automotive domain, a reference process for software engineering has been developed. This process was used to develop a domain data model and a model-based specification language in order to establish a common exchange format based on consistent domain knowledge. This approach provides a common ground for collaboration to OEMs, suppliers, and software tool developers. It also facilitates traceability from requirements and conceptual elements to concrete, specified elements using the model-based specification language.

Categories and Subject Descriptors

D.2.1 [Software Engineering]: Requirements/Specifications

General Terms

Documentation, Design, Standardization, Languages

Keywords

Domain Data Model, Automotive, Specification, HMI, Model-based Language, User Interface Design, Interaction Design

1. INTRODUCTION

Car infotainment systems are an important part of modern cars, providing radio functionality, media playback, navigation, and even telephony. They are expected as part of the standard instrumentation in high-priced cars, but are increasingly finding their way into the medium-priced and even low-priced market segments as well.

These infotainment systems need to conform to safety regulations, e.g., they should not distract the driver from his main task of driving the vehicle. They also need to offer high usability, since drivers get no special training on using the systems.

The software part directly interacting with the user of an infotainment system is called an HMI (human-machine interface). HMIs are difficult to specify, implement, and test, since their static and dynamic properties are very complex. A modern HMI consists of around 2000 graphical views, with behavior being described with more than 2500 system states.

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, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright (c) 2012 ACM 978-1-4503-1751-1/12/10 ... $15.00

Currently, a new HMI is specified using text, graphics, and system behavior descriptions (e.g., state charts), which can lead to documents with up to four-digit page numbers. This specification is distributed to implementation and testing teams, which use it as a basis for creating software and test cases. If ambiguities or contradictions are found in the specification, which happens regularly, they are reported to the specification team, which incorporates solutions or clarifications in the next version of the specification. These specifications change a lot over the course of the project, caused by error reports but also due to new functionalities being added or removed, sometimes on a very large scale. Therefore, other teams must find the differences between specification versions. The testing team, in particular, needs to know which version of the software conforms to which version of the specification. Currently this process results in long development times of about three to five years.

In the future, new innovations and user requirements will lead to even more functionality in automotive HMIs. This is partly fueled by innovations in smartphones and tablets (like apps, speech recognition, and mobile Internet access), which are expected by the customer to be translated to the automotive domain. Also, the introduction of reliable speech input and output systems might significantly reduce the distractions currently caused, e.g., by entering a destination into the navigation system while driving. Finally, customers will expect the high-resolution graphics and animations currently found in consumer electronics to appear also in their car HMIs, which further complicates the development of infotainment systems.

The automotive industry therefore faces the challenge of delivering more functionality in less time. To reach this goal, the research project “automotiveHMI” [2] was established. Its aim is to make specifications as the central drivers of the development process more structured, consistent, and easier to handle. This will be achieved by the creation of a specification language that allows traceability throughout the whole development process from initial concepts via detailed specifications to implementations and tests.

This language will also allow describing HMI properties in a more formal way. The benefits of a more formal specification are the automatic derivation of test cases, a better understanding of the requirements, and a decrease in communication caused by misunderstandings and ambiguous information. In this paper, we describe some of the results of automotiveHMI achieved so far.
The project is structured as follows (see Figure 1):

In work package 1, an analysis of the processes, roles, and tools used by the different companies is performed. Based on that, a reference process is created (see chapter 2).

In work package 2, the information gathered in the analysis is used to create a domain data model of automotive HMI development (see chapter 3). This model is used to create a common understanding of the concepts and relations of the domain across all partners (e.g., OEM, 1st tier supplier, 2nd tier supplier, tool provider).

In work package 3, a specification language is derived from the domain data model (see chapter 4). The language aims to capture the essential information required by each role to perform its work. It therefore contains a subset of the domain model, but needs to conform to requirements gathered during the analysis.

Chapter 5 describes evaluation activities prepared and conducted in the project to iteratively improve the domain data model and the specification language throughout the project.

2. REFERENCE PROCESS

In this chapter, we introduce a reference process for HMI development. This process has been derived by eliciting and analyzing current HMI development processes within the scope of work package 1 (see Figure 1). In total, we conducted nine elicitation workshops at the sites of different partners, comprising both major OEMs and suppliers in Germany.

