Concept for automated usability evaluation of graphical user interfaces

Karen Insa Wolf [*] **Rashik Thalappully** [*] **Stefan Goetze** [*]
Frank Wallhoff [*,**]

[*] *Fraunhofer Institute for Digital Media Technology IDMT, Oldenburg, Germany (email: insa.wolf@idmt.fraunhofer.de)*
[**] *Jade University of Applied Sciences, Institute for Technical Assistive Systems, Oldenburg, Germany*

Abstract: In this paper, the concept of a tool CogUA (Cognitive Usability Analysis) for automated usability evaluation of the Graphical User Interfaces (GUI) is presented. Results based on a first prototype implementation are discussed. The tool is designed to support the development of GUIs for an intuitive and safe operation of special-purpose machines. It aims at the prediction of usability before launching new user interfaces. The automated analysis can help to correct design errors at an early stage, which reduces the risk of software modifications during the final development phase. The tool is based on approaches from different areas of computer science: computer vision, data mining and cognitive modeling. Selected aspects of human capabilities can be mapped based on a cognitive model to simulate and analyze human behaviors. The modules of CogUA are based on computer vision and data mining for automated recognition of GUI elements and the identification of use cases respectively. The conceptual link of other modules of CogUA to the cognitive model is elaborated.

Keywords: automated usability analysis, user interaction simulation, cognitive modeling, ACT-R, computer vision

1. MOTIVATION

The manual control of modern machines for complex production processes is performed with software based user interfaces, exclusively or additionally to hardware buttons. The quality of the user interfaces contributes to an efficient and safe production process, cf. e.g. Flaspöler et al. (2009). However, the design concepts of user interfaces for special-purpose machines often do not explicitly consider optimal user interaction.

Industrial partners gave feedback that the focus of development lies on hardware, the software is often second-tier. The operational concept of GUIs follows in many cases the idea to present all possible functions of the machine and not to organize the workflow of the operator in the best possible manner.

The typical phases in the development of a special purpose-machine are shown in Fig. 1. Test runs of the machine carried out by an experienced machine operator reveal deficiencies of the user interface often too late at phase 3 or 4. Software modifications in the final development phases are typically avoided because it is time-consuming or risky to modify the software. This makes the development of easy to use and consistent software more difficult. But incorrect entries due to a poorly designed user interface can harm workpieces, machine or even workers. Therefore, the motivation is to develop a tool that allows automated usability evaluations of user interface concepts and prototypes at an early stage of the development.

2. CONCEPT

The use of a cognitive model to simulate user interactions linked to a user interface is at least in research a common practice, e.g. in gaming scenarios (Smart et al., 2016) or driving a car scenarios (Kujala and Salvucci, 2015). A motivation for applying a cognitive model in usability analysis is given in Ritter et al. (2001) and West and Emond (2002). Summarizing, a usability analysis based on a virtual user can supplement a standard usability evaluation. Advantageous is the full control of the setting (background of the subjects, the number of subjects,

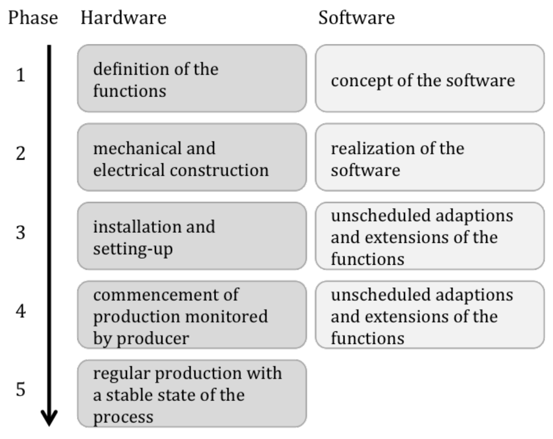

Phase
Hardware
Software
1
definition of the functions
concept of the software
2
mechanical and electrical construction
realization of the software
3
installation and setting-up
unscheduled adaptions and extensions of the functions
4
commencement of production monitored by producer
unscheduled adaptions and extensions of the functions
5
regular production with a stable state of the process

Fig. 1. Typical development phases of a special-purpose machine.

motivation validity, exclusion of an experimental bias e.g. by comments of the experimenter), disadvantageous are the technical limitations of the cognitive model albeit it is already complex and time-consuming to implement.

In the next section, a short overview of existing approaches of automated usability analysis with a link to a cognitive model is given, followed by the description of main features of the tool CogUA, which is under development. The section closes with the specification of its modules.

2.1 State of the art

Cognitive models aim at simulating processes of the human brain. Relevant processes for the mentioned application are perception (vision), motor functions (moving a hand, fingers) and decision making using pieces of knowledge stored in memory. There are different known realizations of a cognitive model, as EPIC (Executive Process Interactive Control, Kieras and Meyer (1997)), SOAR (States, Operators And Reasoning, Laird et al. (1987)) and ACT-R (Adaptive Control of Thought - Rational, Anderson et al. (2004)).

