Automated UI Evaluation based on a Cognitive Architecture and UsiXML

Jan-Patrick Osterloh[1], Rene Feil[1], Andreas Lüdtke[1], Juan Gonzalez-Calleros[2, 3]

[1] OFFIS Institute for Information Technology, Escherweg 2,
26121 Oldenburg, Germany
{osterloh, luedtke}@offis.de, rene.feil@informatik.uni-oldenburg.de
[2] Faculty of Computer Sciences,
Benemérita Universidad Autónoma de Puebla,
72400 Puebla, Mexico
juan.gonzalez@cs.buap.mx
[3] Research and Development Unit, Estrategia 360
Puebla, Mexico

Abstract. In this paper, we will present a method for automated UI evaluation. Based on a formal UI description in UsiXML, the cognitive architecture CASCaS will be used to predict human performance on the UI, in terms of task execution time, workload and possible human errors. In addition, the UsabilityAdviser tool can be used to check the UI description against a set of usability rules. This approach fits well into the human performance and error analysis proposed in the European project HUMAN, where virtual testers (CASCaS) are used to evaluate assistant systems and their HMI. A first step for realizing this approach has been made by implementing a 3D rendering engine for UsiXML.

Keywords: UI evaluation, UsiXML, cognitive architecture, CASCaS, UsabilityAdviser

Introduction

In order to further reduce the cost of Human Machine Interface (HMI) design, while reducing human error and increasing usability at the same time, the HMI development process has to be improved, by integrating the evaluation of User Interfaces (UI) into the design process. The European project HUMAN (7th Framework Programme) aimed at developing virtual testers, in order to improve the human error analysis of new assistance systems, including User Interfaces. In this paper, we will describe how cognitive models can be used to improve the development of UI. The objective is to provide a tool for automated UI evaluation, in terms of predicting cognitive workload, execution times, human error as well as compliance to HMI guidelines. A similar

approach has already been tackled in CogTool [8], a UI prototyping tool, which uses a predictive human performance model to automatically evaluating GUI design.

In the next section, we will discuss CogTool and its application in the industrial process. Then, we will propose another approach for UI evaluation, which should improve some of CogTools shortcomings, and could be integrated in the industrial design process.

State-of-the-Art

Currently there are different approaches to evaluation of UI designs. Beside the classical approach of evaluation with test users, automatic evaluation with tools is used. The common major shortcoming of any evaluation tool is that the evaluation logic is hard coded in the evaluation engine [13], for example, two leaders of the web evaluation market, Bobby and A-Prompt only provide the choice between the guidelines of W3C or Section 508, which makes them very inflexible for any modification of the evaluation logic or any introduction of new guidelines. In addition, many of them do not offer much possibilities of controlling the evaluation process like choosing which guideline to evaluate, or the level of evaluation at evaluation time. Not only existing tools cannot accommodate different and multiple bases of guidelines or usability knowledge but also they force the evaluator to evaluate the GUI in a predefined way: it is not possible to focus the evaluation on only some parts of the GUI, for instance by considering only those guidelines that are concerned with the contents. The goal here is to develop an evaluation tool that addresses the above shortcomings, such as the support of multiple bases of guidelines (accessibility, usability, or both) on-demand (partial or total evaluation), with different levels of details (a presentation for a developers and a presentation for the person who is responsible for attributing the accessibility certification). For this purpose, an evaluation engine should be developed that perform guidelines evaluation or other independently of guidelines and usability knowledge.

Another, newer approach is the evaluation based on cognitive models. CogTool is a general purpose UI prototyping tool, which uses a predictive human performance model to automatically evaluating GUI design ([8], [3]). In order to perform an analysis, the analyst defines first a prototype of the interface (based on standard set of UI widgets, like buttons, sliders, menus), including possible transitions between different interfaces. Then, a number of tasks are demonstrated on the design, which are recorded and build the basis for the interaction tasks. Then the cognitive architecture ACT-R [2] is used to predict e.g. cognitive workload, and task execution times.

While CogTool allows fast prototyping and evaluation, the UI prototype itself can only be imported and exported as HTML code, and cannot be reused for the final interface. In the transportation domain, model driven development has become standard for development of assistance systems. Using CogTool in an industrial process would require that a given design has to be re-implemented in CogTool, and after the improvements are made within CogTool, these have to be implemented in the final version of the system, as CogTool is currently neither integrated in a UI

development tool, nor in a modelling tool used in the industry (like Scade, Matlab, or Rhapsody).

