An Approach to Generating Human-Computer Interfaces from Task Models*

Matthew L. Bolton
Industrial and Systems Engineering
The State University of New York at Buffalo
Buffalo, NY 14260

Samaneh Ebrahimi
Mechanical and Industrial Engineering
University of Illinois at Chicago
Chicago, Illinois 60607

Abstract

Human-computer interaction (HCI) can often occur in situations unanticipated by designers. As such, usability may not always be maintained or human operators may not be able to achieve task goals. This can result in poor system adoption, decreased productivity, and unsafe operating conditions. Formal methods can be used prove properties about different elements of HCI. Unfortunately, these methods require the creation of formal models of interface designs, a non-standard practice that is prone to modeling error. This work describes an approach (based on L* learning) that is being developed to automatically generate formal human-computer interface designs guaranteed to always adhere to usability properties and support human operator tasks. The paper presents a formulation of this approach and describes plans for its implementation. These are discussed along with avenues for future research.

When designing a human-computer interface, it is critical that the design have good usability and allow the human operator to always fulfill his or her task goals. A failure to do this could result in poor acceptance of the system, reduce productivity, or lead to unsafe operating conditions. Unfortunately, the concurrent nature of human-computer interaction (HCI) can result in designers missing poor usability conditions or situation that could prevent operators from performing their tasks. Luckily, researchers have developed techniques that can help designers address these issues.

User-centered design is an approach for creating human interactive systems that emphasizes the need to facilitate the human operator's tasks (DIS, ISO 2009). Tasks are documented using task analyses (Schraagen, Chipman, and Shalin 2000; Kirwan and Ainsworth 1992), where the purpose is to capture the space of possible behaviors that a human operator can or should use to achieve goals with a system. The product of a task analysis is a task model that describes the ordinal relationships between the different activities and actions human operators can perform. It also contains strategic knowledge (usually as logical conditions) to specify when activities and actions can execute and what they must accomplish. The products of task analyses and user-centered designs can also be used with formal methods.

Formal methods are robust mathematical tools and techniques that give analysts means of guaranteeing that a system model satisfies certain properties (Wing 1990). Formal verification have been used to formally model task analytic and human-computer interface behaviors and prove usability and safety properties about them (Bolton, Bass, and Siminiceanu 2013; Campos and Harrison 1997). The vast majority of these analyses use formal verification, where proof techniques show that a system model adheres to formal specification properties. Model checking is an approach to formal verification where a model of a system's behavior is automatically proven to either satisfy or not satisfy logically formulated specification properties (Clarke and Wing 1996).

While powerful, such analyses require the creation of formal human-computer interface models, a process that can be difficult and prone to modeling error (Heitmeyer 1998). Further, such an approach, with its emphasis on interface modeling, can ignore the human operator tasks and thus not be compatible with user-centered design.

A variety of work has investigated how to generate human-computer interfaces from object-oriented design models, including task models (Tran et al. 2010; García et al. 2008; Mahfoudhi, Abid, and Abed 2005; España, Pederiva, and Panach 2007). However, such methods are predominantly concerned with rapid software development and not providing formal guarantees about the produced design.

Thus, there is a real need for formal approaches that will enable designers to automatically develop interfaces guaranteed to allow human operators to usably perform their tasks.

Researcher have shown that the L* learning algorithm (Angluin 1987) can be used to generate interfaces that adhere to usability properties (Combéfis et al. 2011; Combéfis and Pecheur 2012). However, these designs are created from models of system automation and are thus not user-centered and do not provide guaranteed support for human tasks.

Objectives

In this paper, we describe preliminary steps we have taken to develop a user-centered interface design method. Our approach uses formal human task behavior models, formal usability properties, and L* learning to automatically generate formal interface designs. Given the use of task models

*This material is based upon work supported by the National Science Foundation under Grant No. IIS-1353019.

and usability properties in the generation, the designs will be guaranteed to always usably support the human operator's task, thus supporting user-centered design.