Different approaches of automated usability analysis in the past 15 years also have a similar motivation as sketched above, e.g. Misker et al. (2001); Ritter et al. (2001, 2002); John et al. (2004); Heinath and Urbas (2007); Halbrügge (2013); Quade et al. (2014); Russwinkel and Prezenski (2014). Especially, the integration of a cognitive model into usability analysis should be simplified to allow a practical application in daily work of a software developer of user interfaces. A framework of templates of common user interactions (like mouse click, keyboard stroke) linked to the cognitive model ACT-R is proposed in Salvucci and Lee (2003). A similar idea is realized in Heinath and Urbas (2007). The purpose is that the person who sets up the usability test does not require detailed knowledge of a cognitive model.

The CogTool (John et al., 2004) therefore provides a graphical user interface with the help of which a user can set up a visualization of the specific user interface he/she wants to analyze. Alternatively, Hyper Text Markup Language (HTML) code can be imported as the description of the user interface. Based on this visualization of the user interface a specific task consisting of different user interactions (mouse click, keyboard strokes) can be defined interactively with the CogTool GUI. Finally, the user runs the analysis based on the cognitive model. The outcome is a protocol of the predicted interactions including time stamps. The duration of interaction is a key criterion in the usability evaluation, cf. ISO 9241-210. The approach of Misker et al. (2001) goes one step further by avoiding the re-implementation of the user interface. On a Microsoft Windows system Misker et al. (2001) take the information about the user interface of an application from the window handlers. This information is transferred to the cognitive model. Actions, like a keyboard stroke, are returned from the cognitive model to the application. In this way, the cognitive model is directly linked to the user interface of the original application. The approach is evaluated for specifically defined use cases consisting of a sequence of interaction steps.

2.2 Main features of CogUA

The question arises why such virtual usability tests, as described above, are not common practice in the industry? The expectation is that the workload to define such a test is still too high. Therefore, we conceptually propose a tool - implemented already in parts - to automate as much of the process as possible.

The approaches cited above, especially the one of Misker et al. (2001) and CogTool, John et al. (2004), provide a good basis, as no or only litte knowledge of a cognitive model is required. But in the case of CogTool, it is still necessary to re-implement the user interface (except for web pages coded in HTML). In both approaches, each specific use case must be defined step by step to set up the test scenario. This definition of the use case requires manual work by analyzing the user interface in detail, e.g. by applying the Hierarchical Task Analysis, cf. Heinath and Urbas (2007).

CogUA aims at the reduction of the effort of defining the use cases manually. The idea in CogUA is to monitor user interactions and to derive interaction traces. These traces can then be analyzed to identify use cases. Additionally, the approach of Misker et al. (2001) is extended by automated analysis of the graphics of the user interface. Thereby, the information of the GUI elements is not limited by the restricted content of the window handler.

The recording of user interaction traces is applicable if an implementation of the user interface already exists, either (a) as prototype or (b) as running application, which should be modernized. In case of (a) the steps of a use case can be defined by executing them on the prototype user interface. A manual pre-analysis of the workflow or expert knowledge is here necessary in order to select appropriate use cases. In case of (b) the idea is that use cases are derived based on interaction traces recorded during the use of the software in practical work. The identified use cases are the basis to define the final use cases for the cognitive model. Additionally, the use cases can already be helpful for the design of a new version of the user interface with a better mapping of the workflow. Former workarounds, detected in the traces, can be designed as standard process.

The elements of the user interface are detected by automated recognition based on computer vision. As byproduct the graphics of the user interface can be analyzed regarding ergonomic criteria like contrast or font size.

The tool CogUA comprises four major steps:

Step 1: Analysis of the graphical user interface based on ergonomic criteria, e.g. contrast, font size,

Step 2: Recording of user interaction traces and identification of use cases,

Step 3: Simulation of possible interactions to check for consistency,

Step 4: Determination of interaction times for specific tasks based on cognitive models.

Concerning the identification of use cases, it is expected that this can not be fully automated. An expert of the specific application has to interpret the results of the automated analysis e.g. to filter out the relevant use cases.

