Interactors as Boundary Objects

José Creissac Campos
Departamento de Informática, Universidade do Minho, Portugal
jose.camposdi.uminho.pt

1 Background

I am a lecturer at the Informatics Department of the University of Minho, having worked for a few years as a software engineer developing both custom made and 'off the shelf' products. I am integrated in a group with three main areas of interest: Human-Computer Interaction (HCI), object oriented (OO) software engineering (including UML), and computer graphics. My main research interests lay in the intersection between HCI and software engineering. My teaching duties include HCI and OO.

I hold a *Licenciate* degree in Software Engineering from the University of Minho, and a M.Sc. in Computer Science, also from the University of Minho, with a dissertation on the rapid prototyping of user interfaces from functional specifications.

I got my D.Phil. in Computer Science at the University of York, with a thesis on applying automated reasoning tools to model based usability analysis. This work was developed under the supervision of Prof. Michael Harrison while at the York HCI group, a interdisciplinary group with members coming both from the Computer Science and Psychology departments.

2 Position

Too often software engineers see the user interface as a last layer that must be placed on top of the functional (or business logic) layer in order (simply?) to enable users access to its functionality. This vision rests on the assumption that all the application's logic is at the functional level, and is independent of the user interface. The user interface then is simply a information transmission layer between the user and the "*application*" (i.e., the functional layer). Even a software development method such as RUP, which is strongly based around the notion of Use Case, pays little attention to the design of the user interface. RUP transforms use cases directly to the architectural design of the functional layer of the application. Such approaches compromise the quality of both the users' interface and of the code, in terms of

- poor interaction experience.

  User interface layers in practice are required to implement control logic. In current applications, control logic of the user interface layer is usually tightly coupled with the logic of the functional layer. Both layers must be adequately designed so that they can provide a high quality of user interaction.

- poor code.

  An inadequate design specification leads to a user interface layer that is developed in a more or less *ad-hoc* fashion. Design and the implementation will require frequent update as problems are found and improvements requested.

In order to address these issues more effectively, better integration of human factors concerns into the software engineering life cycle is needed. This can be achieved by carrying out usability analyses of system designs early in the development process. Usability analysis should not be left to the later stages of development when changes will be more difficult and expensive to make. Lightweight techniques are

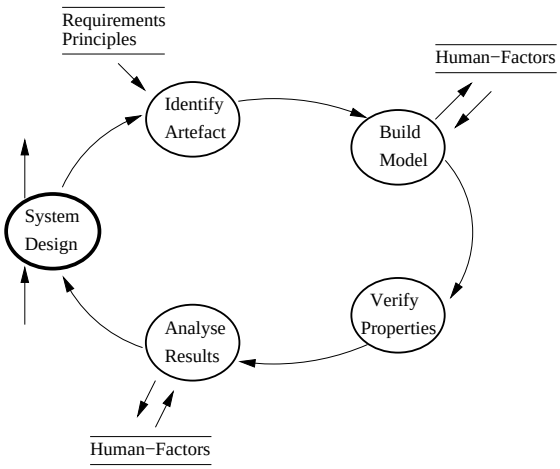

Figure 1: Integration of verification in development

needed that enable reasoning about the usability of systems as early as possible so that design can be shaped by usability criteria and concerns.

Discount methods for the analysis of usability have been proposed that are cheaper to apply than more traditional empirical methods. They have the advantage that they do not require substantial planning, and can be used in the early stages before the design is implemented but they do tend to require human factors expertise. This creates problems when they form part of a process that is carried out by software engineers who do not have such skills. There is thus a need for an interdisciplinary development process involving software engineers, human-factors experts and designers.