In addition, the need to demonstrate the tasks performed in each scenario from start to the end seems for a larger set of scenarios to time consuming. Re-usage of the UI prototype that allows model driven development, as well as re-usage of the tasks that are performed, are main requirements for the proposed method.

Method

In the HUMAN project, a method for system evaluation has been proposed, that integrates cognitive testers into the design process of aircraft manufacturers, as well as integrating an offline evaluation tool. Main idea is to use the system models (e.g. defined in Matlab) in a simulation together with virtual testers, in order to test the system in an early design phase. For the UI development, the HUMAN method proposes to use UsiXML, which stands for USer Interface eXtensible Markup Language. UsiXML is a XML-compliant mark-up language that describes the UI for multiple contexts of use, i.e. interactive applications with different types of interaction techniques, modalities of use, and computing platforms can be described in a way that preserves the design independently from the physical computing platform. Fig. 1 shows a possible architecture for automated UI evaluation, with UsiXML, and the cognitive architecture CASCaS.

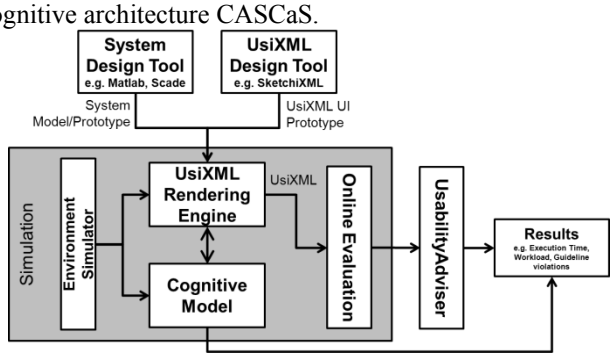

Fig. 1. Architecture for automated UI evaluation

The first step in the proposed method is to model the system functionality in a design tool like Matlab or Scade, and to model the UI using UsiXML. There are multiple tools to create rapid prototypes of GUI in UsiXML, e.g. SketchiXML [6], with no need for writing XML directly. In the next step, a simulation is used for the evaluation: A rendering engine for UsiXML is used to display the UsiXML to the virtual tester, or a human user respectively. On the same time, it controls the interaction between the system model and the UI, i.e. if the virtual tester presses a button this is propagated to the system model and the UI. Each interaction may result in changes on the UI, which are then re-translated into UsiXML and send to an online evaluation tool. This evaluation tool calculates then online the workload that is

needed for this status of the UI. The cognitive model, which is described in more detail in the next section, also calculates workload (e.g. for motor actions, goal switches, etc.) for the overall simulation, as well as task execution times, gaze distribution and predicts possible human errors. A simulator provides additional information, e.g. route, traffic and weather information.

In an offline evaluation, it is also possible to use the UsabilityAdviser [4] for analysing the UI on compliance to certain usability rules, like certain undesired colour combinations (e.g. yellow on white background).

Cognitive Model

The cognitive architecture CASCaS has initially been developed in the 6th European Commission Framework Programme project ISAAC (see [9]), and has been widely extended and used in other projects since then. CASCaS has been used to successfully model perception [10], attention allocation [15], decision making (of drivers) [14] and human errors [9] of aircraft pilots and car drivers.

CASCaS is based on Rasmussen's [12] three behaviour levels in which cognitive processing takes place: skill-based, rule-based and knowledge-based behaviour. The levels of processing differ with regard to their demands on attention control dependent on prior experience: skill-based behaviour is acting without thinking in daily operations, rule-based behaviour is selecting stored plans in familiar situations, and knowledge-based behaviour is coming up with new plans in unfamiliar situations. Anderson [1] distinguishes very similar levels, but uses the terminology of autonomous, associative, and cognitive level, which will be used throughout the paper. Fig. 2 gives an overview on the components of CASCaS. These components form the following control loop: The "Perception" component retrieves the current situation from the "Simulation Environment", and stores the information in the "Memory" component. The "Processing" component contains components for the behaviour layers. These layers can retrieve information from the memory and process this information according to their cognitive cycle (rule-based or knowledge-based). The layers may store new information in the memory, or start motor actions in the "Motor" component. Each component is based on psychologically and physiologically sound theories, e.g. from cognitive psychology. They implement detailed models of timing, e.g. for eye movements, such that CASCaS allows prediction of task execution times. In addition, the attention allocation can be predicted, based on top-down (rules) and bottom-up (peripheral view/selective attention) processes [10]. For the calculation of eye- and hand movements, CASCaS needs information on the positioning of the instruments. We call this information the "topology", which is currently defined in a customized XML format, which should be exchanged by a UsiXML format in future implementations.