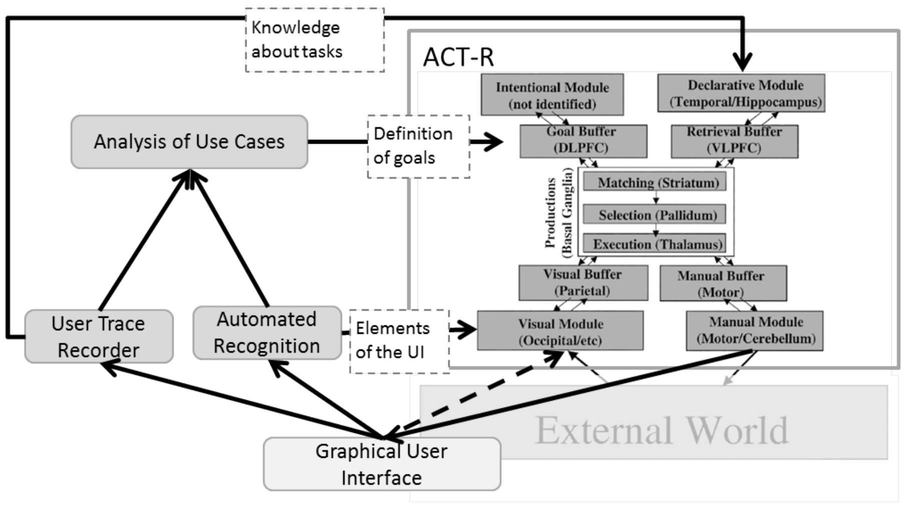


Fig. 2. A schematic diagram of CogUA linked to the architecture of ACT-R, cf. Anderson et al. (2004). The information which is transferred from the CogUA modules to ACT-R is depicted in the dashed boxes.

But the automated analysis should speed up the definition of the use cases compared to complete manual preparation.

2.3 Modules of CogUA

A schematic diagram of CogUA is shown in Fig. 2. Three modules are proposed as link to ACT-R:

Automated recognition: Based on screen shots of the user interface graphical elements such as buttons and text are automatically recognized. Hence, a little or no predefined knowledge of the software to be tested needs to be known. Other approaches require task models, source code or a representation of the user interface in HTML, e.g. Mahajan and Shneiderman (1997); John et al. (2004); Heinath and Urbas (2007); Quade et al. (2014). Technical details how the recognition is realized are described in Section 3.

Trace Recorder: Based on an event tracker user interactions (keyboard stroke, mouse clicks) are monitored and stored as a graph, cf. Section 4.

Analysis of Use Cases: Based on the trace graph use cases of user interactions are identified. A first approach is defined but not yet tested extensively. The concept is described in Section 5.

The cognitive model based on ACT-R is initialized considering the outcomes of these modules. The transferred information is depicted in the dashed boxes in Fig. 2. The recognized elements of the user interface can be transferred as visual objects detected by the visual module. The detailed traces are necessary to transfer knowledge about the interaction tasks into the declarative module. In the declarative memory, chunks are defined as pieces of memory. Results of the use case analysis can help to simplify the definition of goals within the goal buffer. The realization of this link is ongoing work.

3. AUTOMATED RECOGNITION

The basic objective of the automated recognition of a GUI obtained from the screen shots is to transform pixel-level input to objects and symbols, which a cognitive model can interpret. The responsibility of the automated recognition is to recognize the text embedded in the user interface and to detect the location of GUI elements such as buttons, drop down boxes etc. Existing algorithms of optical character recognition (OCR) are applied to minimize the amount of a priori information. Other approaches, e.g. Halbrügge (2013), require predefined templates of GUI elements.

3.1 Approach

The algorithm used to determine the embedded text and GUI elements is illustrated in Fig. 3. Initially, the acquired screen shot of the GUI is converted from RGB color space into gray scale Y according to Equation 1 as described in Gonzalez (2009).

$$\mathrm{Y} = 0.229\mathrm{R} + 0.587\mathrm{G} + 0.114\mathrm{B}. \tag{1}$$

Generally, binarization is performed on gray-scale images that are embedded with information. During the step, a threshold is chosen that separates the foreground and background information. In adaptive thresholding, the threshold value is determined based on the intensity values in the neighborhood. In order to identify GUI elements, the adaptive thresholded image is subjected to morphological operations (Gonzalez, 2009). Erosion and dilation are applied sequentially to determine horizontal patterns in the image. A similar operation is performed to determine vertical patterns in the image. Hence, the morphological operations generate two images, one with the vertical and horizontal structures in the image and the other with the text regions as shown in Fig. 3. Connected component analysis (Gonzalez, 2009) are performed on both of these images to obtain bounding boxes around the GUI elements and text regions. These image patches are passed on to the Tesseract-OCR (Smith, 2007) for text recognition. Tesseract-OCR returns the recognized text and a corresponding confidence score. The regions with a score less than an empirically determined threshold are neglected. The automated recognition is implemented using OpenCV python (Bradski, 2000).

