Model-Based Testing of Infotainment Systems on the Basis of a Graphical Human-Machine Interface

Linshu Duan[1], Alexander Höfer[2], and Heinrich Hussmann[3]

[1] AUDI AG and Ludwig-Maximilians-Universität München
[2] AUDI AG, Infotainment Development and Testing
[3] Ludwig-Maximilians-Universität München, Institut für Informatik

Abstract. Automotive infotainment systems were getting more and more features in recent years. The complexity of HMI (human-machine interface) softwares is growing with the increasing requirements for stability, interactions, usabilities and internationalization of the automotive markets. Testing the HMI became a very time consuming and costly task. Because of multiplicity of HMI variants, a better test coverage is a goal for the development process of most manufacturers. Model-based testing is one way to achieve a better test coverage and keep the costs and complexity acceptable. However, the existing research approaches in the area of model-based HMI testing can not satisfy the needs for our testing purposes. In this doctoral thesis a model-based testing approach will be proposed for testing both the logical behavior and the graphical interface of automotive infotainment HMIs. As an important part of the testing approach a test-oriented HMI specification model will be proposed. It is a model, which describes the theoretical behavior of the HMI and contains the necessary information for automated HMI testing. Test generation methods and the design of the tests to be generated will also be proposed. These results can be generally used for testing advanced GUI-driven applications. In the work, some infotainment specific coverage criteria will be defined. Methods will be introduced, how the generated tests can be executed automatically and how the HMI behavior of the system under test can be observed and verified.

1 Introduction

A human-machine interface of an infotainment system is the interface, through which the user communicates with the infotainment system. It could consist of a graphical user interface, a control unit, a touch pad, a multifunction wheel and possibility speech input and output facilities. The focus of the doctoral research is to test the graphical user interface of infotainment systems. Both the logical behavior and the graphical interface of the HMI are our test purposes. The term HMI used in this work is resigned to the GUI part of the HMI.

The complexity of HMI softwares are growing with the increasing functionalities and complexity of infotainment systems very strongly [4]. A user interface

giving a faultless experience is one of the most important requirements of today's infotainment systems. However HMI errors are an essential part of all infotainment system errors during the development phase, because testing HMI at full length is very demanding, time consuming and costly. Test designers have to spend a lot of time for defining tests and specially adapting the existing tests for different system variants and software updates. The testers have to execute the tests step by step manually. For foreign-language systems testers are in demand, who have to be native-speaking and also adept at testing. The absence of native-speaking testers could lead to a low test coverage and hence more errors at end customers.

In the doctoral thesis a model-based testing approach will be proposed to resolve or support resolving theses problems. The following researches have to be done for the approach:

- Identification of an HMI specification concept, which is suitable for testing
- Identification of necessary information needed for testing
- Designing a test-oriented HMI specification model
- Identification of infotainment system specific test generation criterion
- Proposition of test generation methods suitable for HMI test generation
- Definition of methods and interfaces for test execution
- Definition of methods and interfaces for automatic observation and verification of the SUT (system under test) behavior

The doctoral thesis is still in progress. This paper only illustrates the identified purposes. Design details of the specification model and concrete methods still have to be completed and introduced in future papers.

2 Related Work

A number of research efforts have addressed the test automation of GUI applications. Some of them are based on record/ playback concepts [5]. The recorded sessions are later played back whenever it is necessary to recreate the same GUI states. Several attempts have been made to automate test case generation for GUIs, for which two different ways can be identified in the research. 1) Planning: Given a set of operators, an initial state and a goal state, a planner produces a sequence of operators that will transform the initial state to a goal state [8]. 2) Model-based approach: The behavior of SUT is verified with the expected behavior defined as models. Finite-state machine (FSM) [1] is a model usually used for defining the expected behavior. Other models e.g., event-flow model are also proposed in this area [7]. In [2] NModel is proposed for model-based testing of GUI-driven application. The NModel framework developed at Microsoft Research allows us to create a FSM model and generate tests automatically with its *offline test generator*. Each screen will be identified as a state in the model. The NModel approach has made experience with a sample system containing 11 screens and a simple menu behavior. The approach focuses GUI-driven applications on a PC and is not applicable for HMI testing without additional adaption