In the material that follows, we present the background necessary for understanding our approach. We then describe the approach itself. This is followed by an examination of implementation possibilities. Finally, the method is discussed and avenues of future research are explored.

Background

There is a significant literature covering the way that formal methods can be used in the analysis of HCI (Bolton, Bass, and Siminiceanu 2013). For this work, formal human-computer interface modeling, usability specification, formal human task behavior modeling, and L* learning are most relevant. All are discussed below.

Formal Models of Human-computer Interfaces

To be formally evaluated, the behavior of the human computer interfaces must be modeled. This can occur using many notations including statecharts (Degani and Heymann 2002), interactors (Harrison and Duke 1995), and Object-Z (Bowen and Reeves 2008). Despite the diversity of techniques, all generally treat the interface as a finite state automaton (FSA) that can transition between states based on human operator actions or other system conditions (Parnas 1969).

This work is relevant to this project because it shows that human-computer interfaces can be formally modeled and thus reasoned about using formal verification tools.

Formal Interface Usability Properties

Formal verification can be used to evaluate the usability of modeled interfaces using formal usability specification properties. Campos and Harrison (1997) identified four categories of such properties for which generic specification patterns have been formulated. Reachability properties assert qualities about whether and when interface conditions can be attained (Campos and Harrison 2009; Abowd, Wang, and Monk 1995; Paternò 1997). Visibility properties describe how feedback to human actions should manifest (Paternò 1997; Campos and Harrison 2008). Task-related properties specify that task goals should be achievable and that actions can be undone (Paternò 1997; Bolton et al. 2013; Abowd, Wang, and Monk 1995; Campos and Harrison 2008; Bolton 2013). Reliability describes properties important to safety such as behavioral consistency, the absence of deadlock, and the lack of hidden mode changes (Campos and Harrison 2009; Abowd, Wang, and Monk 1995; Joshi, Miller, and Heimdahl 2003).

Formal Models of Human Task Behavior

Methods have additionally been developed that allow models of human task behavior to be model formally and thus used in formal analyses. In these, a task model is either represented using a formalism or translated into one where it interacts with other formally modeled elements of a target system. Formal verification is then used to prove that the system is safe or free from deadlock (Basnyat et al. 2007; Gunter et al. 2009; Campos 2003; Aït-Ameur, Baron, and Girard 2003; Bolton and Bass 2010; 2009; Bolton, Siminiceanu, and Bass 2011; Paternò and Santoro 2001; Palanque and Paternò 1997; Navarre et al. 2001; Barboni et al. 2010) or that it exhibits other desirable usability properties (Bolton et al. 2013; Bolton 2013).

The L* Learning Algorithm

The L* algorithm is a machine learning process capable of generating a formal model based on a series of queries to a teacher and/or oracle (Angluin 1987). Specifically, an L* algorithm will learn a minimal FSA for accepting a language. It does this by iteratively generating queries that it must receive specific answers to. The L* algorithm will generate two different types of queries. In the first type, it creates a string representing a word that may or may not be in the target language. The algorithm's teacher must supply an answer to the algorithm indicating if the string can be accepted, rejected, or if it doesn't matter ("Don't Care") (Angluin 1987; Giannakopoulou and Păsăreanu 2009; Chen et al. 2009). After a number of such queries, the L* algorithm will have enough information to generate a FSA. It will send this to an oracle. It is then the job of the oracle to examine the automaton and determine if it is satisfactory. If it is, then the algorithm completes. If it is not, the oracle must generate a string (counterexample) it considers to be unacceptable that can be accepted by the FSA created by the L* algorithm. The oracle will continue generating strings and candidate automata until a satisfactory automata has been found.

L* learning is pertinent to this project because it gives us the ability to learn a FSA, thus enabling the automatic generation of human-computer interface models.

