Automatic Generation of Full-Control System Abstraction for Human-Machine Interaction

Sébastien Combéfis
Computer Science and Engineering Dept.
Université catholique de Louvain
Place Sainte Barbe, 2
1348 Louvain-la-Neuve, Belgium
Sebastien.Combefis@uclouvain.be

Charles Pecheur
Computer Science and Engineering Dept.
Université catholique de Louvain
Place Sainte Barbe, 2
1348 Louvain-la-Neuve, Belgium
Charles.Pecheur@uclouvain.be

ORIGIN AND UNDERLYING PRINCIPLES

Automated systems are increasingly complex, making it hard to design interfaces for human operators. Human-machine interaction (HMI) errors like automation surprises are more likely to appear and lead to system failures or accidents as testified by several cases detailed in the literature [9, 13, 16]. Researchers in psychology, human factors and ergonomics have been working on HMI issues for several years. Since the mid-1980s, researchers are investigating the use of formal methods to analyse behavioural aspects of HMI. Initially focused on the analysis of specific situations and on the system and it properties [17, 3], the field moved to more generic results based on theories like graph theory, model-checking or theorem proving [19, 2, 8].

Different questions might be asked in the analysis of HMI. The first kind of problem is the *verification* of some properties such as: *"May a system exhibit potential mode confusion for its operator?"* or *"No matter in which state the machine is, can the operator always drive the machine into some recover state?"*. Another kind of problem is the *generation* of some elements that help in a correct interaction, such as user's manuals, procedures and recovery sequences or user interfaces.

Recently, Degani et al. [11] pioneered a new approach consisting in the automatic generation of a user mental model for a system model described as a statechart. In this context, a mental model is not meant to capture a human cognitive model; rather, it is meant to capture the implicit and intended model of operation according to which the system developer designs the system.

The work we are pursuing follows the work of Degani et al. by defining formally the problem of automatic generation of a user mental model satisfying properties which allows a perfect user who follows that model exactly to operate the system without being surprised during the interaction. The definition of these properties, which we call *full-control* [7], and the development of corresponding verification and generation algorithms, is the core of our work.

This paper describes the proposed approach to formally analyse HMI. The remainder of the paper is organised as follows. The first section draws up the motivation of this work and poses the context and the problem that is tackled. The second section presents our techniques to generate automatically system abstractions. The next section presents briefly the prototype tool that has been developed. The fourth section discusses the ongoing work and perspectives for the proposed approach and finally the conclusion sums up our contribution.

MODELLED RELATIONSHIPS

Automatic generation of mental models needs to be driven by the intended characteristics of the resulting models. The *full-control* property [7] formalizes the following notion of a correct mental model: a user following a full-control mental model will know at any point how to command or observe the system to achieve a goal, based on the history of previous commands and observations performed. The models are formally represented as enriched labelled transition systems (LTS) where a distinction is made between the actions [12]. *Commands* are executed by the user on the system (inputs) and *observations* are controlled by the system and just observed by the operator (outputs). *Internal actions* are purely internal to the system, not observable by the user at all. All those aspects are detailed in [7]. In more recent work, we are also considering additional state-based observation and we show that those new enriched models can be translated into the initial framework. Figure 1 shows the graphical representation of a system model of a vehicle transmission system example coming from [11].

Generating a minimal full-control mental model from a given system model helps to get a better understanding of the system. The full-control property captures the knowledge an operator needs to have about a system to be able to control it properly. Such a mental model can be used to build training materials such as user manuals [18]. Providing a system that the user can learn, minimizing her memory load, and allowing her to operate it without error is a desirable usability property [15].

PROBLEMS ADDRESSED

This work proposes the full-control property to highlight an aspect of a good system abstraction which will ensure good

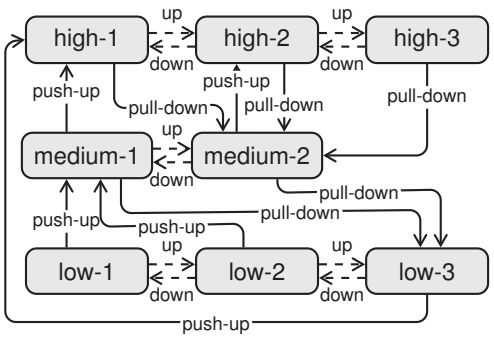

Figure 1. The vehicle transmission system example.

human-machine interaction. Algorithms to check that property and to generate minimal full-control system abstractions have been developed, based on a reduction approach [7] and also on a learning approach [5]. Also, an analysis methodology and an associated framework for using such algorithms in a practical setting to support the design and analysis of HMI systems are proposed in [4, 6]. The proposed framework can be used for modelling HMI systems and analyzing models against HMI vulnerabilities. The analysis can be used for validation purposes or for generating artifacts such as mental models, manuals and recovery procedures; it can also be used to help redesign or update a system model to avoid detected vulnerabilities.