A complicating factor, when it comes to the analysis step, is the sheer size and complexity of the models that must be considered when developing complex systems. Usually, the above mentioned discount methods tend to focus on what could be called surface issues in the interaction, with little consideration of more complex behavioural issues of the interaction between user and device that will arise in real usage conditions. Techniques and tools are required that enable a more thorough analysis of interactive systems, in an interdisciplinary context. I (with Michael Harrison) have been working towards a method for the design of the user interface that involves joint analysis by software engineers and human-factors experts. The approach taken is that depicted in figure 1. Whenever usability issues must be considered, a model is built that captures salient interactive features of the design. The model is used to capture the intended design, and at this stage different alternatives can be considered. Once a satisfactory model is reached, the desired usability properties must be expressed and afterwards verified of the model. If the verification fails further redesign is needed.

In order to enable the analysis of the behaviour of non-trivial systems, models are built from components using a modelling language with rigorous semantics. Components are described using the notion of interactor in the style of [DH93]: an object-like entity which is capable of rendering (part of) its state into some presentation medium. The state of each interactor is described by a set of attributes and annotated with the rendering relation. The behaviour is described by axioms in Modal Action Logic (MAL). The resulting MAL specification is similar to a production systems' style specification. This style of specification has been found useful and understandable by psychologists (see [CM95]).

Properties that are to be checked of the system (in much the same style as usability inspection) are written in Computational Tree Logic (CTL). CTL enables the expression of such properties over the behaviour of the model as: some state can eventually be reached; or some state will inevitably be reached.

We have developed a tool that enables the translation of interactive systems models into the SMV model checker input language (i2smv) [CH01]. When the properties do not hold, SMV attempts to provide a counter-example in the form of a trace leading to a state where the property does not hold. Analysis of the trace will lead to a new iteration of the process where either the model or the property have been modified.

3 Boundary objects

In the work described it can be said that a number of boundary objects are involved:

- the interactor models — they are used to describe the user interface (with particular emphasis on its behaviour). Human-factors experts, interaction designers, and software engineers can use them as a basis for discussion about the user interface design. The models capture both the structure of the interface's presentation (at a high level of abstraction), and the behaviour of the interface in terms of the effect of user actions on the interactors' state. Interaction designers can use these models to express both what information should be present in the interface (and how it should be presented) and how the interface behaves in response to user input. Usability experts can use the models to identify potential usability related problems and desirable properties. Software engineers can use the models as road-maps to the implementation of the systems.

- the CTL formulae — the formulae capture desirable user interface properties and test the model. Once properties required to verify the model are identified, software engineers code them in CTL. Defining these properties, and how to encode them in CTL, will promote discussion about the characteristics of the user interface, and how they impact on the design. Assumptions about how the user will interpret specific user interface elements might be an issue at this stage. Such assumptions will have to be discussed with interaction designers and usability experts.

- the SMV traces — these traces represent behaviours where the user interface fails the property under consideration. They can be used as triggers for scenarios that falsify the property, and can be used to discuss why the user interface does not hold that specific property (characteristic) and how to best correct the problem. For example, circumstances (narratives) including the sequence defined by the trace can be constructed by a domain expert. It might happen that some specific behaviour can be considered irrelevant (for example, when analysing how an airplane's autopilot supports the pilot in acquiring a target altitude, we might not want to consider the case where the pilot turns the autopilot off). In this case, the property will have to be changed to rule out unwanted behaviours and again these changes will be negotiated between the software engineer and human factors expert. In this context, the property can also be seen as triggering a scenario: the scenario under which the property will hold of the system.

Together, these *boundary objects* enable the characterization of (specific features of) an interactive system, of its desirable properties, and of scenarios relating to such properties (both where a given property holds and where it fails).

Acknowledgments

I wish to thank Michael Harrison for his valuable input on this paper.

References

[CH01] José C. Campos and Michael D. Harrison. Model checking interactor specifications. *Automated Software Engineering*, 8(3-4):275–310, August 2001.

[CM95] Martin B. Curry and Andrew F. Monk. Dialogue modelling of graphical user interfaces with a production system. *Behaviour & Information Technology*, 14(1):41–55, 1995.

[DH93] David J. Duke and Michael D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3):25–36, 1993.