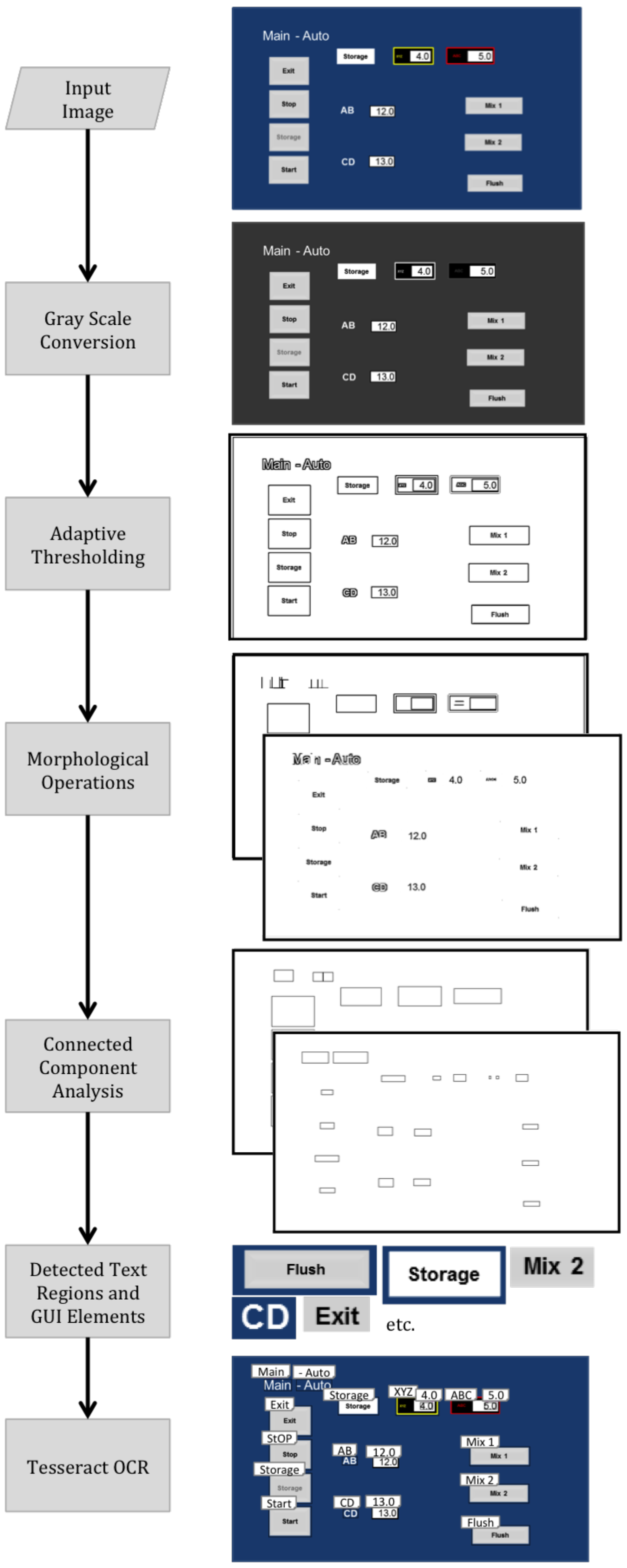


Fig. 3. The process steps of the automated recognition of text and GUI elements from the screen shot are shown on the left hand side. Results obtained after applying each step on a dummy user interface are shown on the right hand side.

3.2 Results

A sample result from the graphical analysis is listed in Table 1. The result is based on the dummy user interface, depicted in Fig. 4. The following properties of the recognized GUI elements and detected text are calculated:

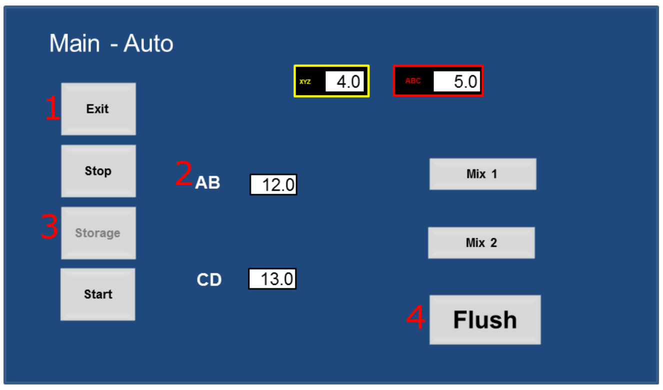


Fig. 4. Dummy user interface of height 507 pixels and width 879 pixels for which a sample of the results of the graphical analysis are tabulated in Table 1. For easy mapping, the respective ID is marked in red adjacent to recognized text or the GUI element.

Tab. 1. A sample of the results of the graphical analysis based on the dummy user interface shown in Fig. 4.