and extension. An infotainment HMI of the latest generation consists of up to about 1500 screens and very advanced mechanisms to support the complexity and to ensure the correctness of the HMI behavior. E.g., synchronization mechanism, history and deep history states, mediators observing data changes etc. It could be a very challenging task to make NModel supporting the complexity of a infotainment HMI and to adapt the concept for special characters of infotainment system tests. This approach does not face the problems of user inputs or verification of the input validity. The approach addresses PC applications, which contain the complete logic in itself. These applications do not exchange data with external logic or database. This abates the complexity of the model and the difficulties have to be faced. In [10] a model based software testing method for the web applications is discussed. This method focuses on testing the functionalities of the front end of web applications, i.e., the linking behavior of the links and forms in web applications. The behavior of the front end of the web application is modeled with statecharts. Tests verifying the linking behavior can be generated according to known criteria. However, this method has not yet found solutions to the problems of modeling the back-ends of a web application. The back-ends means the application logic of an GUI-driven application. The application logic is responsible for e.g., verification of user input validity or generating content as reaction of a button click. The verification result and calculated content should usually be displayed in the web or decide the next behavior of the GUI applications. Without the behavior of the back-end the generated tests do not have any conditions for the execution. It could lead to uncompleted or invalid tests for many test executions. In infotainment systems the behavior of the back-end is very complex and error-prone. Testing this logic is a very important test purpose.

The main objectives of the doctoral thesis is to design the complete testing process for the practice. This process covers realization of a test-oriented specification model, automatic test generation, test execution, as well as the observation and verification of the system behavior.

3 Proposed Approach

3.1 A Widely Used HMI Framework

Most of today's the HMI development tools base on GUI frameworks following the MVC (model-view-controller pattern) [9]. MVC defines the separation of the view, model and controller layer. In the view layer screens and contained graphical elements present data contained in the model. The model layer contains data to be displayed and in many of the implementations also the business logic. A controller processes the user inputs, changes the model data, informs the business logic and updates the visual data in the screens. The left side of Figure 1 shows the principle of the MVC pattern: the solid line presents a direct association, the dashed an indirect association, e.g., via an observer.

The right side of Figure 1 shows one of the ways, how infotainment HMIs implement the MVC. In infotainment HMIs, widgets are widely used. Widgets are graphical elements, which usually contain both the graphical presentation

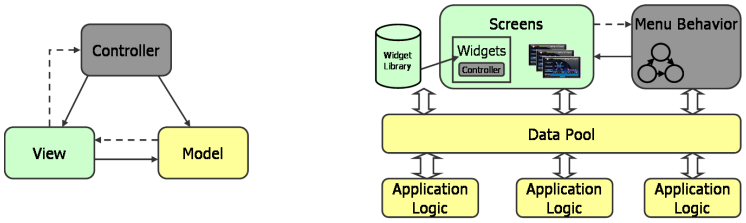

Fig. 1. Model-View-Controller Pattern and A Widely Used Structure of Infotainment HMIs

and the controllers, e.g., event handling. Some of the widgets own even some behaviors e.g., in a select list, entries can be focused by scrolling up or down and the focused entry will be highlighted by the list. Widgets actually breach the pattern, even so such of implementations are regarded as MVC implementation. Usually the widgets are preprogrammed and available for the view creation as libraries. To create instances of widgets, they have to be parametrized.

Today's HMI development tools support model-based implementation of controllers (We are not talking about the widget controllers here). A widely used model is the statechart diagram [6]. The statecharts define the behavior of the controller, thus it processes events and operates correspondingly. E.g., By a user event it enters into another state and replaces one view with another. With other words the statechart defines the menu behavior of the HMI.

