High assurance interactive computing systems

José Creissac Campos
Departamento de Informática, Universidade do Minho & HASLab / INESC TEC
Braga, Portugal
jose.campos@di.uminho.pt

ABSTRACT

If interactive computing systems development is to be considered an engineering discipline, we need methods and tools to help us reason about and predict the quality of systems, from early in the design process. This paper provides a brief overview of work we have been carrying out in the general area of evaluating and ensuring the quality of interactive computing systems. Some of the work currently being carried out is also discussed. Discussed approaches range from the formal verification of user interface models through model checking, to the reverse engineering and model based testing of implemented interactive computing systems.

INTRODUCTION

Software has become present in all aspects of our lives, from safety critical applications, such as medical devices and the cars we drive, to social networks and games running on our phones. As the losses and disruption caused by software failures rise in severity, so does the need to guarantee that software will function *correctly*. This is a particular challenge for interactive computing systems, given not only the presence of the human factor, which must be considered during the analysis, but also the continuous evolution of interaction and implementation technologies, which make it difficult to asses, *a priori*, the quality of a system at development stages. Nevertheless, if the development of user interfaces is to be an engineering discipline, it must have techniques and tools to enable this analysis during development so that quality can be measured and predicted.

This paper briefly describes work that we have been carrying out to support the analysis of interactive computing systems. Building on that work it then outlines our views on some future lines of research. The rational behind the proposals for future work is that we need techniques and tools that better fit the typical development process of interactive computing systems. That is to say, techniques and tools that enable us to analyse the systems as they are developed. To achieve this we need means of leveraging, not only the analysis of any models that might have been produced during development, but also of the source code produced.

PREVIOUS WORK

One major goal of our work is to support the exhaustive, systematic and, and much as possible, automated analysis of interactive commuting systems. With this aim as the backdrop, a number of research direction have been pursued and are now discussed. A common trait of all the approaches is the focus on the system. That is, the main requirement is that either a model of the system, or the actual system, is available for analysis. An alternative approach, not explored here, would be to focus on cognitive models of the user (see, for example, ACT-R [1] or PUM [19]).

In order to support the systematic and exhaustive analysis of user interfaces, we have developed the IVY workbench[1] [6]. The tool supports the formal verification of interactive computing systems using model checking [8]. It aims to cater for the full cycle of analysis, from modelling to interpretation of the verification results. Models are expressed in a domain specific language (MAL interactors), and properties for verification in Computational Tree Logic (CTL) [8]. IVY has been applied to a number of different systems, mostly in safety critical domains (for example, medical [5] and aerospace [18]). How best to fold considerations about the user into the analysis has been a recurring concern, and is typically done by making explicit assumptions about how the user will react to the user interface [4]. The goal being to guarantee that only cognitively plausible behaviors are considered during the verification.

IVY has proven suitable to analyze control panel and WIMP style interfaces, but less so when considering larger interaction contexts, such as when considering ubiquitous computing systems. With the APEX framework[2] [16] we have specifically targeted ubiquitous computing environments. The framework combines a modelling tool (CPN Tools [13]) with a 3D application server (OpenSimulator[3]), in order to combine formal verification and prototyping. On the one hand, the models capture the behavior of active objects in the environment (for example, sensors or public displays), and are amenable to formal verification of their behavior [15]. On the other hand, the use of the 3D application server for prototyping purposes enables an empirical assessment of the user experience of the systems. The approach supports a multi-level evolutionary prototyping approach, where simulate devices can gradually be replaced by their physical counterparts. The simulation itself can resort to models of device

[1] `http://ivy.di.uminho.pt` (last visited 09/04/2014)
[2] `http://ivy.di.uminho.pt/apex` (last visited 09/04/2014)
[3] `http://opensimulator.org` (last visited 09/04/2014)

behavior or be migrated to code in the target virtual reality environment.

Both of the above approaches depend on the development of models (although the multilevel nature of APEX also supports programming the virtual devices directly in the environment). This begs two questions. The first regards the availability of models for analysis, the second the extent to each the models faithfully capture the relevant features of the system once developed, given the specific analysis being carried out. This last issue impacts the validity of the analysis, and what it means of the actual system once deployed.

Regarding the first issue, while Model Based User Interface Development approaches (consider, for example, [3]) advocate the use of models, there are concerns about the quality (from a user's perspective) of the user interfaces developed in this fashion (mainly when automatic refinement of models into code is considered). Indeed, human-centered design approaches typically advocate a process based on iterative prototyping and testing [10]. Additionally, agile approaches focus much earlier on the production of code [11]. This means that it is not guaranteed the models will necessarily be available for analysis.