ID	Contrast	Font Size (pt)	Position (x, y)	Recognized Text
1	0.87	9	(91, 122)	Exit
2	0.74	13	(271, 247)	AB
3	0.35	9	(91, 288)	Storage
4	0.87	17	(590, 417)	Flush
. . .				

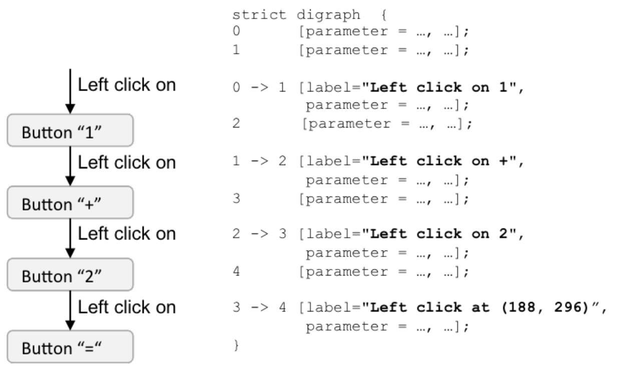
Contrast: Contrast c of the image region is calculated by taking the difference between the gray scale intensity of the foreground Y_f and of the background Y_b of the image region. The obtained value is divided by 255 to get a normalized value, yielding

$$c = |Y_f - Y_b|/255. \tag{2}$$

A high contrast value close to 1.0 is obtained if there is a large difference between the gray scale intensity of the foreground and background regions and vice versa. The contrast values in Table 1 range from 0.35 with a low contrast for the label "Storage", written in gray color, up to 0.87 for the labels "Exit" and "Flush", written in black color.

Position: The location of detected text or the GUI element is expressed as (x, y) using pixel indices. The top left corner of the image is considered as the origin, y value increases downward, while x increases to the right. The top left coordinate of the detected text or the GUI element is tabulated in Table 1.

Font size: The font size is expressed in terms of point (pt). The size of recognized text is obtained in number of pixels from the Tesseract-OCR based on the height of the bounding box of the text. This height value is transformed into pt considering the resolution of the specific display (94 px/inch). The derived font size is an approximation, which can be too small, if certain letters, as "g" with lower descender or "h" with a higher



```
strict digraph  {
0       [parameter = …, …];
1       [parameter = …, …];

0 -> 1 [label="Left click on 1",
        parameter = …, …];
2       [parameter = …, …];

1 -> 2 [label="Left click on +",
        parameter = …, …];
3       [parameter = …, …];

2 -> 3 [label="Left click on 2",
        parameter = …, …];
4       [parameter = …, …];

3 -> 4 [label="Left click at (188, 296)",
        parameter = …, …];
}
```

Fig. 5. Interaction trace for the use case to compute "1+2" with the Microsoft Windows calculator. On the right hand side the principle structure of the graph encoded in dot format is shown. As parameter font size, position, etc. can be stored.

ascender, are not included in the text label. A correction factor is planned to consider this.

Recognized text: The text recognized by the Tesseract-OCR is listed in Table 1. This information will be combined with the results of subsequent trace recorder steps to generate relevant and meaningful traces.

Contrast, text size and numbers of buttons can already give first hints in the evaluation of the usability based on ergonomic criteria as described e.g. in Mahajan and Shneiderman (1997). Also, consistency checks can be done based on the position and text information, e.g. that a button labeled with "Exit" remains at the same position on different "screens" of the user interface. Current problems in the automated text recognition depend on the contrast, resolution and the font type. Some letters are difficult to distinguish, like "i" and "l", or upper and lower case of letters are confused, e.g. "StOP" in Fig. 3. Special training of the text recognition predefining the text font can help to improve.

4. TRACE RECORDER

The trace recorder logs all the user interaction events such as mouse clicks and keyboard strokes happening on the graphical user interface. These monitored events along with information from the automated recognition step are combined and encoded as a graph. Graphviz (Ellson et al., 2001) library is used to display this graph of a sequence of monitored events encoded into dot format.

An example of such a graph is shown in Fig. 5. This graph was generated using the calculator application of Microsoft Windows, shown in Fig. 6. The task was to compute "1 + 2", as it could be seen looking at the content of the labels in the transition elements `0 -> 1`, `1 -> 2` and `2 -> 3` (`"Left click on ..."`). The last transition `3 -> 4` represents the left click on the equality sign which was not recognized in the text recognition process. Therefore, the coordinates of the click are mentioned in the label description. Sequences of such graphs are the input information for the use case analysis, described in the following section.

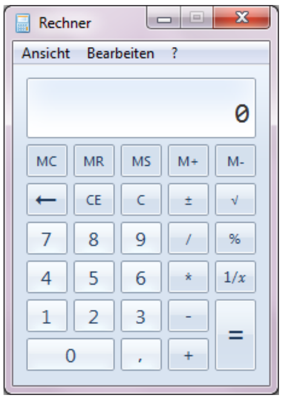


Fig. 6. Sample application: the Microsoft Windows calculator.