The reference process that will be presented in the following should not be considered as “the only” process that is executed at all partners’ sites. It rather combines all the best practices we identified by analyzing all individual processes on a higher level of abstraction without communicating too much detail about individual processes and activities. However, it is essential that all partners can identify themselves within this process, as we further analyzed this process in terms of relevant artifacts that are typically produced within the particular phases for use as a baseline for the domain data model presented in chapter 3.

Figure 2 provides an overview of the different phases of the reference process we identified by analyzing all individual processes on a higher level of abstraction without communicating too much detail about individual processes and activities. However, it is essential that all partners can identify themselves within this process, as we further analyzed this process in terms of relevant artifacts that are typically produced within the particular phases for use as a baseline for the domain data model presented in chapter 3.

Figure 2. Overview Reference Process

2.1 Research Phase

This phase is dedicated to prototypically developing and evaluating new conceptual ideas and technologies before they actually enter the HMI development process in serial production. New ideas are derived based on current market trends, customer demands, etc. These ideas are then realized in the form of software, hardware, or design prototypes and finally evaluated in terms of their qualification and suitability for serial production. The decision about whether a certain concept has good potential for entering serial production is typically made by the management. That is, promising concepts will then enter the subsequent specification phase where detailed requirements will be derived and specified as input for further development activities.

2.2 Specification Phase

The main aim of this phase is to derive and specify detailed HMI requirements that will be used as a baseline for development (see chapter 2.3) and testing activities (see chapters 2.4 and 2.5).

Input for specification activities includes, for instance, evaluated concepts of the research phase (see chapter 2.1) and requirements derived from elicitation activities like market studies or user studies. Furthermore, there also exist some internal guidelines such as style guides that basically define best practices and lessons learned from previous projects and specification templates as well.

This specification phase is also supplemented with analysis and review activities to identify any inconsistencies between the requirements, determine the feasibility of the requirements, etc. Typically, requirements are refined in an iterative manner, e.g., based on a feature roll-out plan defining implementation milestones that were agreed upon by an OEM and the supplier. Furthermore, change management processes are defined that have to be considered if requirements are changed after a certain point in time (e.g., feature freeze).

In fact, the specification phase is a challenging phase today due to the interaction between the OEM as the customer that delivers the requirements and the supplier who is responsible for realizing the requirements. It can be observed that the responsibilities related to the specification differ between OEM and supplier. In some cases, the OEM provides requirements specifications to the supplier that are already very detailed, whereas in other cases, the supplier is only given high-level specifications and has to detail the requirements specification on his own. This typically requires some communication effort, which varies according to the collaboration experience and the familiarity between the two parties gained from previous projects. However, for the purpose of the reference process, we do not explicitly define who is actually responsible for the specification – we rather focus on typical artifacts that are produced during this specification phase.
as this information is essential for defining the domain data model (see chapter 3) and the model-based language (see chapter 4) as well. These artifacts comprise, for instance, sequence diagrams, widget catalogs, use cases, animations, etc.

2.3 Development Phase

During the development phase, specified HMI requirements (see Section 2.2) are realized in the form of hardware and software. During development, the suppliers often follow V-model-like processes that are aligned to the feature roll-out plan (and thus work in an iterative manner) [1] [3].

Finally, all development activities are typically supplemented with test activities (see Section 2.4). Based on a defined time and milestone plan (e.g., a feature roll-out plan), a new and updated version of the developed HMI is delivered to the OEM by the supplier. In case of upcoming changes in the requirements detected during the development phase, the requirements specification has to be updated based on a predefined change management process (see Section 2.2).

2.4 Test Phase (Supplier)

Before a new HMI version is actually delivered to the OEM, several test activities take place at the supplier’s site. These tests typically include module / component tests, integration tests, as well as system tests. All test activities at the supplier’s site are embedded in a predefined test process comprising activities related to creating test plans, test designs, test cases, test data, etc. Furthermore, all test results are typically documented, e.g., in the form of test reports and test protocols.

2.5 Test Phase (OEM)

Once a new version of an HMI has been delivered by the supplier, test activities start also at the OEM site. These activities mainly aim at testing the correct behavior of the HMI as a whole system against its specification (see chapter 2.2). This includes correct design and structure of the HMI, functional and nonfunctional behavior, etc.