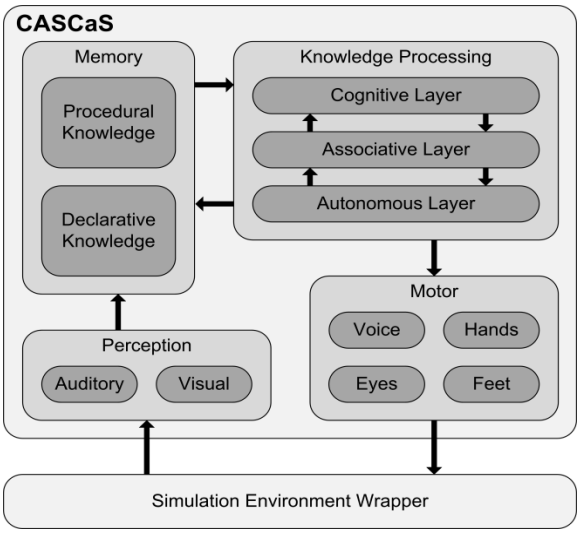

Fig. 2: Architecture and components of CASCaS

UsiXML

UsiXML is a XML-compliant mark-up language which consists of a declarative User Interface Description Language (*UIDL*). It describes user interfaces for multiple contexts of use such as Graphical User Interfaces (GUIs), Auditory- and Multimodal User Interfaces and their constituting elements such as widgets, controls and containers [7]. Using UsiXML, a UI developer is able to model a description of interactive applications with different types of interaction techniques and modalities in a device and computing platform independent notation.

UsiXML provides an MDE approach for the specification of user interfaces and is based upon the architecture of the CAMELEON Reference Framework [5]. This framework defines UI development steps for multi-context interactive applications. Fig. 3 shows a simplified version of this development process.

The rendering engine is placed between the layers three and four in Fig. 3. A UsiXML Concrete User Interface description serves as input data. This description is converted by a UsiXML parser and forwarded to the rendering engine. The result

after this step is a Final User Interface according to the CUI.

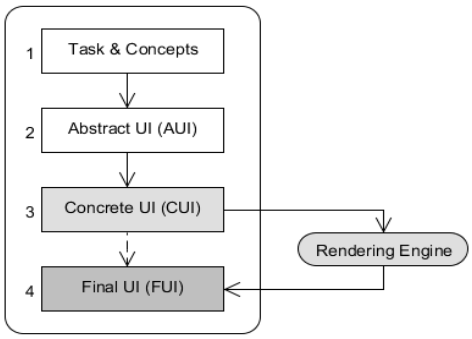

Fig. 3. The User Interface Reference Framework, (cf [11])

Rendering Engine

Fig. 4 shows a simplified architecture of the rendering engine which consists of four main parts. First, a UsiXML parser, which conforms to a language processing system, converts a CUI description into an internal and renderable format. After this step, the converted data is passed to the rendering engine for further handling. The second component is based upon the MVC architectural pattern and handles the user actions, provides the user interface, stores the converted CUI data, supplies the application's main loop and delivers strategies for the program flow. Configuration files and log files are handled by this part of the application, too. The fourth component is a mathematical library including a useful set of algebraic and calculus functions. Finally, the rendering engine itself consists of 6 ancillary parts, as shown Fig. 5. A further component shown in Fig 4 is a module for inter process communication (IPC). This part is planned for future implementation steps, e.g. to connect the system model, CASCaS, or other tools.

