2010 Second International Conference on Advances in System Testing and Validation Lifecycle

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

Linshu Duan*, Alexander Höfer†, Heinrich Hussmann‡
*AUDI AG and Ludwig-Maximilians-Universität München
Email: linshu.duan@audi.de
†AUDI AG,Infotainment System Development and Testing
Auto-Union-Str., 85055 Ingolstadt Germany
Email: alexander.hoefer@audi.de
‡Ludwig-Maximilians-Universität München,Institut für Informatik
Amalienstraße 17, 80333 Munich Germany
Email: heinrich.hussmann@ifi.lmu.de

***Abstract*—Automotive infotainment systems were getting more and more features in recent years. The usability of their HMIs (human-machine interfaces) has been improved considerably. However the complexity of the HMI software is growing. Testing the HMI became very demanding and time consuming. Because of multiplicity of HMI variants, a better code coverage is a goal for the development process of most manufacturers. Model-based testing is one way to achieve a better code 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 the work a model-based testing approach will be proposed for testing both the logical behavior and the graphical interface of the automotive infotainment HMI. As an important part of the testing approach a test-oriented HMI specification model will be designed. It is a model, which describes the required behavior of the HMI and contains the necessary information for the testing process. Test generation methods and the design of tests will also be proposed. These results can be generally used for testing advanced GUI-driven applications. Specific coverage criteria for infotainment HMIs, methods for automatic test generation and verification of the system behavior are also focuses of the work. The paper introduces the ideas and the goals of our model-based testing approach for infotainment HMIs.**

***Keywords*-Model-Based Testing; Automated HMI Testing; Infotainment System Tests**

I. 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 possibly speech input and output facilities. The focus of our research is also to test the graphical human-machine 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 software is growing with functionality and complexity of infotainment systems very strongly [1]. A user interface giving a faultless experience is one of the most important requirements of today's infotainment systems. However HMI faults are an essential part of all infotainment system faults during the development phase, because testing HMI at full length is a very demanding, time consuming and costly task. Test designers have to spend a lot of time for defining tests and adapting existing tests for different system variants and software updates. Test executors have to execute the defined tests step by step manually. For foreign-language systems test executors are in demand, who are not only native speakers, but also adept at testing. Absence of native-speaking testers could lead to low code coverage and higher fault rates for foreign-language systems.

The following researches have to be done for the model-based HMI testing approach:

- Designing a test-oriented HMI specification model
- Identification of test generation criteria and test generation methods specific for infotainment HMIs
- Definition of methods and interfaces for automatic test execution, observation and verification of the SUT (system under test)

Researches and methods in GUI-testing area are discussed in Section II. Section III explains a widely used HMI framework as preliminaries for the introduction of the test-oriented HMI specification model. Four kinds of HMI faults and three kinds of tests are introduced in Section IV. Afterwards some specific coverage criteria are introduced in V. Test execution methods are presented in VI. In Section VII metrics for the evaluation of the concept are mentioned.

II. Related Work

A number of research efforts have addressed the test automation of GUI applications. Some of them are based on record and playback concepts [2]. 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 [3]. 2) Model-based approaches: The behavior of the SUT is verified with the expected behavior defined as models. Finite-state machine (FSM) [4] is a model usually used for defining the expected behavior. Other models e.g., event-flow model are also proposed in this area [5]. In [6] NModel is proposed for model-based testing of GUI-driven applications. The NModel framework developed at Microsoft Research allows us to create a FSM model and generate tests automatically. Experience has been made with a sample system containing 11 screens and a simple menu behavior. Complex user inputs and the system reaction to the user inputs are not considered. An infotainment HMI of the new 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. 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. So this approach can not satisfy the needs for testing infotainment HMIs directly. Furthermore, most of the existing approaches focus on testing the menu behavior. Detection of widget and displaying faults are not faced. The main objectives of the work are a model-based HMI testing approach, which satisfies the needs and complexity of testing infotainment HMIs.

III. Test-Oriented HMI Specification Model

This section at first introduces a widely used HMI framework. Based on this knowledge, we discuss the test-oriented specification model needed for test generation.

A. A Widely Used HMI Framework

Most of today's HMI development tools are based on GUI frameworks following the MVC (model-view-controller) pattern [7]. MVC defines the separation of the view, model and controller layer. In the view layer screens and contained graphical elements present the data contained in the model. The model layer contains the data to be displayed and in many of the implementations also the business logic. Controller processes the user inputs, changes the model data, informs the business logic and updates the visual data in the screens. The left part of Figure 1 shows the principle of MVC: the solid line presents a direct association, the dashed an indirect association, e.g., via an observer.