All test activities are systematically planned and monitored in this case as well. As mentioned above, the HMI is typically delivered in different versions, which requires all test activities to be executed with each delivery. Therefore, OEMs increasingly strive to automate test activities in order to save time and effort.

In addition to these system tests, further quality assurance activities take place also at the OEM site throughout the whole development process. These activities include, for instance, the derivation of test cases based on requirements specifications once these have reached a stable state. Besides preparing the actual test activities, such measurements also allow identifying inconsistencies early in the development process, rather than at a later stage. In addition, test plans and test cases are reviewed often in order to assure quality criteria such as correctness, completeness, etc. As HMI development takes place iteratively, changes between different versions are also captured in the form of delta models, which are then used to update and adjust affected elements in the test plan, test cases, etc.

Finally, in parallel to the quality assurance activities mentioned above, both risk management activities and version and configuration management activities take place during the whole HMI development lifecycle.

2.6 Acceptance Test

Finally, the developed HMI has to pass an acceptance test, where the HMI module is presented to company directors and management (e.g., project managers), who basically have to approve and accept the final HMI. Any change requests resulting from this acceptance test is communicated to the teams involved in the specification phase so that appropriate development activities can be initiated.

2.7 Maintenance

Once the HMI has been integrated into the vehicle and thus brought into the market, the maintenance phase starts. During that time (3-5 years), software updates may take place (usually at an automotive service center) that fix possible software problems, update data like navigation maps, etc.

3. DOMAIN DATA MODEL

Based on the information gathered during the development of the reference process, the domain data model of the automotive HMI software development was assembled. This model is used to get a common understanding of the concepts and relations between those concepts across all possible development partners. The domain data model consists of six views (see Figure 3), which allow covering the whole HMI development process.

This paper focuses on the detailed specification of an HMI that describes the major conceptual elements of a specification. This view is illustrated in three sub-views: the detailed specification overview, which shows a coarse overview of the detailed specification; the HMI view, which shows the conceptual elements that compose the HMI; and the HMI-Interaction view, which shows the conceptual elements that are directly related to the interaction between a human and the system. One of the main benefits is to obtain traceability from functional specification elements down to single user interface widgets. The following subchapters only show artifacts and their relationships, while attributes were excluded from the views to facilitate readability.

All views of the domain data model are divided into three main categories: conventional and display elements, which are related to the interaction between a human and the system; functional and non-functional properties of the HMI; and the interaction between the HMI and the system.
3.1 Detailed Specification Overview
The detailed specification overview (see Figure 4) shows the basic conceptual elements of an automotive HMI specification.

During HMI development, the Detailed Specification represents information about requirements that are necessary for the current system development in terms of serial production. It consists of Functional Specification Elements that describe the provided system functions at different development maturity stages. In contrast to non-functional requirements, these requirements do not express attributes the product should have, but rather concrete features the product should have.

A Usage Scenario is a Functional Specification Element, which might have different concrete notations (e.g., Use Case, User Scenario, etc.) but is always performed on exactly one product variant of a concrete HMI. It usually consists of Interactions representing actual human-machine interactions, but may also contain System Actions that do not require any human involvement (e.g., system notifications that are only shown temporarily and do not require any user action).

![Figure 4. Detailed Specification Overview](image)

3.2 HMI View
The HMI View (see Figure 5) shows how the HMI is assembled and which conceptual elements of the HMI are relevant for software development and especially for user interaction with the software. Usually the HMI consists of several Devices, which can either act as Input Interface (e.g., multi-function button), Output Interface (e.g., screen), or both Input and Output Interface (e.g., touch screen). The distinction between Input and Output Interface is relevant because especially in the automotive domain, the input and output of an interaction are often not related to the same device (e.g., controls on the steering wheel that operate an output device on the dashboard), and this needs to be considered explicitly when designing the interaction.

When a Device acts as Input Interface, it provides Input Modalities (e.g., speech, touch); if it acts as Output Interface, it provides Output Modalities (e.g., audio, vibration, video). Both types of Interfaces potentially consist of Software Elements (e.g., buttons, lists, virtual keyboards) and Hardware Elements (e.g., knobs, switches), which can also be combined. At least one Software or Hardware Element is necessary for an interface to exist.