Using L* to Generate Interface Designs

L* learning has been used to generate human-computer interface designs (Combéfis et al. 2011; Combéfis and Pecheur 2012). In this approach, the L* algorithm learns a formal interface design as a FSA. The alphabet associated with the FSA includes the human actions and other system conditions that can alter interface state. The interface's state represents an abstraction of the underlying automation's state. Using this representation, Combéfis et al. (2011; 2012) have shown that L* learning can generate a minimal interface design from a model of system automation while ensuring that the human operator retains "full control" (a condition that will nominally prevent mode confusion; Combéfis and Pecheur 2009). While this work did not consider human task behavior, it does demonstrate that L* learning can generate interfaces that adhere to certain usability properties.

Interface Generation Approach

In this work, we synthesizes each of these concepts into a single novel approach that use L* learning to automatically generate formal human-computer interface designs from formal task analytic behaviors models and usability properties. Given the nature of L* learning, this approach should guarantee that the produced interface design will always usably support the human operator's task.

We assume the same formal interface representation used by Combéfis et al. (2011; 2012), where a learned interface design is a FSA that transitions between interface states based on human actions and other system conditions. However, instead of using a model of system automation as input, our approach takes a task analytic behavior model (which describes human normative interaction with the system) and a set of desirable formal usability properties as inputs. The method extracts the "alphabet" of the interface from the actions and strategic knowledge in the task model. Then, an L* algorithm learner will progressively learn a minimal interface FSA by generating queries and design candidates that are evaluated by a teacher and oracle (respectively). The teacher and oracle will compare the outputs created by the learner to the task model and the usability properties using formal verification techniques. These will produce the responses that will tell the learner if the produced queries and designs are or are not acceptable. When complete, the L* algorithm will produce a formal human-computer interface design that will support the human operator'is tasks and satisfy the included usability properties.

A graphical representation of the approach can be seen in Figure 1. It proceeds as follows:

A.1 A task analytic behavior model is evaluated by an alphabet extraction process that identifies the human actions and system conditions (found in task analytic behavior model strategic knowledge) that will constitute the alphabet of the formal interface design's FSA representation. This is sent to the L* learner.

A.2 The L* learner uses the alphabet to generate a unique execution sequence (a string of actions and system conditions) that it sends to the teacher.

A.3 The teacher process compares the string to the original task analytic behavior model to see if the execution sequence is compatible with its contained behavior. If it is, the teacher produces an "Accept" response. If not, it produces a "Reject". If the string is incomplete (doesn't constitute a full execution sequence through the task model) it issues a "Don't Care" response.

A.4 The response is observed by the L* learner. Then, if the L* learner has enough information to produce a design, it will go to step A.5. Otherwise, it will go back to step A.2 to produce a new string.

A.5 The learner creates a formal interface design (a FSA) that is capable of accepting and/or rejecting all of the previously considered execution sequence strings (respectively accepted or rejected by the teacher). This is examined by the oracle that checks it to ensure that it is compatible with the provided task analytic behavior model and the included formal usability properties. If the interface design passes these evaluations, then the candidate design is output as the final design. Otherwise, the oracle produces an unacceptable execution sequence that is allowed by the current interface design (a counterexample). This is examined by the learner who then proceeds at step A.2.

The oracle process (in step A.5) produces its results through a sequence of model checking verifications (see Figure 2). This is a multistep process that proceeds as follows:

O.1 The oracle translates the task analytic behavior model and the produced interface design into a larger formal system model. Concurrently, it also produces task-related usability specifications that assert properties about the proper execution of the task when interacting with the interface.

O.2 The formal system model is iteratively checked against these properties using a model checker. If all of these are satisfied, then the process proceeds to the next step. Otherwise, any produced counterexample is output.

O.3 The original formal interface design is iteratively checked against all of the formal usability properties included in the analysis. If a counterexample is ever produced, it is output. If no counterexample is produced, then the candidate design is output as the final design.