When models are available, and especially when the generation of the system is not automated, the question arises of how faithful a representation of the code the model is. This not only is a problem due to the need to guarantee that the code is correctly generated from the original model, but also to the need to guarantee that both ends of the development process (models and code) are kept synchronized. Indeed, a known problem of model based software engineering is maintaining the consistency between model and source code.

Considering the above, we have been exploring the use of model based testing to directly analyze actual running application [17]. Model based testing works by comparing the prescribed behavior of a system (captured in a model – the oracle), with the actual behavior of the system while running. Task models were used to generate the oracle, as they describe the intended use of the system. The possibility of introducing mutations in the task model was also explored, so that deviation from the norm (for example, user errors) could be considered during the analysis [2]. The need for an oracle, however, means that a model is still needed to perform the analysis.

CURRENT AND FUTURE WORK

While the approaches described above have proven able to provide insights into the design and trustworthiness of interactive computing systems, the need for models presents a barrier to adoption. Lighter-weight alternatives are needed when the cost of the modelling step in not justifiable.

We have been exploring alternatives to reason about the quality of an interactive computing system directly from its implementation. The goal is to better integrate the analysis with development contexts more centered around the production of code, such as some agile approaches or less structured development processes, as is sometimes the case on Web and mobile applications' development.

One first approach was to use static analysis techniques to reverse engineer models from source code [7]. This allowed us, not only to apply the model analysis techniques we already had available on the outcome of the reverse engineering step, but also to identify problems directly in the code during the reverse engineering step itself. For example, user interface components declared but never used. This type of approach, however, is hard to generalize, which, especially in the case of Web applications, becomes a problem given the multiplicity of options regarding implementation technologies. It is also not easy to apply when we consider adaptive or dynamically generated user interfaces, as the concrete interface is only known at run time, and typically not easy to deduce from the source code alone.

More recently, we have started looking at alternative approaches to analyzing the quality of the implementation. One direction explores the know how obtained with the reverse engineering and model based testing work. It consists in applying hybrid analysis techniques to perform both direct analysis and reverse engineering [14]. The approach, targeted at Web applications, combines dynamic exploration of the user interface, with static analysis of the event listeners. This approach presents a number of benefits. Compared with dynamic analysis, it enables us to achieve better coverage of the system's state space during exploration, as well as a more detailed model of the system. Compared with static analysis, it enables us to significantly reduce generalization problems, and problems with dynamically generated user interfaces, as we analyze and extract code from the running application, limiting the amount of static analysis to a minimum. Given the distributed nature of Web applications, and to minimize generalization problems, code instrumentation is used to simulate different responses from the business logic.

Another direction is exploring the idea of code smells [9]. A *code smell* highlights some feature of the code that, while not necessarily an error, might indicate a weakness in the system's implementation. Our ultimate goal is to apply the concept to user interfaces analysis and define a set of usability related smells. As a starting point we are looking at traditional WIMP interfaces, but we envisage that for different interaction techniques different smells will have to be defined. Thus far, we have found that while some of the smells we have identified relate to the implementation's quality, others relate to the quality of the resulting user interface. How to automate their analysis is still an open issue. For code related smells, the know how on reverse engineering can once again be leveraged. For user interface related smells, we intend to explore which type of models might be needed (and possible to obtain) in order to support their detection.

An alternative to attempting to avoid the need for models in the analysis, is to improve their added value. If more value can be obtained from the modelling process, and if its cost can be lowered, then the cost of developing a model can be better justified. APEX already points in that direction by making models the basis for both formal verification and prototyping. We are currently extending those ideas into IVY, exploring

the feasibility of using MAL models to support the prototyping of the user interfaces in the style of [12].

A related but somewhat different role is performed by models' animation. While a prototype helps validate the design with users by presenting them with a version of the user interface, which is derived from (and controlled by) the model, an animation is intended to help in an initial validation of whether the model is the intended one. This is achieved by supporting direct interaction with the model itself. In the case of APEX this is supported by CPN Tools. Regarding IVY, however, model validation is currently carried out by exploring the design through proving properties. This is an expensive process that can be made more cost effective by supporting direct interaction with the model. Once some degree of confidence about the model is achieved via animation, further analysis can then be carried out through verification. Bringing together model animation, formal verification and prototyping, all based in a single model will considerable raise the cost effectiveness of building the model.

CONCLUSION

The pervasiveness of software makes us more and more dependent on its quality. It is thus unfortunate that for the most part the quality of the software being produced is still somewhat lacking. This is also true, and particularly relevant, of interactive computing systems. If their development is to be considered an engineering discipline, then we need methods and tools to help us reason about, and predict, the quality of systems from early in the design process.

