Fast-Time User Simulation for Dynamic HTML-based Interfaces

Marc Halbrügge (marc.halbruegge@tu-berlin.de)
Quality & Usability Lab, Telekom Innovation Laboratories
Technische Universität Berlin, Ernst-Reuter-Platz 7, 10587 Berlin

Keywords: Cognitive Modeling; Modeling Tools; ACT-R; HTML; POMDP.

Introduction

Using cognitive modeling to predict user behavior in the human-computer interaction domain is a promising field, but often hindered by practical problems. Especially the creation of a mock-up of the technical system under evaluation is often a tedious and time-consuming task. For the growing number of HTML-based applications, the modeling toolbox ACT-CV (Halbrügge, 2013) provides direct access to the user interface, removing the necessity to create a mock-up before the actual modeling can start. A down-side of this approach is that unaltered applications often cannot be used for fast-time simulation. This paper presents a new tool that solves this problem by capturing UI states and also the *control flow* of HTML applications and by transforming both into finite automata that can be used for fast-time simulation.

Related Work

The amount of work that needs to go into the creation of a link between a user interface (UI) and a cognitive architecture should not be underestimated, a fact that has been well put by Kieras and Santoro (2004, page 102): "Programming the simulated device is the actual practical bottleneck". There are solutions to this, but they often require instrumented applications (e.g., Urbas et al., 2006; Büttner, 2010). Other approaches like SegMan (Ritter, Van Rooy, St. Amant, & Simpson, 2006) allow unaltered applications, but work on a pixel-by-pixel basis, forcing the modeler to re-create the symbolic information from its lowest-level graphical equivalent.

ACT-CV

The cognitive modeling toolbox ACT-CV has started as a visual device for ACT-R (Anderson et al., 2004) that uses computer vision (hence ACT-*CV*) for the creation of symbolic representations of a visual scene from a video camera or a computer screen capture. HTML support was added in version 2 (Halbrügge, 2013), building ACT-R's visicon (visual icon) directly from the textual and clickable elements in the browser window and applying ACT-R's mouse clicks to them.

While direct access to real-world user interfaces eliminates the tedious need to create mock-ups, some inconveniences remain: Fast-time simulation is impossible, especially in the presence of graphical transitions and animations that take fixed amounts of time. It is also very hard to parallelize the model runs during batch processing as every model would need at least its own browser instance. In the case of stateful user interfaces, extra checks would be necessary to ensure that the model sessions do not interfere with each other.

Creating Finite Automata from Dynamic Web Pages

When we bring to our mind that ACT-R's visual module only uses the geometry, color, and (textual) content of a screen element, it should be obvious that creating and holding a complete browser instance to create this small amount of information is highly inefficient. How can we improve this?

The approach taken in this paper is to keep a history of the observed visicons in their reduced form, and taken together with the actions taken by the model, using this history to create a computational representation of the system that completely replaces the original browser content. As a cognitive model can only "see" through ACT-R's visicon, the removal of the actual browser is transparent to it.

Formalization The state $s \in S$ of the user interface is the set of currently visible elements as represented in ACT-R's visicon. The visicon is a table of chunks of type visual-object that contain their positions, dimensions, colors, and symbolic values. For reasons of simplicity, we consider every visicon entry as a possible action $a \in A$. An action is executed by interacting (i.e., clicking or tapping) with its corresponding UI element and usually leads to a new UI state. We are assuming discrete time, i.e., every transition to a new state denotes a new time step.

How can we represent the UI logic? If the UI had Markov properties, i.e., no hidden states, it could be captured by a deterministic transition function $\delta(S,A) \mapsto S$. This is usually not the case, though. Stateful interfaces are the rule rather than the exception; examples on the web are shopping carts, stored billing information, or personalized news and ads with the help of browser cookies.

In order to support hidden states, ACT-CV uses a deterministic finite partially observable Markov decision process (POMDP) representation for the UI logic. If a pair of state and action does not lead deterministically to another state, we go back in history until we can establish a deterministic transition function again, e.g., $\delta(S,A,S,A) \mapsto S$. Implementation-wise, we only store the part of the history that is needed to reproduce the observed behavior of the UI.[1] During the learning phase, the complete history needs to be kept in order to be able to adapt to newly discovered hidden states.

Exploration of the User Interface Finding an optimal solution to a POMDP problem is known to be NP-hard (i.e., not

[1] The flexible history approach presented here is not always optimal, e.g., because states with self-transitions can fill up the history without adding any information. A better, but more complex solution would be Looping Suffix Trees (Holmes & Isbell Jr, 2006).

easily computable for non-trivial cases, Kaelbling, Littman, & Moore, 1996). Fortunately, we are not in need of optimal solutions, often not even complete ones. The part of the state space that is actually visited by the cognitive model under investigation is most of the time much smaller than the complete state space of the application.