The core contribution of this work is the automatic generation of minimal full-control system abstraction given a system model. As introduced in the previous section, the full-control property ensures that a user following a user mental model satisfying the property will always keep control of the system and furthermore will be able to execute all the possible interactions with the system. That is, if at any time during the interaction, a command can be executed on the system, the user will know it. Moreover if an observation occurs, the user will not be surprised as he will expect it according to his user mental model.

Two algorithms were developed in [7, 5], which are focused on the automatic generation of a minimal full-control mental model for a given system. The first is based on the definition of a bisimulation-based relation between the states of the system, stating which of them can be merged together because they can be handled similarly from the standpoint of the operator. The second uses a learning algorithm which iteratively builds mental model guesses. The algorithm relies on a teacher to answer whether proposed execution sequences must, may or cannot be part of the mental model. The teacher uses the system model to answer such queries.

Figure 2 shows the minimal full-control system abstraction for the vehicle transmission system example. As illustrated, the operator does not need to know the difference between the three high states of the system, and between the two medium states. For the low states, the operator must distinct the three states in order to be able to control the system (according to the full-control property). In practice, it means that the operator must pay attention to the up and down observations in order to control the system.

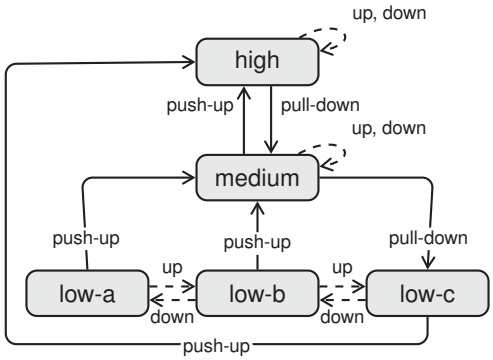

Figure 2. The minimal full-control mental model for the vehicle transmission system.

APPLICATIONS

A prototype tool has been implemented in Java. The prototype is based on the *Java Pathfinder* (JPF) model-checker [1]. This section briefly presents the tool.

The first part of the framework aims at providing tools for encoding models using statecharts [10], a widespread graphical notation to model systems. The statecharts can be designed in any existing tool which supports export in XMI file, e.g. ArgoUML [1]. The statechart is converted into a Java program encoding it, following the conventions of the JPF statecharts extension [14]. With that extension, the resulting Java program can be explored and used by the JPF model checker, for example to check temporal logic properties. Finally, the framework uses JPF with the JPF statecharts extension to explore all the transitions of the complete behaviour of the system and builds the full expanded LTS.

The second part of the framework consists of the analysis and mental model generation part. It is possible to check whether a mental model allows full-control of a given system. The tool takes two LTSs as input (a system and a mental model) and outputs true if the mental model allows full-control of the system and false otherwise. It is also possible to generate a minimal full-control mental model given a system model as input (provided such a model exists). Both algorithms from [7, 5] (reduction and learning) can be used. The tool produces an LTS corresponding to one minimal full-control mental model or says that no such model exists providing a problematic sequence from the system.

The benefit of using JPF is that it is a versatile model checker. It can therefore be used to perform additional types of analysis on the statechart model, for example application-specific safety properties as supported by the JPF framework.

The generation of minimal full-control system abstraction can be used in several phases of the design process. During the design of the system, the approach can be used to control if the system could be controlled through the existence of a minimal full-control abstraction. The system abstraction can also reveal clues about the system complexity as a mental model

[1] `http://argouml.tigris.org/`.

should not be ideally too large to fit in the human memory. One application of the proposed techniques is to help in the design phase so that system are designed in a way to ensure the possibility for an operator to control it without being surprised during the interaction.

LIMITATIONS AND DEVELOPMENT OPPORTUNITIES

Most system models used in the literature includes state-based observations. In the current framework, labelled transition systems are used which means that all the information is on the transitions. We are currently working on more general models where there is also information on the states. The first results tend to prove that this new problem can be translated in the current setting.