Implementation Plans

To realize the interface generation approach presented, there are three major components that will need to be identified or developed: (a) A suitable task analytic modeling system needs to be selected; (b) An L* learning algorithm needs to be implemented and/or a suitable existing implementation needs to be found; and (c) A formal evaluation framework for implementing the teacher and model checking component of the oracle needs to be chosen. Although this project is still in its early stages, we have identified some prime candidates for each of these. All three are discussed below.

Task Modeling

There are many different task analytic modeling techniques. However, the formal task modeling notation that is most relevant to this work is the Enhanced Operator Function Model (EOFM) (Bolton, Siminiceanu, and Bass 2011).

EOFM is an XML-based task modeling language specifically designed to allow task analytic human behavior to be included in formal analyses. EOFMs represent human behavior as a hierarchy of goal driven activities that decompose into lower level activities, and finally, atomic actions. A decomposition operator specifies the temporal and cardinal relationships between the decomposed activities or actions (when they can execute relative to each other and how many can execute). EOFMs express strategic knowledge explicitly as conditions on activities. They can assert what must be true before an activity can execute (preconditions), when it can repeat (repeat conditions), and what must be true when it finishes (completion conditions). Most importantly, EOFM has formal semantics which specify how an instantiated EOFM model executes. This enables its use with formal methods.

EOFM is relevant for this work because it can formally represent normative human task behaviors and can thus be included in formal verification analyses (Bolton and Bass 2010; Bolton, Siminiceanu, and Bass 2011; Bolton and Bass 2012; Bolton, Bass, and Siminiceanu 2012; Bolton and Bass 2013b; 2013a). Thus it can be used by the proposed oracle

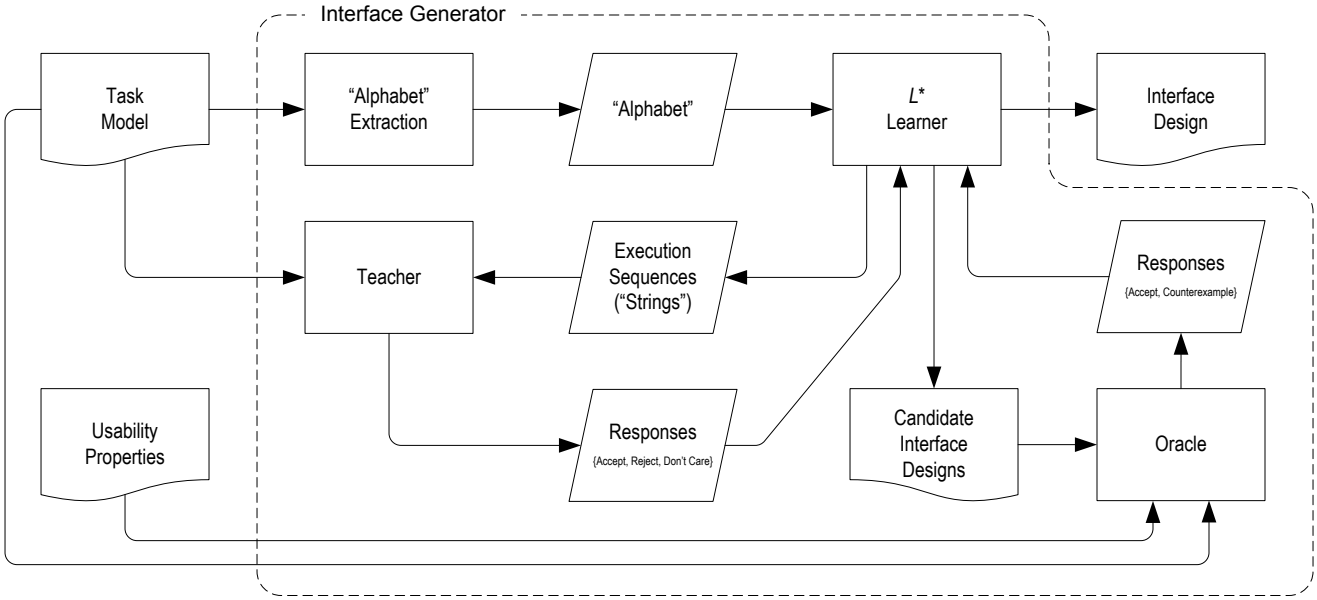

Figure 1: The approach for automatically generating human-computer interfaces from task models and usability properties. Note that a description of the inner workings of the oracle can be found in Figure 2.