![Figure 5. HMI View](image)

3.3 HMI Interaction View
Figure 6 illustrates the HMI Interaction View. This view basically comprises all elements that are relevant for specifying concrete interactions with the HMI and thus supports the complete HMI specification.

In general, an Interaction consists of (1) one or more Elementary Interactions, which represent exactly one step during the Interaction and (2) exactly one Interaction Flow, which specifies interaction possibilities and user flows. One Elementary Interaction comprises exactly one Human Action and one System Action, whereby the conceptual model does not explicitly prescribe whether an interaction is triggered by a Human Action or a System Action. However, a Human Action is always performed on a Device acting as Input Interface and ultimately defines the Input Modality used, which is selected from the available modalities provided by the interface. Similarly, the System Action is always performed on a Device acting as Output Interface and ultimately defines the Output Modality used.

To structure the System Action in more detail, it consists of Input Feedback (e.g., button changes color on tap), which gives the user feedback about the accomplishment of a Human Action. From a conceptual standpoint it is not necessary to have Input Feedback, but we strongly recommend addressing it explicitly in order to increase the usability of the HMI. Furthermore, Application Feedback and System Function complete the System Action, while Application Feedback represents those System Functions that represent the actual system response (e.g., displaying the navigation map and starting to navigate) to the previous Human Action. This part of the System Action is always represented in a Screen. A Screen explicitly includes combinations of all possible kinds of Widgets, consisting again of Widget Primitives. For instance, a widget that only consists of auditory output can be seen as “one screen”; an animated 3-D model of a car including both auditory output and a menu widget at the same time can also be seen as “one screen”.

The Interaction Flow is conceptually assembled using two Screens and one Screen Transition.
3.4 Traceability Example

This example shows the traceability of the HMI development process with a focus on the concrete interaction specification shown in Figure 6. The HMI, which is specified in this example, is a built-in navigation system. We specify a rudimentary navigation system to illustrate the data given in Figure 6, but not to show what a navigation system should look like in general.

When a navigation system is used, the user generally knows the main coarse steps needed to achieve the goal of getting navigation to a particular destination: he has to transmit the information that he wants a route to be calculated, he has to enter the relevant data, and he has to start the calculation of the route and thus the navigation process. These three steps are the Interactions he has to make with the HMI. Up to this point, no concrete interaction has been performed. The concrete ways of interacting with the navigation system are given and performed via Elementary Interactions. In our example, the user has to push a button to transmit the information that he wants the system to calculate a route. Afterwards, the system shows a form to enter the destination data. Pushing the button is the Human Action, showing the destination input form is the System Action.

With these two actions, exactly one Human Action and one System Action, the first Elementary Interaction is fulfilled, since the user perceives the achievement of his first Interaction. Whereas the Interaction is the coarse functional plan of interchanging information, the Elementary Interaction is the concretely designed implementation of the interaction, according to the given devices, abilities, and preferences. The Human Action must be performed in a manner that is given by the Input Interface of the navigation system; in our example, the Input Interface is a dashboard with hardware buttons. Each of these hardware buttons is specified as a Hardware Element.

This dashboard also forces the input modality: The user has to press the button, so the input modality is “Touch”. To confirm the Human Action, Input Feedback is given. In our example, the button, that the user has just pressed flashes.

The Output Interface of our example navigation system is a simple multi-color display. By displaying the destination input screen, the system gives Application Feedback to the user. With this Application Feedback, the user knows that the system understood what he wanted it to do and that it supports him in achieving his goal of letting the system calculate a route and navigate him to his destination. Since an Application Feedback does not appear from nowhere, there must always exist an internal System Function that fulfills the action (showing the destination input screen), prepares and finally shows the Application Feedback. But these internal system functions are beyond the scope of this paper and even beyond the scope of a concrete interaction with an HMI, since the user does not perceive these System Functions.