In an HMI framework the component containing the data is widely called data pool. The data pool and the application logic in infotainment HMIs can be regarded as the model layer of the MVC pattern. Data pool contains variables which can be modified by the application logic. The controller can access this data directly e.g., to decide the behavior at a decision point. Frequently some widgets contained in different screens should display the same data. To avoid inconsistency between the screens, widgets are usually implemented as observers for the data. Applications logic in infotainment systems are usually connected with bus systems joining some databases and physical ECUs (electronic control units), e.g., CD player or navigation receiver. The application logic evaluates e.g., the validity of a city name given by the user based on the navigation database content. The evaluation result will be written in an agreed variable in the data pool. Widgets or the controller who are observers of this value will be informed about this data change. The applications logic are usually implemented manually, thus not model-based.

Almost all infotainment HMIs are event-based. Altogether a ready HMI implementation framework composes usually: 1. Preprogrammed widgets and a screen editor for creating screens, 2. Data models widely called data pool containing variables whose values are modified by the application logic and displayed by the screens, 3. Agreement of events for user inputs and internal changes, 4. A modeling editor for defining the menu behavior

3.2 Test-Oriented Specification Model

An HMI test-oriented specification model describes the theoretical behavior of the HMI and contains the necessary information for automated HMI testing. An assumption of the testing concept is that a test-oriented HMI behavior model is available in the proposed form. The most convenient way to achieve such a model is to add the testing required information into an HMI specification model emerged from the specification phase. A specification describes the theoretical behavior the implementation should conform to.

Model-based HMI specification [11] is the trend today. However, at many manufactures HMI development tools are still in use for the specification phase till now. The development tools are not specially designed for specifications being created by people with a different background than software developers. This leads to many informal description and errors in the specification model. By this reason some researches are arisen to find out specification concepts suitable for the design phases. Different specification concepts implement the model-view-controller pattern in different ways. It is important for our testing purpose, that the testing required information are completely accessible from the specification model for the test generation. Criterion important for testing will be identified in my work at first. Different specification concepts will be evaluated based on these criterion. One specification concept will chosen, which fills the requirements proximately. It will be used as the basis of the test-oriented model and extended for the testing purpose. The result is a test-oriented specification model, which has the minimal specification complexity but sufficient information for testing.

Tests will be generated from the test-oriented behavior model according to different coverage criteria.

3.3 Test Case Generation from the Test-oriented Specification Model

Three kinds of tests are defined: Menu behavior tests, widget behavior tests and design tests.

Menu Behavior Tests As the name implies, menu behavior tests verify the HMI menu behavior: whether the GUI switches to the correct screen in response to some inputs from the user or underlying application. A frequent error is that the GUI goes to an unexpected screen after the return button is pressed. It was described in the section above, that the menu behavior is defined by the statechart diagrams. Erroneous implementation of a history state in the statechart digram is usually the reason. In order to generate menu behavior tests, the statechart diagrams have to be traversed. Tests can be selected based on different coverage criteria. Selected tests contain both user actions and the expected screens. For test execution, user actions are transformed into test steps. The expected behavior is the occurrence of the expected screen. If the expected screen is defended as a ID, there must be an interface or a method reporting the ID of the currently displayed screen on the SUT.

Widget Behavior Tests Static widgets e.g., titles display particular static data and do not have a behavior. Whereas dynamic widgets e.g., select lists own some behaviors and change there looks depending on the context. E.g., the typical behavior of a list is, that the list can be scrolled through the contained lines, and the focused line is highlighted. There is typically a arrow, which appears to indicate that the user still can scroll down or up. Dynamic widgets are very error-prone. Tests acting with such of dynamic widgets inside the same screen without causing a menu change are defined as widget tests. In the most of the GUI frameworks, widgets are preprogrammed manually. I.e., the behavior of dynamic widgets is only described in program code. For the test-oriented HMI specification concept a widget model describing the widget behavior will be proposed. This model is similar to menu behavior model. Widget tests will also be generated automatically. The integration and usage of these models will be introduced in my work. Created widget tests are driven by the menu behavior tests. I.e., when an expected screen has been reached, the appropriate widgets tests will be invoked. E.g., the highlighting of the focused line and the arrow direction will be verified. For observation of the widget behaviors simple image processing methods are applied. They can recognize e.g., the text and color of a defined pixel area.