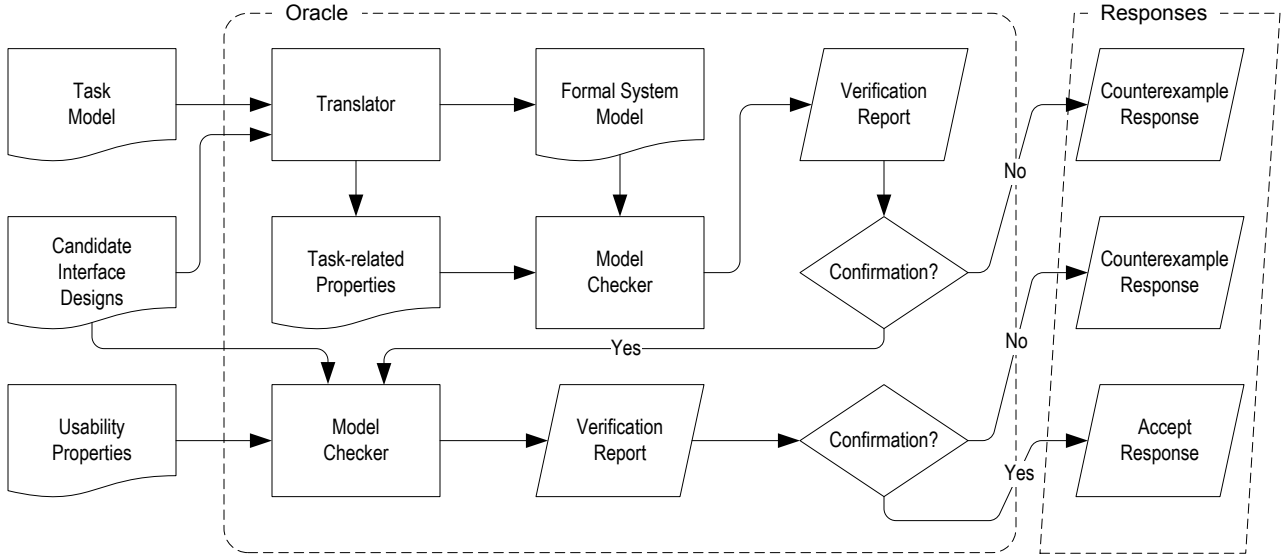

Figure 2: The approach used by the oracle (see Figure 1) to evaluate an interface designs produced by the L* learner.

and teaching procedures to evaluate learner outputs. Further, EOFM supports the ability to automatically generate specification properties for evaluating how well an interface supports tasks (Bolton et al. 2013; Bolton 2013). This can be used by the oracle to check that an interface design is properly supporting the task (step O.1 from above).

L* Learning

There are several preexisting implementations of the L* learning algorithm that could be used in this project. These include learnlib (Raffelt and Steffen 2006) and libalf (Bollig et al. 2010). However, neither automatically work with a particular model checker nor have they been used to learn human-computer interface designs previously. Alternatively, the Java Pathfinder (Havelund and Pressburger 2000) environment offers an implementation of L* that can work with its native model checking capabilities (Combéfis et al. 2011). Further, Java Pathfinder's L* learner was successfully used in Combéfis et al.'s (2011; 2012) interface generation effort. Thus, while all options are still being considered, Java Pathfinder's L* algorithm will likely be used in the implementation of our approach.

The Teacher and the Model Checker

Both the teacher and oracle are effectively searching through models to evaluate execution sequences and interface designs. As such, a model checker should be able to serve both purposes. Given the probable choice of Java Pathfinder's L* Learning algorithm for our implementation, its model checker is a prime candidate for use in our teacher and oracle. However, this choice will necessitate several developments. Firstly, a translator will need to be constructed to convert EOFM task models into a notation readable by Java Pathfinder. It will also require that all utilized specification properties (including those generated from EOFMs) be adapted to work in Java Pathfinder.