This paper has provided a brief overview of the work we have been carrying out under the generic umbrella of reasoning about, and ensuring the quality of, interactive computing systems. The described techniques and tools should be seen as an addition to the toolbox of already existing techniques and tools available for interactive computing systems development. Some ideas currently under development have also been highlighted.

ACKNOWLEDGMENTS

IVY development is currently funded by the North Portugal Regional Operational Programme (ON.2 – O Novo Norte), under the National Strategic Reference Framework (NSRF), through the European Regional Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT) within the LATiCES project (NORTE-07-0124-FEDER-000062).

The APEX project is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-015095.

Work on model based testing is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-020554.

REFERENCES

1. Anderson, J. R. *How can the human mind occur in the physical universe?* Oxford University Press, New York, NY, USA, 2007.
2. Barbosa, A., Paiva, A. C. R., and Campos, J. C. Test case generation from mutated task models. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, F. Paternò, K. Luyten, F. Maurer, P. Dewan, and C. Santoro, Eds., ACM (2011), 175–184. ISBN: 978-1-4503-0778-9.
3. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., and Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. *Interacting with Computers 15*, 3 (2003), 289–308.
4. Campos, J., Doherty, G., and Harrison, M. Analysing interactive devices based on information resource constraints. *International Journal of Human-Computer Studies 72*, 3 (March 2014), 284–297.
5. Campos, J., and Harrison, M. Modelling and analysing the interactive behaviour of an infusion pump. *Electronic Communications of the EASST 45: Formal Methods for Interactive Systems 2011* (2011). ISSN: 1863-2122.
6. Campos, J. C., and Harrison, M. D. Interaction engineering using the ivy tool. In *ACM Symposium on Engineering Interactive Computing Systems (EICS 2009)*, ACM (New York, NY, USA, 2009), 35–44.
7. Campos, J. C., Saraiva, J., Silva, C., and Silva, J. C. GUIsurfer: A reverse engineering framework for user interface software. In *Reverse Engineering - Recent Advances and Applications*, A. Telea, Ed. InTech, 2012, ch. 2, 31–54.
8. Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.
9. Fowler, M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
10. International Organization for Standardization. ISO 9241-210:2010 Ergonomics of human-system interaction – part 210: Human-centred design for interactive systems, 2010.
11. Memmel, T., Gundelsweiler, F., and Reiterer, H. Agile human-centered software engineering. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...But Not As We Know It - Volume 1*, BCS-HCI '07, British Computer Society (Swinton, UK, UK, 2007), 167–175.
12. Oladimeji, P., Masci, P., Curzon, P., and Thimbleby, H. PVSio-web: a tool for rapid prototyping device user interfaces in PVS. In *Proceedings of the 5th International Workshop on Formal Methods for Interactive Systems (FMIS 2013)* (2013).

13. Ratzer, A. V., Wells, L., Lassen, H. M., Laursen, M., Qvortrup, J. F., Stissing, M. S., Westergaard, M., Christensen, S., and Jensen, K. CPN Tools for editing, simulating, and analysing coloured Petri nets. In *Proceedings of the 24th international conference on Applications and theory of Petri nets*, ICATPN'03, Springer-Verlag (Berlin, Heidelberg, 2003), 450–462.

14. Silva, C. E., and Campos, J. C. Combining static and dynamic analysis for the reverse engineering of web applications. In *Proceedings of the 5th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2013)*, P. Forbrig, P. Dewan, M. Harrison, K. Luyten, C. Santoro, and S. Barbosa, Eds., ACM (2013), 107–112.

15. Silva, J., Campos, J., and Harrison, M. Formal analysis of ubiquitous computing environments through the APEX framework. In *ACM Symposium on Engineering Interactive Computing Systems (EICS2012)*, ACM (2012), 131–140.

16. Silva, J., Campos, J., and Harrison, M. Prototyping and analysing ubiquitous computing environments using multiple layers. *International Journal of Human-Computer Studies 72*, 5 (May 2014), 488–506.

17. Silva, J. L., Campos, J. C., and Paiva, A. Model-based user interface testing with spec explorer and concurtasktrees. *Electronic Notes in Theoretical Computer Science 208: 2nd International Workshop on Formal Methods for Interactive Systems (FMIS 2007)* (2008), 77–93.

18. Sousa, M., Campos, J., Alves, M., and Harrison, M. Formal verification of safety-critical user interfaces: a space system case study. In *Formal Verification and Modeling in Human Machine Systems: Papers from the AAAI Spring Symposium*, AAAI Press (2014), 62–67.

19. Young, R. M., Green, T. R. G., and Simon, T. Programmable user models for predictive evaluation of interface designs. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '89, ACM (New York, NY, USA, 1989), 15–19.