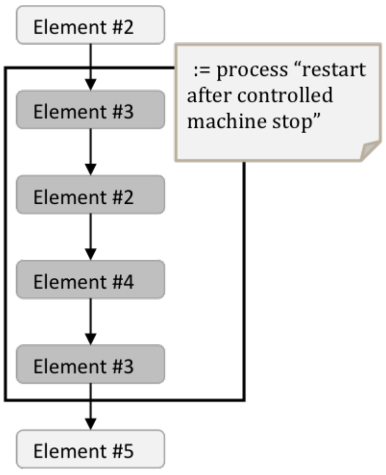


Fig. 7. Identification of a use case based on the user interaction traces.

5. USE CASE ANALYSIS

Besides the application of a cognitive model, the identification of use cases is a main feature of CoGUA. The idea is that based on the trace information, described in Section 4, repetitive user tasks, called use cases, can be filtered out. This is sketched as a simplified example in Fig. 7. A pattern of interactions steps $\{3, 2, 4, 3\}$, specified with integer numbers, has to be identified in the traces. For the example based on the Microsoft Windows calculator, shown in Fig. 5, the interaction sequence reads $\{0, 1, 2, 3, 4\}$, neglecting the specific action "left click" as it is the same for all elements. The algorithm, which is used to identify these patterns, is described in the following section. The testing of the algorithm based on real user interaction traces is ongoing work, therefore no experimental results can be shown yet. Instead, relevant aspects for the application of the algorithm in the context of CoGUA are discussed in Section 5.2.

5.1 Approach

A prototype implementation of the algorithm of sequential pattern mining is based on the work of Stroulia et al. (1999); El-Ramly et al. (2002a,b). The motivation for the original approach is reverse engineering of software tools as basis for the efficient migration of the software tool to

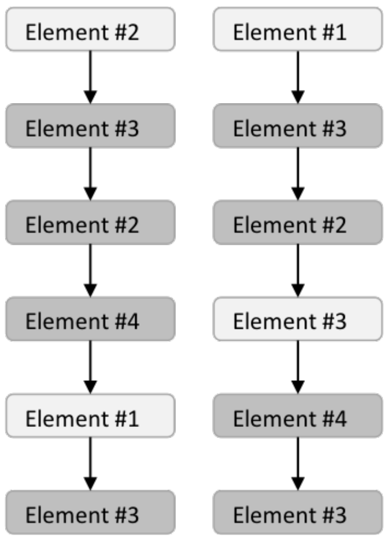


Fig. 8. Examples of two sequences s_1 and s_2 of trace elements. The use case - highlighted in dark grey - consisting of elements $\{3, 2, 4, 3\}$ is recognized considering a maximum error $\epsilon_{max} = 1$ by the sequential pattern mining algorithm.

new platforms or to optimize the GUI. By monitoring user interactions, a task model is derived. The task model can be used "as basis for specifying a new GUI tailored for the task in question" (Stroulia et al., 1999).

Based on the example given in El-Ramly et al. (2002b) the algorithm of sequential pattern mining is shortly explained. An interaction trace is summarized as an ordered sequence of ID numbers, e.g. a sequence $s_1 = \{1, 3, 2, 3, 4, 3\}$ and $s_2 = \{2, 3, 2, 4, 1, 3\}$. The task is to identify a repetitive pattern in these sequences, as sketched in Fig. 8. The selection of a matching part, called episode e, in the sequences depends on specified criteria:

minimum length ℓ_{min}: The pattern should have at least the specified minimum length.

minimum support σ_{min}: The minimum support defines the minimum number of occurrences of the pattern within the sequences.

maximum error ϵ_{max}: The maximum error defines the maximum number of not matching insertions within the episode, where the pattern matches a part of the sequence. For example, the pattern $\{1, 2, 3\}$ matches the episode $\{1, 2, 4, 3\}$ of a sequence with a maximum error of 1.

minimum score S_{min}: El-Ramly et al. (2002b) defines a score

$$S = \log_2(|p|)\log_2(n_\sigma(p))D(p). \tag{3}$$

Therein, p is the pattern with the length of $|p|$. The number n_σ is the number of occurrences of the pattern within the sequences, the support. The density D is defined as

$$D(p) = \frac{|p|n_\sigma}{\sum_{i=1}^{n_\sigma}|e_i|} \tag{4}$$

using the length $|e|$ of an episode e including eventually insertion errors.