Full-control property may be too strong for some kind of analyses since it forces all the commands that are available on the system to be present in the user mental model. In some particular situations, what is interesting for the operator is to be able to only control a certain subset of the full behaviour of the system. We are currently exploring a variant of the full-control property where the user is not required to know exactly all the possible commands of the system but only those who are of interest to the user, for example described in a user task model.

This work describes a formal framework for the analysis of human-machine interactions, with a focus on controllability aspects of the system based on a distinction between commands and observations. The analysis is based on a formal characterization of an adequate control of the system by the user. That characterization, captured by the full-control property, is used as a validation criterion for system models during the design process cycle. The full-control property is a desirable property since it helps to prevent the operator from being surprised when interacting with a system. Two algorithms, one based on a reduction approach and one based on a learning approach, have been proposed. The framework has been implemented in Java within the JPF model checker environment.

REFERENCES

1. JavaPathfinder (JPF). http://babelfish.arc.nasa.gov/trac/jpf/.
2. Campos, J. C., and Harrison, M. D. Model checking interactor specifications. *Automated Software Engineering 8*, 3–4 (2001), 275–310.
3. Campos, J. C., and Harrison, M. D. Systematic analysis of control panel interfaces using formal tools. In *Proceedings of the 15th International Workshop on the Design, Verification and Specification of Interactive Systems*, no. 5136 in Lecture Notes in Computer Science, Springer-Verlag (July 2008), 72–85.
4. Combéfis, S., Giannakopoulou, D., Pecheur, C., and Feary, M. A formal framework for design and analysis of human-machine interaction. In *Proceedings of the 2011 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2011)*, IEEE (Oct. 2011), 1801–1808.
5. Combéfis, S., Giannakopoulou, D., Pecheur, C., and Feary, M. Learning system abstractions for human operators. In *Proceedings of the 2011 International Workshop on Machine Learning Technologies in Software Engineering (MALETS 2011)*, ACM (New York, NY, USA, Nov. 2011), 3–10.
6. Combéfis, S., Giannakopoulou, D., Pecheur, C., and Mehlitz, P. A JavaPathfinder extension to analyse human-machine interactions. In *Proceedings of the Java Pathfinder Workshop 2011* (Nov. 2011).
7. Combéfis, S., and Pecheur, C. A bisimulation-based approach to the analysis of human-computer interaction. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2009)*, G. Calvary, T. N. Graham, and P. Gray, Eds., ACM (New York, NY, USA, July 2009), 101–110.
8. Curzon, P., Rukšėnas, R., and Blandford, A. An approach to formal verification of human-computer interaction. *Formal Aspects of Computing 19*, 4 (Nov. 2007), 513–550.
9. Degani, A. *Taming HAL: Designing Interfaces Beyond 2001*. Palgrave Macmillan, Jan. 2004.
10. Harel, D. Statecharts: A visual formalism for complex systems. *Science of Computer Programming 8* (June 1987), 231–274.
11. Heymann, M., and Degani, A. Formal analysis and automatic generation of user interfaces: Approach, methodology, and an algorithm. *Human Factors: The Journal of the Human Factors and Ergonomics Society 49*, 2 (Apr. 2007), 311–330.
12. Javaux, D. A method for predicting errors when interacting with finite state systems. How implicit learning shapes the user's knowledge of a system. *Reliability Engineering and System Safety 75* (Feb. 2002), 147–165.
13. Leveson, N. G., and Turner, C. S. Investigation of the therac-25 accidents. *IEEE Computer 26*, 7 (July 1993), 18–41.
14. Mehlitz, P. C. Trust your model - verifying aerospace system models with JavaPathfinder. In *Aerospace Conference, 2008 IEEE* (Mar. 2008), 1–11.
15. Nielsen, J. The usability engineering life cycle. *Computer 25* (Mar. 1992), 12–22.
16. Palmer, E. Oops, it didn't arm. — a case study of two automation surprises. In *Proceedings of the 8th International Symposium on Aviation Psychology* (1996), 227–232.
17. Rushby, J. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety 75*, 2 (Feb. 2002), 167–177.
18. Thimbleby, H. Creating user manuals for using in collaborative design. In *Proceedings of the Conference Companion on Human Factors in Computing Systems*, ACM (New York, NY, USA, 1996), 279–280.
19. Thimbleby, H., and Gow, J. Applying graph theory to interaction design. In *Engineering Interactive Systems 2007/DSVIS 2007*, J. Gulliksen, Ed., no. 4940 in Lecture Notes in Computer Science, Springer-Verlag (2008), 501–518.