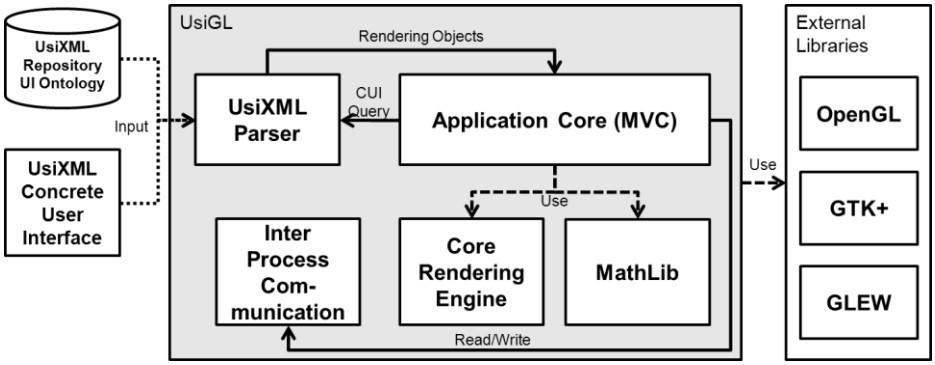

Fig. 4. Architecture for Rendering Engine

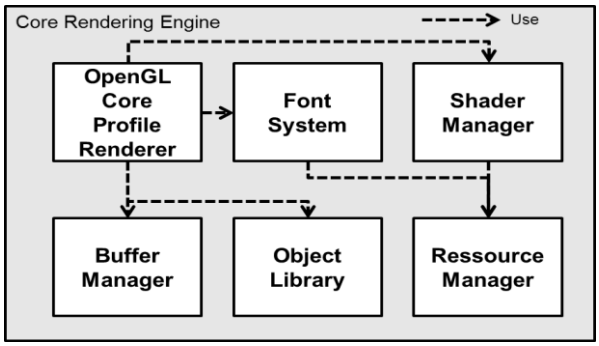

Fig. 5. Components of the Core Rendering Engine

The main component of the Core Rendering Engine is an OpenGL core profile renderer, which includes the functionality for drawing primitives and complex geometric objects, and allows geometry and scene management respectively manipulation. Summarized, it serves as a programming framework for creating and preparing the input for OpenGL. Further, this part includes a font system for font rendering, a buffer manager, a shader[1] manager for loading and preparing shader programs including a common set of shader pairs, and a resource manager for loading external resources like textures, fonts and additional shaders. The sixth component is an object library which contains a pre-rendered set of GUI objects like buttons and labels. OpenGL itself is a low-level rendering API. It doesn't include functions for drawing geometric objects like cylinders or spheres or GUI elements like buttons. It's up to the application developer to implement algorithms for drawing these objects. For that reason there is a need for the development of such an object library. The included object library is in an early stage and accordingly limited.

UsabilityAdviser

The global process for automatic evaluation with the UsabilityAdviser is depicted in Fig. 6. The “Knowledge Base” contains a formalisation of rules for good ergonomy and accessibility. This knowledge base is a collection from ergonomic guidelines, for instance, structures (Smith and Mosier) or various recommendations that are encoded in a formal format, using the UsiXML language. The knowledge base is used by the “Formal rules compiler” to load and parse the rules. Once this internal structure is created the tool performs a data analysis of the UI, encoded in UsiXML, which may be developed in a UsiXML editor. The UsabilityAdviser search for violations of rules formalized through the automatic evaluation of UI data. Finally, a report on the found violations of ergonomics and accessibility is presented. One major challenge is to create and update the knowledge base on ergonomic rules, which requestes a

1 Programmable shading is the current state of the art in real-time computer graphics. Today's graphics cards are highly programmable and the term of shader refers to according programs, written in high level languages like GLSL, HLSL or Cg, which are executed by programmable chips on modern graphics card.

complete review and compilation of existing rules from different sources. These rules are often expressed in a natural language that is normally more complex and open compared to a programming language. Anyway, the UsabilityAdviser provides an extensible way of evaluation from multiple sources of guidelines for (parts of) a User Interface.

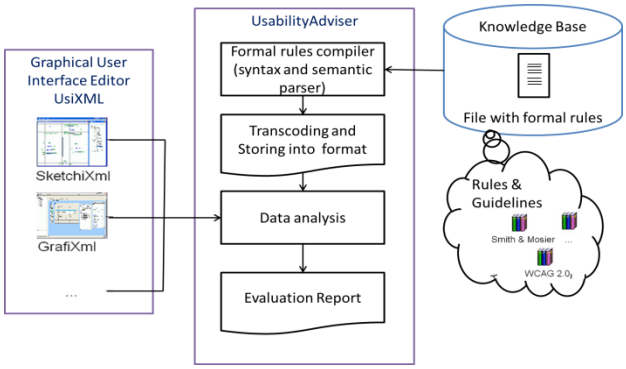

Fig. 6. Global process for automatic evaluation

Summary and Next Steps

We proposed an approach for automated UI evaluation with a cognitive architecture, which is usable in industrial application. It uses a model driven approach, and is connectable to tools that are already in use in the industry, like Matlab or Scade, which allows re-use of the models defined by the system designers. UsiXML provides a model driven development for the industry. These models together can be used for automated UI evaluation, for which the HUMAN project has already implemented some prototypical tools. Still an open issue is the connection between CASCaS and UsiXML. A first step in this direction is the implementation of a UsiXML rendering engine, which can be connected to CASCaS. Another open issue is the link between the UsiXML UI description and the system model. As tools like Matlab or Scade use the mechanism of Events for interaction, an extension to UsiXML with a mapping to such events could be the solution. The rendering engine could then be extended to trigger such events when there is interaction with the UI elements, e.g. on button clicks, and to transfer events back to certain changes in the UI (e.g. opening of a dialog).