Starting with all possible patterns of length 2 which can be build based on the set $\{1, 2, 3, 4\}$ a list of matching episodes can be derived considering the mentioned criteria (especially the minimum error). To avoid an exhaustive search of all possible patterns of all lengths the next longer patterns are generated based on those patterns, which fulfill the criteria, named above. The idea is that a longer pattern cannot meet the constraints unless its sub patterns meet them.

Assuming $\ell_{min} = 2$, $\sigma_{min} = 2$, $\epsilon_{max} = 1$, the longest pattern which can be found in the two sequences s_1 and s_2 is $p = \{3, 2, 4, 3\}$. It gets a score of $S = 1.6$ which is higher than the scores of shorter patterns fulfilling the criteria ($\{2, 3, 4\}$, $\{1, 3\}$,$\{3, 3\}$).

El-Ramly et al. (2002b) suggested that possible irrelevant noise can be neglected by using the run-length encoding algorithm, which replaces immediate repetitions with a count followed by the item being repeated. The sequence $\{1, 2, 3, 3, 3, 3, 2, 4, 4\}$ is for example then encoded as $\{1, 2, (4)3, 2, (2)4\}$. This procedure increases the chance to identify a larger number of patterns.

5.2 Aspects for the application within CogUA

The testing of the algorithm based on real user interaction traces is ongoing work. The approach has to be evaluated for the targeted application. The analysis of the interaction traces should provide primarily a basis for the composition of use cases, which will be deployed by the usability analysis based on the cognitive model. This aims at reducing the work effort applying a cognitive model for automated usability analysis, cf. Section 2.2.

Beyond this, the identified use cases can help to design future user interface concepts with much better workflow orientation. An additional but even more challenging goal is to identify inconsistencies in the workflow. That means that a specific task can be solved via the user interface following different interaction traces. This is sketched in Fig. 9 as workflow of the initialization of a fictitious process. If it is confirmed, that a parameter set A is ok, then the check of a second parameter set B pops up. This check is missing - in this simple, theoretic example - if the parameter set A had to be reconfigured.

In the industrial context, the software is often developed incrementally over the years and this increases the risk of generating such inconsistencies. Due to growing functional features of a system new modules have to be integrated into the user interface without changing the basic interaction concept. The analysis of the recorded traces can contribute to give hints for such inconsistencies by comparing traces of similar use cases. This simple check does not at all raise the claim of a complete proof of consistency. Such a proof requires complex analysis based on formal verification techniques as used in safety-critical application, cf. Wortelen et al. (2014). The identification of multiple interaction traces for the same task based on the simple comparison of traces can also help to find out the most efficient trace as best workflow.

However, the analysis of the interaction traces is a demanding task in the context of the targeted application area. In the case of machine operation, it is mostly not obvious what is the final goal of the user. The operator typically starts the interaction only if there is a problem,

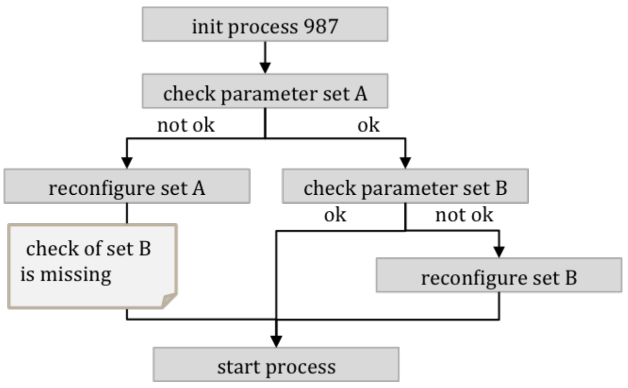


Fig. 9. A simple workflow of a fictitious process, where an inconsistency occurs in the traces. If parameter set A is reconfigured no check of parameter set B is initiated.

otherwise the machine runs in an automated process. The interaction intention contains therefore a phase of analyzing the problem (e.g. looking for sensor data) followed by a phase of solving the problem (e.g. correcting parameters). Therefore, test runs on real applications are essential as next step in the development of CogUA.

6. CONCLUSIONS AND OUTLOOK

Summarizing, the concept and first results of a tool for automated evaluation of graphical user interfaces are described. The development and testing of the tool is ongoing work. The application area is the operation of special-purpose machines, leading to specific challenges in order to build up a practice-oriented tool.

The next step will be extensive test runs on different user interfaces to evaluate in detail the first prototype implementation of the automated recognition, trace recorder and, especially the analysis of use cases. Subsequently, the link to the cognitive model has to be realized and tested. Comparisons between human users and the simulation are planned.

In perspective, the tool can help to improve the development of consistent and efficient user interfaces. This reduces the risk of software modifications during the final development phase. The application of the tool as a regular quality check within the development process can reinforce the link to existing design guidelines in practice. Even though the production systems become step by step completely automated, the user interface plays a critical role as long as humans have the ultimate control of the machines.