Design Tests Tests, which specially detect displaying errors are grouped into design tests. Overlapping texts, text out of the defined pixel area and erroneous color are frequent displaying errors. Design tests are based strongly on image processing and are also driven by menu behavior tests. When an expected screen has been reached, design tests using image processing methods will be invoked to verify the displaying effect of the data. A style-sheet containing the necessary information will be integrated in to the test-oriented specification.

Infotainment HMIs contain a lot of widgets and the behaviors of dynamic widgets can also be complex. It is usually not necessary to test all widget behaviors or all displaying effects. So test purposes have to be appointed at first. Error statistics of past projects will be evaluated. Depending on the test requirements, tests can be generated to detect the occurred errors or frequent errors of past projects.

Other Errors Currently, there are many errors which are easier to be found by manual tests then by the test automation. E.g., a title text in a foreign language, which is translated inconsistently to the context. The proposed test concept does not defined methods to detect this kind of errors directly, but it can be helpful. Screenshots can be made for all reached screens by the tests and provided for (native-speaking) testers for review. In this way, the testers do not have to execute the tests step by step manually.

Coverage Criteria In this work, some infotainment system specific coverage criteria are proposed, which can be combined with criteria known in model-based testing area [3].

1. Module Coverage
   Infotainment system functions can be divided into the following modules: phone, navigation, media, radio, address books and settings. Infotainment system tests are also divided into these modules and executed separately on different SUTs, because it is unusual that a test unit is equipped with full hardware devices for all the modules. Furthermore, many software updates affect only a subset of the modules. So generating tests among certain modules is very important for infotainment HMI tests.
2. Usage Oriented Coverage
   Division of infotainment system tests into different test levels are very important for the daily testing life. Levels differ the frequency and order of the test execution, and priority of handling the errors. Functions offered by infotainment system have very different usage frequency. For example: Entering a street occurs more often than entering a country. This usage frequency can be used to generate tests of different test levels.
3. Application Oriented Coverage
   GUI reacts to data coming from an application. In some situations, data changes are initialized by the application without any user request. For example: warning about empty tank. This kind of information is always very important for the safety of drivers. This behavior should be tested consciously. Test generation has to able to generated tests covering all of these cases.
4. Screen Coverage
   Each screen of the HMI presents a state of the infotainment system. According to screen coverage tests should be generated, which reach each system state at least once. This criterion is specially useful to help detecting errors mentioned above. Screenshot can be made from each reached screen. Many graphical and language errors can be found quickly by review of the screenshots.

3.4 Execution of Test Cases

There are several test tools for infotainment system in the market. This tools allow definition of tests and support automatic execution of defined tests on the SUT. Figure 2 demonstrates the principle how defined tests are executed automatically with help of a test tool. A test tool provides interfaces to bus systems and physical ECUs contained in the infotainment system. Test steps containing user events defined in the tests can be given in to the system. Behavior of the system and their ECUs are observed via particular interfaces and evaluated by means of expected behavior defined in advance.

Tests generated from our test-oriented model can be executed automatically with the help of such a test tool.

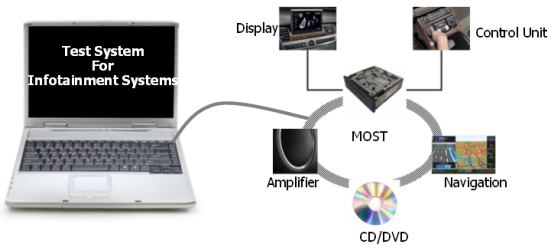

Fig. 2. Infotainment Testing System

4 Evaluation Methods

Two approaches are planned to evaluate the concept. The first one is using a beginning HMI project and creating an sample HMI specification for a chosen module, e.g., navigation. Tests should be generated and executed on the real test system. The number of the detected errors depends on the project phases and the level of system maturity heavily. But this is basically the way to detect the problems of the concept occurring in the practice. A second approach is defined to evaluate the success of the concept. The evaluation will base on the following metrics:

1. The number of automatically findable errors
   Error statistic of a past project in a comparable volume will be evaluated. The number of the errors, which were detected manually in the past project but automatically findable via our concept will be measured.
2. The timing of the found errors
   The later an error is found the more expensive it is. The concept allows to test the system automatically and systematically as soon as the SUT is available. Based on the error statistics, the number of errors which can be found earlier via our concept will be measured.
3. Test coverage
   Tests will be generated from the sample model. It will be evaluated whether tests defined by a test designer are covered by the generated tests and whether generated tests verify more states and functions. However the sample model can not provide enough coverage to demonstrate the test coverage for the whole HMI. Therefore the test coverage has to be estimated in some way.
4. Feedback of screen reviewer
   Screenshots made by menu behavior tests, specially of foreign language systems will be given to the (native-speaking) reviewers. The number of errors found by reviewing the screenshots will be measured. Also feedbacks about the duration and the comfort to find the errors are considered.

5 Conclusion

A model-based testing approach for graphical HMIs of infotainment systems is researched in the thesis. A test-oriented HMI model is designed, which describes the theoretical behavior of the SUT and contains test-relevant information. This concept appropriates to testing advanced GUI-driven applications. Special criterion, methods and interfaces for testing infotainment system HMIs are proposed. Based on this approach tests can be generated and executed automatically. Testing of foreign-language systems can be supported. A better test coverage can be achieved in this way. The approach considers the real complexity of advanced HMIs and addresses practical problems and requirements.

References

1. F. Belli, *Finite-state testing and analysis of graphical user interfaces*, in ISSRE '01: Proceedings of the 12th International Symposium on Software Reliability Engineering, Washington, DC, USA, 2001, IEEE Computer Society, p. 34.
2. V. Chinnapongse, I. Lee, O. Sokolsky, S. Wang, and P. L. Jones, *Model-based testing of gui-driven applications*, in Software Technologies for Embedded and Ubiquitous Systems, vol. 5860/2009, Springer-verlag New York Inc, 2009, pp. 203–214. 7th IFIP WG 10.2 International Workshop, SEUS 2009 Newport Beach, CA, USA, 2009 Proceedings.
3. C. Gaston and D. Seifert, *Evaluating coverage based testing*, in Model-Based Testing of Reactive Systems, Springer-Verlag New York, LLC, 2005.
4. Q. S. S. GmbH, *Ready for the future: Software trends for in-car infotainment systems*, (2005). `http://www.epn-online.com/page/18329`.
5. Hammontree, M. L., Hendrickson, J. J., and H. andBilly W., *Integrated data capture and analysis tools for research and testing on graphical user interfaces*, (1992), pp. 431–432. CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems.
6. D. Harel, *Statecharts: A visual formalism for complex systems*, Sci. Comput. Program., 8 (1987), pp. 231–274.
7. M. A. M., *An event-flow model of gui-based applications for testing: Research articles*, Softw. Test. Verif. Reliab., 17 (2007), pp. 137–157.
8. M. A. M., P. M. E., and S. M. Lou, *Hierarchical gui test case generation using automated planning*, IEEE Trans. Softw. Eng., 27 (2001), pp. 144–155.
9. H. Mössenböck, *Objektorientierte Programmierung in Oberon-2*, Springer-Verlag, Oktober 1998.
10. H. Reza, K. Ogaard, and A. Malge, *A model based testing technique to test web applications using statecharts*, in ITNG '08: Proceedings of the Fifth International Conference on Information Technology: New Generations, Washington, DC, USA, 2008, IEEE Computer Society, pp. 183–188.
11. G. Wegner, P. Endt, and C. Angelski, *Das elektrische lastenheft als mittel zur kostenreduktion bei der entwicklung der menschen-machine-schnittstelle von infotinament-systemen im fahrzeug*, in Infotainment Telematik im Fahrzeug, Expert-Verlag GmbH, 2004, pp. 38–45.