References

1. Anderson, J. R.: Learning and Memory, John Wiley & Sons, Inc. (2000)
2. Anderson, J. R.; Bothell, D.; Byrne, M. D.; Douglass, S. A.; Lebiere, C.; Qin, Y.: An Integrated Theory of Mind. Psychological Review, Vol 111(4), Oct 2004, pp. 1036-1060 (2004)
3. Bellamy, R., John, B. E., Richards J., and Thomas J.: Using CogTool to model programming tasks. In Evaluation and Usability of Programming Languages and Tools (PLATEAU '10). ACM, New York, NY, USA, Article 1, 6 pages. http://doi.acm.org/10.1145/1937117.1937118 (2010)
4. Bossche, P. vanden: Développement d'un outil de critique d'interface intelligent : UsabilityAdviser, M.Sc. thesis, Université catholique de Louvain, Louvain-la-Neuve (2006).
5. Calvary, G., Coutaz, J., Bouillon, L., Florins, M., Limbourg, Q., Marucci, L., Paternò, F., Santoro, C., Souchon, N., Thevenin, D., Vanderdonckt, J.: The CAMELEON Reference Framework, Deliverable 1.1, Version V1.1, CAMELEON Project (2002)
6. Coyette, A., Vanderdonckt, J., and Limbourg, Q., SketchiXML: A Design Tool for Informal User Interface Rapid Prototyping, in Proc. of International Workshop on Rapid Integration of Software Engineering techniques, RISE'2006 (Geneva, 13-15 September 2006), N. Guelfi, D. Buchs (Eds.), Lecture Notes in Computer Science, Vol. 4401, Springer-Verlag, Berlin, 2007, pp. 160-176 (2006)
7. Guerrero García, J., Lamaigre, C., González Calleros, J. M., Vanderdonckt, J.: Model Driven Approach to Design User Interfaces for Workflow Information Systems, Journal of Universal Computer Science, vol 14, no 19, pp 3160-3173 (2008)
8. John, B. E.: Using Predictive Human Performance Models to Inspire and Support UI Design Recommendations. Proceedings of CHI 2011, ACM New York (2011)
9. Lüdtke, A., Cavallo, A., Christophe, L., Cifaldi, M., Fabbri, M., Javaux, D.: Human Error Analysis based on a Cognitive Architecture. In: Reuzeau, F., Corker, K., Boy, G. (eds) Proceedings of the International Conference on Human-Computer Interaction in Aeronautics (HCI-Aero), 20.-22.09.2006, Seattle, USA. Toulouse, Cépaduès-Editions, France, pp. 40-47 (2006)
10. Lüdtke, A., Osterloh, J-P.: Simulating Perceptive Processes of Pilots to Support System Design. In T. Gross, et al. (eds), Proceedings Human-Computer Interaction (INTERACT), Springer, pp. 471-484 (2009).
11. Molina, J.P., Vanderdonckt, J., Montero, F., González, P.: Towards Virtualization of User Interfaces based on UsiXML, Proc. of Web3D 2005 Symposium, 10th International Conference on 3D Web Technology (Bangor, 29 March-1 April 2005), ACM Press, New York, pp. 169-178 (2005)
12. Rasmussen, J.: Skills, Rules, Knowledge: Signals, Signs and Symbols and other Distinctions in Human Performance Models, IEEE Transactions: Systems, Man and Cybernetics, SMC-13, pp.257-267 (1983)
13. Vanderdonckt, J., Beirekdar, A., Automated Web Evaluation by Guideline Review, Journal of Web Engineering, Vol. 4, No. 2, 2005, pp. 102-117.
14. Weber, L., Baumann, M., Lüdtke, A., Steenken, R.: Modellierung von Entscheidungen beim Einfädeln auf die Autobahn. In A. Lichtenstein, C. Stößel, C. Clemens (Hrsg), 8. Berliner Werkstatt, Mensch-Maschine-Systeme, Düsseldorf: VDI Verlag, S. 86-91 (2009)
15. Wortelen, B., Lüdtke, A.: Ablauffähige Modellierung des Einflusses von Ereignishäufigkeiten auf die Aufmerksamkeitsverteilung von Autofahrern. In A. Lichtenstein, C. Stößel, C. Clemens (Hrsg.), 8. Berliner Werkstatt, Mensch-Maschine-Systeme, Düsseldorf: VDI Verlag, S. 80-85 (2009)