Alternatively, the model checkers found in the symbolic analysis laboratory (SAL) could be used. This would be advantageous because EOFM's translator and property generation tools already work with SAL (Bolton, Siminiceanu, and Bass 2011; Bolton 2013; Bolton et al. 2013). However, this would require that an additional translators be constructed to coordinate between SAL and the L* learner.

Future work will evaluate these options.

Conclusions and Future work

Humans are increasingly relying on computer technology to help them safely and efficiently do their work and conduct everyday tasks. It is therefore critical that the human-computer interfaces they interact with allow them to usably achieve their goals. If an interface fails to do this, it could severely limit the ability of human operators to do their jobs and effectively live their lives. As such, this work is important because, if successful, it will allow for the automatic generation of user-centered human-computer interface designs with guaranteed usable support for operator tasks. Because this work is in its early stages, there is still much to be done. However, we have identified an approach and developed an implementation plan that will guide future efforts.

To validate our approach, we will use it to design an interface for a realistic system. The first application will be a patient controlled analgesia (PCA) pump. A PCA pump is a medical device that intravenously delivers pain medication to patients based on a prescription programmed into it by a practitioner. This is an appropriate application for this project because task analyses already exist for describing PCA pump interaction (Bolton and Bass 2010; 2013b) and PCA pump usability is important to patient safety. Further, a formal reference standard exists for describing the interface behavior of PCA pumps (Arney et al. 2007). This provides a robust design against which a generated model can be compared. Future work will investigate this and other potential applications for our approach.

Finally, by creating formal interface designs as formal models, our approach enables human-computer interfaces to be evaluated in ways that may not have been available to designers otherwise. For example, formal verification could be used to prove properties about system safety and formal tools could be used to automatically generate test cases for certification. Thus, future work should investigate how to best adapt our approach to support these analyses.

References

Abowd, G. D.; Wang, H.; and Monk, A. F. 1995. A formal technique for automated dialogue development. In *Proceedings of the 1st Conference on Designing Interactive Systems*, 219–226. New York: ACM.

Aït-Ameur, Y.; Baron, M.; and Girard, P. 2003. Formal validation of HCI user tasks. In *Proceedings of the International Conference on Software Engineering Research and Practice*, 732–738. Las Vegas: CSREA Press.

Angluin, D. 1987. Learning regular sets from queries and counterexamples. *Information and computation* 75(2):87–106.

Arney, D.; Jetley, R.; Jones, P.; Lee, I.; and Sokolsky, O. 2007. Formal methods based development of a pca infusion pump reference model: Generic infusion pump (GIP) project. In *Proceedings of the Joint Workshop on High Confidence Medical Devices, Software, and Systems and Medical Device Plug-and-Play Interoperability*, 23–33. IEEE.

Barboni, E.; Ladry, J.-F.; Navarre, D.; Palanque, P.; and Winckler, M. 2010. Beyond modelling: An integrated environment supporting co-execution of tasks and systems models. In *Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems*, 165–174. ACM.

Basnyat, S.; Palanque, P.; Schupp, B.; and Wright, P. 2007. Formal socio-technical barrier modelling for safety-critical interactive systems design. *Safety Science* 45(5):545–565.

Bollig, B.; Katoen, J.-P.; Kern, C.; Leucker, M.; Neider, D.; and Piegdon, D. R. 2010. libalf: The automata learning framework. In *Computer Aided Verification*, 360–364. Springer.

Bolton, M. L., and Bass, E. J. 2009. A method for the formal verification of human interactive systems. In *Proceedings of the 53rd Annual Meeting of the Human Factors and Ergonomics Society*, 764–768. Santa Monica: HFES.