The Screen Transition shows the first screen disappearing to the left and the destination input screen appearing from the right side of the display in our example. This Screen Transition is the same during all the Interactions when an Interaction is accomplished. The destination input screen consists of several Widgets: First of all, the screen itself is a Widget. It specifies the main look of the screen (background color, background image, fonts, font colors, etc.). On the Screen, single Widgets for entering the first relevant data, namely city and postal code, are shown. These Widgets consist of the Widget Primitives “Input Field” and “Text”, which labels the input fields. These Widgets are the Software Elements of the Output Interface. If a touch screen was to be used, Widgets could be used as Software Elements of the Input Interface as well.
With a multi-function button, the user enters the destination data. After that, he presses the “OK” button. Since not all data are entered, the interaction is not fulfilled. Nevertheless, an Elementary Interaction is made and Application Feedback shows the next relevant data, namely street and house number. Here, the Screen Transition specifies that the Screen moves upwards and the new data input fields appear from the bottom. That is, the user knows when an Interaction and an Elementary Interaction, respectively, is made just by looking at the Screen Transitions. After he has entered the house number, all relevant data are transmitted, the Interaction is fulfilled, and the user gets to the next Screen. Once again, the Screen disappears to the left and the new Screen appears from the right. On the new Screen, the user gets an overview of the data he has entered and is requested to confirm these data. With another press on the “OK” button, the calculation of the route and the navigation process begins. With the coarse specification of Interactions on the functional level, the altering Human Actions and System Actions in combination with corresponding Screens and Screen Transitions specify the Interaction Flow.

With all of these data, we are able to trace concrete interactions with an HMI from Functional Specification Elements to Widget Primitives.

4. MODEL-BASED LANGUAGE

4.1 Introduction

The specification language allows creating models that contain information relevant for the development of automotive HMIs (visual properties, behavior, and communication with the middleware)[4].

The language covers only parts of the domain model, since not all domain model entities can or need to be specified.

The language aims to solve several problems currently found in HMI development, like different versions of the specification, long turnaround times for questions from the developers, or the need for the testers to create their own formal understanding of the specification.

There are additional requirements for the language: for example, it should be compatible with the processes of different OEMs and suppliers.

4.2 Information stored in the model

The specification language forms the central pillar in the automotiveHMI project, since it provides the basis for model-based testing and specifies the interactions with the middleware. It combines information from the data model with requirements gathered during the analysis to form an exchange format that allows importing and exporting specifications into and from different tools and holds this information on different levels of formalization.

The specification language embodies a subset of the information found in the domain model. This subset, which contains all parts of the information that are actually needed for the development of the HMI, can be divided into three categories:

Appearance: This information includes the look of graphical representations as well as audio or haptic output. Audio output can be specified directly as a sound file or as configuration for speech output; haptic output is normally performed outside the HMI and is therefore just triggered by an event that the HMI sends. Thus the graphical output is currently the most important and complex part. Since modern HMIs are mostly dialog-based user interfaces, the graphical output is described by means of distinct ‘screens’, which are shown depending on the HMI’s internal state. Each screen can be decomposed into a number of widgets, which in turn may be either primitive (implemented) widgets or composed widgets, which can be decomposed further. The look of a widget is specified with images and potentially textual descriptions. Additionally, the widget structure and the positions and sizes of widgets are part of the specification.

Behavior: What does the HMI do when it receives an event under certain conditions? This information is commonly described with a state machine, where each state describes a state of the HMI and each transition a possible state change when the HMI receives the event related to this transition. Additionally, each transition can have actions that are performed when executing the transition (e.g., signaling the CD player to load information about the tracks when changing to the “media” context) and guard conditions, which can prevent a transition being executed (e.g., no changing to the “navigation” context if no navigation hardware is installed).

Application connections: HMIs communicate with an underlying ‘application layer’. This layer contains software components that embody specific services (e.g., a navigation component that contains path-finding algorithms, a geographical database, and rendering parts for the map) or act as drivers for hardware devices (e.g., buttons, screens). From the perspective of the HMI, it is important to know which functions of the application components can be invoked, which events can be received and sent, and which information is provided by the applications. All of this information together forms the interface to the rest of the infotainment system and allows implementing the HMI in parallel with the other components.

4.3 Varying degrees of formality

The central idea behind the language is to offer the freedom to specify an HMI completely informally (with text), completely formally, or something in the middle. Each OEM and supplier can therefore regulate the amount of formality in the specification.

This by itself is too generic for an exchange format, because there can be virtually anything in a model. OEMs and suppliers (or different teams) can therefore negotiate a level of formality for a certain project or even for a delivery. This level can then be expressed as a set of predefined constraints on the language. These constraints can be used to check if a model satisfies the negotiated level of formality when the model is delivered. Internally, it can also be used to track the progress of the specification. Since even very formal specifications start as an unordered stack of ideas, sketches, and notes, they can be initially put into the specification and can then be iterated until the desired level of formality is reached.