If the cognitive model is deterministic, we can build the UI representation from the observation history of a single complete model run. Non-deterministic models may visit much more states than deterministic ones. Due to this, we need to explore the application beforehand and independently of the model. ACT-CV procides a simple exploration mechanism that is inspired by the RMax algorithm (Brafman & Tennenholtz, 2003) for this situation.

Application Example and Results ACT-CV has been used during a modeling attempt targeted at a HTML-based home assistance system (Halbrügge & Engelbrecht, 2014). The work comprised an analysis of sensitivity towards global ACT-R parameters that needed hundreds of model runs. If this analysis had to be done in real-time, it would have needed several weeks to complete. With the help of ACT-CV and the state machine approach introduced above, the analysis could be run approximately 150 times faster on a commonplace desktop computer and finished within a few hours.

Open Issues While the automaton approach has worked very well in the example given above, issues remain. First, graphical transitions like fade-ins make it hard to determine when human users can actually see and act upon different elements on the screen. As ACT-CV does not capture the fade-in, some timing information may get lost. Secondly, page scrolling behavior has not yet been modeled using the automaton approach. Scrolling can lead to many different UI states and may need extra treatment. And finally, if the UI allows data entry, e.g., in text fields, this instantly blows up the amount of possible states of the application. Free exploration cannot be used in this case.

Conclusion

Especially when using cognitive modeling to capture rare events like errors, it is often necessary to execute many model runs. In this case, it is very important to be able to run the model in fast-time simulation. In the case of HTML-based interfaces, this is only possible if a non-HTML mock-up is available. While some modeling tools (e.g., CogTool, John, Prevas, Salvucci, & Koedinger, 2004) have import functions for static HTML content, dynamic applications were not yet covered. ACT-CV closes this gap by allowing to extract the information on the screen that is needed by ACT-R's visual module from the rendered browser content. The application logic is transformed into a finite state machine by the means of guided or free exploration using a POMDP approach.

ACT-CV is freely available for download at http://act-cv.sourceforge.net.

Acknowledgements

The author gratefully acknowledges financial support from the German Research Foundation (DFG) for project "Automatische Usability-Evaluierung modellbasierter Interaktionssysteme für Ambient Assisted Living" (AL-561/13-1).

References

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological review*, *111*(4), 1036–1060.

Brafman, R. I., & Tennenholtz, M. (2003). R-MAX: A general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, *3*, 213–231.

Büttner, P. (2010). Hello Java! Linking ACT-R 6 with a Java simulation. In D. D. Salvucci & G. Gunzelmann (Eds.), *Proceedings of the 10th international conference on cognitive modeling* (pp. 289–290).

Halbrügge, M. (2013). ACT-CV: Bridging the gap between cognitive models and the outer world. In E. Brandenburg, L. Doria, A. Gross, T. Günzlera, & H. Smieszek (Eds.), *Grundlagen und anwendungen der mensch-maschine-interaktion* (pp. 205–210). Berlin: Universitätsverlag der TU Berlin.

Halbrügge, M., & Engelbrecht, K.-P. (2014). An activation-based model of execution delays of specific task steps. *Cognitive Processing*, *15*, S107-S110.

Holmes, M. P., & Isbell Jr, C. L. (2006). Looping suffix tree-based inference of partially observable hidden state. In *Proceedings of the 23rd international conference on machine learning* (pp. 409–416).

John, B. E., Prevas, K., Salvucci, D. D., & Koedinger, K. (2004). Predictive human performance modeling made easy. In *CHI '04: Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 455–462). New York, USA: ACM Press.

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Kieras, D. E., & Santoro, T. P. (2004). Computational GOMS modeling of a complex team task: Lessons learned. In *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 97–104).

Ritter, F. E., Van Rooy, D., St. Amant, R., & Simpson, K. (2006). Providing user models direct access to interfaces: an exploratory study of a simple interface with implications for HRI and HCI. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, *36*(3), 592–601.

Urbas, L., Heinath, M., Trösterer, S., Pape, N., Dzaack, J., Kiefer, J., & Leuchter, S. (2006). AGImap: A tool-chain to support the modeling of the interaction level of dynamic systems. In *Proceedings of the 7th international conference on cognitive modeling* (p. 409). Trieste: Edizioni Goliardiche.