Bolton, M. L., and Bass, E. J. 2010. Formally verifying human-automation interaction as part of a system model: Limitations and tradeoffs. *Innovations in Systems and Software Engineering: A NASA Journal* 6(3):219–231.

Bolton, M. L., and Bass, E. J. 2012. Using model checking to explore checklist-guided pilot behavior. *International Journal of Aviation Psychology* 22:343–366.

Bolton, M. L., and Bass, E. J. 2013a. Evaluating human-human communication protocols with miscommunication generation and model checking. In *Proceedings of the Fifth NASA Formal Methods Symposium. Moffett Field: NASA Ames Research Center*, 48–62. Moffett Field: NASA Ames Research Center.

Bolton, M. L., and Bass, E. J. 2013b. Generating erroneous human behavior from strategic knowledge in task models and evaluating its impact on system safety with model checking. *IEEE Transactions on Systems, Man and Cybernetics: Systems*. In Press.

Bolton, M. L.; Bass, E. J.; and Siminiceanu, R. I. 2012. Using phenotypical erroneous human behavior generation to evaluate human-automation interaction using model checking. *International Journal of Human-Computer Studies* 70:888–906.

Bolton, M. L.; Bass, E. J.; and Siminiceanu, R. I. 2013. Using formal verification to evaluate human-automation interaction in safety critical systems, a review. *IEEE Transactions on Systems, Man and Cybernetics: Systems* 43:488–503.

Bolton, M. L.; Jimenez, N.; van Paassen; M., M.; and Trujillo, M. 2013. Formally verifying human-automation interaction with specification properties generated from task analytic models. In *Proceedings of the 6th IAASS Conference*. Montreal: IAASS. In Press.

Bolton, M. L.; Siminiceanu, R. I.; and Bass, E. J. 2011. A systematic approach to model checking human-automation interaction using task-analytic models. *IEEE Transactions on Systems, Man, and Cybernetics, Part A* 41(5):961–976.

Bolton, M. L. 2013. Automatic validation and failure diagnosis of human-device interfaces using task analytic models and model checking. *Computational and Mathematical Organization Theory* 19(3):288–312.

Bowen, J., and Reeves, S. 2008. Formal models for user interface design artefacts. *Innovations in Systems and Software Engineering* 4(2):125–141.

Campos, J. C., and Harrison, M. 1997. Formally verifying interactive systems: A review. In *Proceedings of the Fouth International Eurographics Workshop on the Design, Specification, and Verification of Interactive Systems*, 109–124. Berlin: Springer.

Campos, J. C., and Harrison, M. D. 2008. 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*, 72–85. Berlin: Springer.

Campos, J. C., and Harrison, M. D. 2009. Interaction engineering using the ivy tool. In *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 35–44. New York: ACM.

Campos, J. C. 2003. Using task knowledge to guide interactor specifications analysis. In *In Proceedings of the 10th International Workshop on Interactive Systems. Design, Specification, and Verification.*, 171–186. Berlin: Springer.

Chen, Y.-F.; Farzan, A.; Clarke, E. M.; Tsay, Y.-K.; and Wang, B.-Y. 2009. Learning minimal separating dfa's for compositional verification. In *Proceedings of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 31–45. Berlin, Heidelberg: Springer-Verlag.

Clarke, E. M., and Wing, J. M. 1996. Formal methods: State of the art and future directions. *ACM Comput. Surv.* 28(4):626–643.

Combéfis, S., and Pecheur, C. 2009. A bisimulation-based approach to the analysis of human-computer interaction. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. New York: ACM.

Combéfis, S., and Pecheur, C. 2012. Automatic generation of full-control system abstraction for human-machine interaction. 9.

Combéfis, S.; Giannakopoulou, D.; Pecheur, C.; and Feary, M. 2011. Learning system abstractions for human operators. In *Proceedings of the 2011 International Workshop on Machine Learning Technologies in Software Engineering*, 3–10. New York: ACM.

Degani, A., and Heymann, M. 2002. Formal verification of human-automation interaction. *Human Factors* 44(1):28–43.