The right part of Figure 1 shows one of the ways, how infotainment HMIs implement the MVC pattern. In infotainment HMIs, widgets are widely used. Widgets are

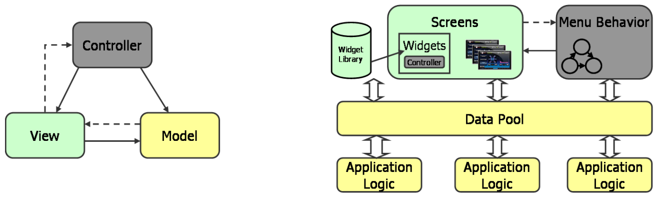

Figure 1. Model-View-Controller Pattern and A Widely Used Structure of Infotainment HMIs

graphical elements, which usually contain both the graphical presentation and the controllers, e.g., event handling. Some of the widgets even own some behaviors, e.g., in a select list entries can be focused by scrolling up or down and the focused entry will be highlighted. Widgets actually breach the pattern. Even so, such implementations are regarded as model-view-controller implementations. Usually the widgets are preprogrammed and available as libraries for the screen creation. 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 [8]. The statecharts define the behavior of the controller, i.e., it processes events and operates correspondingly. E.g., By a user event it enters into another state and replaces one screen with another. In other words the statechart defines the menu behavior of the HMI. In an HMI framework the component containing the data is commonly called data pool. The data pool and the application logic in infotainment HMIs can be regarded as the model of the MVC pattern. The 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. The application logic in infotainment systems is 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 application logic is usually implemented manually, thus not model-based.

B. Test-Oriented Specification Model

A test-oriented HMI specification model describes the required behavior of the HMI and contains the necessary information for the testing process. An assumption of the concept is that a test-oriented HMI specification model is available in the proposed form. Depending on the HMI development process, a HMI test-oriented specification model could be completely created by test engineers for testing goals, or created by enriching the information required for testing into an HMI specification model, if a model-based HMI specification already exists. A HMI specification is usually created by the HMI specification group and describes the required behavior the implementation should conform to. Model-based HMI specification [9] is the trend today. However, at many manufactures HMI development tools are still in use for the specification phase until 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. Some researches are arisen to find out specification concepts suitable for the design phases and different specification concepts implement the MVC pattern in different ways. No matter in which way the test-oriented HMI specification model will be resulted, it is important for our testing purpose that all testing required information can be contained in the specification and is completely accessible for the test generation. Requirements in the HMI framework and the specification concept will be identified for our testing purposes. The work and a prototype will base upon them. The result should be 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.

IV. Test Generation

HMI faults can be categorized into four types: faults in the menu behavior, widget behavior, displaying data and languages. Our concept is designed to detect faults of the first three types automatically and to support testers in finding faults of the last type. So three kinds of tests will be generated to face the first three faults types.

A. 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 diagram. Erroneous implementation of history states in the statechart diagram 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 screen. 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 defined as an ID, there must be an interface or a method reporting the ID of the currently displayed screen on the SUT.

B. 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 their looks depending on the context. E.g., the typical behavior of a list is the highlighting of the focused line by scrolling through the list. There is typically an arrow, which appears to indicate that the user still can scroll down or up. Dynamic widgets are very error-prone. Tests acting with such 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. Thus, 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. 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. Integration plan and methods avoiding impacts have still to be worked out. For observation of the widget behaviors simple image processing methods are applied.

C. Display Tests

Tests specially for detecting displaying errors are grouped into display tests. Overlapping texts, text out of the defined pixel area and erroneous color are frequent displaying errors. Display tests are based strongly on image processing and are also driven by menu behavior tests. A style-sheet containing the necessary information will be integrated into the test-oriented specification.

Language errors occur very frequently during the HMI development process, e.g., a title text in a foreign language, which is translated inconsistently to the context, or another text than specified is displayed as the title text because of misunderstood widget IDs. Currently language errors are found manually, because automating the detection of language errors causes more work than benefits. Our concept does not define methods to detect language errors directly or automatically, 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.

V. Coverage Criteria

Coverage criteria are used to instruct the test generation and to measure the quality or, more precisely, the adequacy of the tests. Well-known coverage criteria [10] such as structural coverage (e.g., state coverage and transition coverage) and functional criteria are also suitable for HMI testing and will be used for our menu and widget behavior tests.