The required formality of a model depends on its usage scenarios. If the model is printed out and used as a reference for the implementers, a low level of formality is sufficient. If, on the other hand, the model serves as input to a code or test-case generator, all information needed by these generators must be completely formal.

A company could decide to make the specification completely formal, regardless of its usage scenarios. However, this approach has two disadvantages:

1. A formal specification is very detailed and exact. It therefore requires a lot more effort than an informal specification.
2. The final HMI represents the combined work and knowledge of all people involved in a project. A more formal specification requires more knowledge of diverse areas like graphical design, interaction design, middleware communication, application interfaces, hardware-dependent loading times, etc. Therefore, a formal specification requires either people with knowledge in several areas, more people with different backgrounds, or guesses by the specification writers. In real projects, the most frequently encountered option is the third one, leading to costly errors in the specification.

The most economical decision would therefore be to formalize only those parts where there is a benefit in terms of time or effort. The language is currently being developed as an XML schema to simplify import and export with different tools. The concept itself can also be represented as a database layout, a memory model, etc. Whether project teams exchange XML files or access a common database depends on the company-specific software setup and on the applicable regulations.

The language also supports model operations like splitting and merging, relations between parts of the specification to allow traceability, and a versioning concept to document all changes.

4.4 Artifacts as building blocks

The specification language allows storing parts of the specification as distinct parts called Artifacts.

Each Artifact has an Artifact Type, which describes in which form the information is stored.

Artifacts can be related to one another by using named Relations. This allows, e.g., to connect a textual requirement to a screen that implements it.

There are Artifact Types that can be used to store more informal information, e.g., text, pictures, or links to external files. These can be used for early specification phases, information for other teams, annotations and notes. Artifacts of this type are used to communicate between the people involved with the project.

There are also formal Artifact Types used for state machine descriptions or hierarchies of graphical widgets. Creating Artifacts of this type requires more knowledge about the respective part of the HMI, since they contain exact information that will be used by generators or prototype simulators.

4.5 A small example

An OEM and a supplier decide to use the specification language for their next project. The testers of the OEM and the implementers of the supplier decide that they need a model with formal parts for code generation and testing. However, the specification writers of the OEM do not possess the necessary knowledge that is needed to specify the middleware communication formally. It is therefore decided that the OEM will create a less formal specification that will be extended by the supplier. The specification writers define the features, the look, and the interaction behavior. The behavior is partly formal since it was used by the interaction designers to generate a prototype for evaluation. Every few weeks, the current version is sent to the supplier.

The supplier extends the specification by adding widget hierarchies derived from the informal description of the OEM. The appropriate specification parts are linked to the corresponding hierarchies. The behavior is also modeled formally based on the text. The prototype helps by allowing to experience the concept. Specified widgets are implemented using standard development environments. A code generator combines the specification (which by now is more a 'development model') with graphics and widgets to generate an implementation. At different milestones, early versions of the implementation and the development model are sent to the OEM.

The testers on the OEM side apply a constraint checker (with project-specific constraints) to the model to verify that the specified screens are connected to the state machine describing the dynamic behavior, that all widgets required by the widget hierarchies are present, etc. Then they enhance the model with test data and use it to generate tests for the implementation. A test execution tool uses the state machine of the model to navigate to each screen and tests for the right widget positioning and look using machine vision and the widget hierarchies.

The model operations allow both developers and testers to seamlessly merge new versions into their current model. A constraint check after the merge makes it easy to see where new requirements were added or existing ones were modified. This automates a step that is currently performed manually and may take weeks.

5. EVALUATION ACTIVITIES

As both the domain data model (introduced in chapter 3) and the model-based language (introduced in chapter 4) are very important contributions to be achieved within the automotiveHMI project, their quality has to be assured throughout the whole project. For this purpose, evaluation activities were prepared and conducted. These activities included an expert workshop as well as the collection of quantifiable data related to important quality characteristics such as correctness, completeness, and understandability by means of a questionnaire.

5.1 Expert Workshop

In order to discuss the current version of the model-based language and the underlying domain data model, an expert workshop was conducted. In preparation for the workshop, the experts (i.e., representatives of all major OEMs and suppliers involved in the automotiveHMI project) had to familiarize themselves with the current versions as documented in the project deliverables and prepare a short presentation summarizing their feedback.