DIS, ISO. 2009. 9241-210: 2010. Ergonomics of human system interaction-part 210: Human-centred design for interactive systems. *International Organization for Standardization (ISO), Switzerland*.

España, S.; Pederiva, I.; and Panach, J. I. 2007. Integrating model-based and task-based approaches to user interface generation. In *Computer-Aided Design of User Interfaces V*. Netherlands: Springer. 253–260.

García, J. G.; Lemaigre, C.; González-Calleros, J. M.; and Vanderdonckt, J. 2008. Model-driven approach to design user interfaces for workflow information systems. *Journal of Universal Computer Science* 14(19):3160–3173.

Giannakopoulou, D., and Păsăreanu, C. S. 2009. Interface generation and compositional verification in javapathfinder. In *Fundamental Approaches to Software Engineering*. Springer. 94–108.

Gunter, E. L.; Yasmeen, A.; Gunter, C. A.; and Nguyen, A. 2009. Specifying and analyzing workflows for automated identification and data capture. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*, 1–11. Los Alatimos: IEEE Computer Society.

Harrison, M., and Duke, D. 1995. A review of formalisms for describing interactive behaviour. In *Proceedings of the Workshop on Software Engineering and Human-Computer Interaction*, 49–75. London: Springer.

Havelund, K., and Pressburger, T. 2000. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer* 2(4):366–381.

Heitmeyer, C. 1998. On the need for practical formal methods. In *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time Fault-Tolerant Systems*, 18–26. London: Springer.

Joshi, A.; Miller, S. P.; and Heimdahl, M. P. 2003. Mode confusion analysis of a flight guidance system using formal methods. In *Proceedings of the 22nd Digital Avionics Systems Conference*, 2.D.1-1–2.D.1-12. Piscataway: IEEE.

Kirwan, B., and Ainsworth, L. K. 1992. *A Guide to Task Analysis*. London: Taylor and Francis.

Mahfoudhi, A.; Abid, M.; and Abed, M. 2005. Towards a user interface generation approach based on object oriented design and task model. In *Proceedings of the 4th International Workshop on Task Models and Diagrams*, 135–142. New York: ACM.

Navarre, D.; Palanque, P.; Paternò, F.; Santoro, C.; and Bastide, R. 2001. A tool suite for integrating task and system models through scenarios. In *Proceedings of the 8th International Workshop on Interactive Systems: Design, Specification, and Verification*, 88–113. Berlin: Springer.

Palanque, P., and Paternò, F., eds. 1997. *Formal Methods in Human-Computer Interaction*. Secaucus: Springer.

Parnas, D. L. 1969. On the use of transition diagrams in the design of a user interface for an interactive computer system. In *Proceedings of the 24th National ACM Conference*, 379–385. New York: ACM.

Paternò, F., and Santoro, C. 2001. Integrating model checking and HCI tools to help designers verify user interface properties. In *Proceedings of the 7th International Workshop on the Design, Specification, and Verification of Interactive Systems*, 135–150. Berlin: Springer.

Paternò, F. 1997. Formal reasoning about dialogue properties with automatic support. *Interacting with Computers* 9(2):173–196.

Raffelt, H., and Steffen, B. 2006. Learnlib: A library for automata learning and experimentation. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering*, 377–380. Berlin: Springer.

Schraagen, J. M.; Chipman, S. F.; and Shalin, V. L. 2000. *Cognitive Task Analysis*. Philadelphia: Lawrence Erlbaum Associates, Inc.

Tran, V.; Kolp, M.; Vanderdonckt, J.; Wautelet, Y.; and Faulkner, S. 2010. Agent-based user interface generation from combined task, context and domain models. In *Proceedings of the 8th International Workshop on Task Models and Diagrams for User Interface Design*, 146–161. Berlin: Springer.

Wing, J. M. 1990. A specifier's introduction to formal methods. *Computer* 23(9):8, 10–22, 24.