In this work, some infotainment system specific coverage criteria are proposed, which can be combined with the known criteria. Some examples are:

1) Usage Oriented Coverage
   Division of infotainment system tests into different test levels is very important for the daily testing life. Levels are used to distinguish the frequency and the order of test execution as well as the priority to handle the found faults. Functions offered by infotainment system have very different usage frequency. E.g., entering a street occurs more often than entering a country. This criterion is relevant for all three kinds of tests, since if a screen is contained in a test of level 1, the behavior and the displaying effect of the contained widgets also have high priority.
2) Application Oriented Coverage
   The GUI reacts to data coming from an application. In some situations, data changes are initialized by the application without any user requests. E.g., 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 generate tests covering all of these cases. This criterion is relevant for menu behavior tests, since the reaction of the HMI to the incoming application data is to switch to another screen (information screen).
3) Screen Coverage
   Each screen of the HMI presents a state of the infotainment system. Tests generated according to the screen coverage should reach as many as possible screens defined in the system. This criterion is especially useful to help finding the language errors. Screenshots can be made from all reached screens. Many graphical and language errors can be found quickly by reviewing the screenshots. This criterion is only useful for menu behavior tests.

Furthermore widget coverage, language coverage and user input coverage are also important criteria for infotainment HMI testing.

VI. Execution of Tests

There are several test tools for infotainment systems in the market. These 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 the help of a test tool. A test tool provides interfaces to bus systems and physical ECUs contained in the infotainment system. Test steps can be executed by the test 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 HMI specification model can be executed automatically with the help of such a test tool.

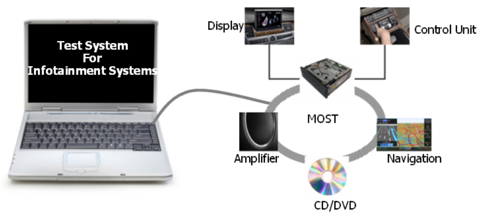

Figure 2. Infotainment Testing System

VII. Evaluation Methods

Two approaches are planned to evaluate the concept. The first one is using a beginning HMI project and creating a sample HMI specification for chosen functions. Tests should be generated and executed on the real test system. This is basically the way to detect problems occurring in the practice. A second approach is defined to evaluate the success of the concept. The evaluation will be based on the following metrics: 1) Error statistic of a past project in a comparable volume will be evaluated. It will be counted how many faults now can be detected automatically, which were found manually. 2) Timing of the detected faults. The later a fault is found the more expensive it is. The concept allows testing the system automatically and systematically as soon as the SUT is available. 3) Code coverage of manually defined tests will be compared with the one of automatically generated tests based on some chosen coverage criteria. 4) Time saved by reviewing screenshots. 5) The quality of the reported faults.

VIII. Conclusion

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

References

[1] Q. S. S. GmbH, “Ready for the future: Software trends for in-car infotainment systems,” 2005. [Online]. Available: http://www.epn-online.com/page/18329

[2] Hammontree, M. L., Hendrickson, J. J., and H. andBilly W., “Integrated data capture and analysis tools for research and testing on graphical user interfaces,” pp. 431–432, 1992, cHI ’92: Proceedings of the SIGCHI conference on Human factors in computing systems.

[3] M. A. M., P. M. E., and S. M. Lou, “Hierarchical gui test case generation using automated planning,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 2, pp. 144–155, 2001.

[4] 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: IEEE Computer Society, 2001, p. 34.

[5] M. A. M., “An event-flow model of gui-based applications for testing: Research articles,” *Softw. Test. Verif. Reliab.*, vol. 17, no. 3, pp. 137–157, 2007.

[6] 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*. Springer-verlag New York Inc, 2009, vol. 5860/2009, pp. 203–214, 7th IFIP WG 10.2 International Workshop, SEUS 2009 Newport Beach, CA, USA, 2009 Proceedings.

[7] H. Mössenböck, *Objektorientierte Programmierung in Oberon-2*. Springer-Verlag, Oktober 1998.

[8] D. Harel, “Statecharts: A visual formalism for complex systems,” *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[9] 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.

[10] C. Gaston and D. Seifert, “Evaluating coverage based testing,” in *Model-Based Testing of Reactive Systems*. Springer-Verlag New York, LLC, 2005.