During this workshop, the feedback was collected and afterwards discussed in working groups, which also elaborated suitable ideas for improvement. The major issues that arose during this workshop have already been addressed in the versions presented in this paper. These issues include, among others, inappropriate level of detail, incorrect relationships between elements within the data model, and inconsistencies between the elements and attributes.

5.2 Data Collection

Besides the qualitative data collected during the expert workshop, we also designed a questionnaire in order to evaluate certain quality characteristics of the domain data model by means of quantifiable data. The purpose of this questionnaire is to review and evaluate the current status of the domain data model in regular intervals throughout the whole project (e.g., every 6 months). That is, besides collecting quantifiable data, the questionnaire also supports the experts in reviewing updated versions, as it can be considered as a review guideline focusing on different quality aspects. The questionnaire itself is composed of
five parts. The first part aims to collect some general data related to the particular viewpoint and typical tasks / responsibilities that the partners have within the project and from which the evaluation will be performed. For each of these viewpoints, a different section has been designed within the questionnaire corresponding to part 2 (“specification view”), part 3 (“development view”), and part 4 (“quality assurance view”). Finally, part 5 (“general view”) is used to evaluate the domain data model from an overall perspective. In order to collect quantifiable data, several statements have been derived for each part, with each statement referring to specific quality characteristics, such as correctness, consistency, conformity, applicability, etc. During evaluation, the experts have to rate these statements on a scale from 1 (“strongly disagree”) to 5 (“strongly agree”). In addition, the experts can also include comments to explain their respective rating.

Figure 7 illustrates an extract from the questionnaire within the “specification view”.

![Figure 7: Statement related to the "Completeness" of the specification view](image)

Currently, the first evaluation is still running so that we cannot provide any valid results yet. However, we consider such an instrument as very useful for systematically collecting feedback in such research projects. We also aim to collect further feedback at later stages in this project to allow us to assess and evaluate whether we were able to improve our results.

6. CONCLUSION AND FUTURE WORK

In this paper, we presented a model-driven user interface design approach that supports HMI developers in the automotive industry. In particular, we focused on car infotainment systems. These systems need to conform to safety regulations and have to provide high usability. With the domain data model (see chapter 3) and the specification language (see chapter 4), we allow HMI designers to specify all relevant parts of an HMI. Moreover, we support the implementation and testing of the specifications and the whole system. By applying this approach, the automotive industry is able to deliver more functionality within less time, even innovative aspects of an HMI. Specifications of an HMI can be traced through the whole development process, from initial concepts via detailed specifications to implementations and tests. The specification itself is done in a more formal way than without this approach. The formalization leads to the automatic derivation of test cases, a better understanding of the requirements, and seamless communication between the stakeholders. Misunderstandings and ambiguous information are minimized.

The approach emerged from elicited and analyzed information regarding existing HMI development processes in the automotive industry (see chapter 2). We used this information to specify a new reference process to which the domain data model and the specification language belong. While the data model provides all relevant aspects, artifacts, and dependencies of an HMI, the specification language is based on the data model and allows concrete specification of the artifacts. Depending on the usage scenario, the concrete specification can be provided formally or non-formally. An advantage of this flexibility is that communication is facilitated by non-formal specifications, while refinements and implementations of specifications profit from formal specifications.

Since the specification language is provided as an XML schema, interoperations with other tools are supported. Furthermore, versioning and storing are possible in different concepts throughout the whole HMI development process.

The completeness, adequateness, and level of detail of the data model and the specification language were evaluated in a conjoint workshop with all project members (see chapter 5). The results of this evaluation will be used for improving both the data model and the specification language. As an important but currently missing artifact of the data model, we have already identified variation management, which will be included as soon as possible. Further evaluations of the data model and the specification language will be performed iteratively on a regular basis until the end of 2012. After that, the plan is to prove the whole approach by applying it to a real HMI development process. Demonstrators will show the applicability of the approach.

7. ACKNOWLEDGMENTS

The research described in this paper was conducted within the project automotiveHMI. The project automotiveHMI is funded by the German Federal Ministry of Economics and Technology under grant number 01MS11007.

8. REFERENCES