REFERENCES

John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036–1060, 2004.

Gary Bradski. The OpenCV library. *Doctor Dobbs Journal*, 25(11):120–126, 2000.

Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson. Mining system-user interaction traces for use case models. In *Proc of 10th International Workshop on Program Comprehension, 2002*, pages 21–29. IEEE, 2002a.

Mohammad El-Ramly, Eleni Stroulia, and Paul Sorenson. Recovering software requirements from system-user interaction traces. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 447–454. ACM, 2002b.

John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen C North, and Gordon Woodhull. Graphviz-open source graph drawing tools. In *Graph Drawing*, pages 483–484. Springer, 2001.

Eva Flaspöler et al. The human-machine interface as an emerging risk. Technical report, European Agency for Safety and Health at Work, 2009.

Rafael C Gonzalez. *Digital image processing.* Pearson Education India, 2009.

Marc Halbrügge. ACT-CV: Bridging the gap between cognitive models and the outer world. In E. Brandenburg, L. Doria, A. Gross, T. Günzlera, and H. Smieszek, editors, *Grundlagen und Anwendungen der Mensch-Maschine-Interaktion - 10. Berliner Werkstatt Mensch-Maschine-Systeme*, pages 205–210. Universitätsverlag der TU Berlin, 2013.

Marcus Heinath and Leon Urbas. Simplifying the development of cognitive models using pattern-based modeling. In *10th IFAC/IFIP/IFORS/IEA Symposium Analysis, Design, and Evaluation of Human-Machine Systems. Seoul, Korea.*, pages 130–135, 2007.

Bonnie E John, Konstantine Prevas, Dario D Salvucci, and Ken Koedinger. Predictive human performance modeling made easy. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 455–462. ACM, 2004.

David E Kieras and David E Meyer. An overview of the EPIC architecture for cognition and performance with application to human-computer interaction. *Human-computer interaction*, 12(4):391–438, 1997.

Tuomo Kujala and Dario D Salvucci. Modeling visual sampling on in-car displays: The challenge of predicting safety-critical lapses of control. *International Journal of Human-Computer Studies*, 79:66–78, 2015.

John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

Rohit Mahajan and Ben Shneiderman. Visual and textual consistency checking tools for graphical user interfaces. *IEEE Transactions on software engineering*, 23(11):722–735, 1997.

Jan Misker, Niels A Taatgen, and Jans Aasman. Validating a tool for simulating user interaction. In *Proceedings of the Fourth International Conference on Cognitive Modeling*, pages 163–168, 2001.

Michael Quade, Marc Halbrügge, Klaus-Peter Engelbrecht, Sahin Albayrak, and Sebastian Möller. Predicting task execution times by deriving enhanced cognitive models from user interface development models. In *Proc EICS 2014*, pages 139–148, 2014.

Frank E Ritter, Gorden D Baxter, Gary Jones, and Richard M Young. User interface evaluation: How cognitive models can help. *Human-computer interaction in the new millennium*, pages 125–147, 2001.

Frank E Ritter, Dirk Van Rooy, and Robert St Amant. A user modeling design tool based on a cognitive architecture for comparing interfaces. In *Computer-Aided Design of User Interfaces III*, pages 111–118. Springer, 2002.

Nele Russwinkel and Sabine Prezenski. ACT-R meets usability. In *Proc 6th International Conference on Advanced Cognitive Technologies and Applications COGNITIVE, 2014. Venice, Italy: IARIA*, 2014.

Dario D Salvucci and Frank J Lee. Simple cognitive modeling in a complex cognitive architecture. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272. ACM, 2003.

Paul R Smart, Tom Scutt, Katia Sycara, and Nigel Shadbolt. Integrating ACT-R cognitive models with the unity game engine. *Integrating Cognitive Architectures into Virtual Character Design. IGI Global, Hershey, Pennsylvania, USA*, 2016.

Ray Smith. An overview of the Tesseract OCR engine. In *icdar*, pages 629–633. IEEE, 2007.

Eleni Stroulia, Mohammad El-Ramly, Lanyan Kong, P Sorenson, and Bruce Matichuk. Reverse engineering legacy interfaces: An interaction-driven approach. In *Proc of 6th Working Conference on Reverse Engineering, 1999*, pages 292–302. IEEE, 1999.

Robert L West and Bruno Emond. Can cognitive modeling improve rapid prototyping. *Carleton University Cognitive Science Technical Report*, 5, 2002.

Bertram Wortelen, Andreas Lüdtke, Denis Javaux, and Sonja Sievi. Experiences from using formal verification techniques to analyze human-machine interaction: A case study. In *Proceedings of European Conference on Cognitive Ergonomics 2014*, 2014.