Verifying user interface behaviour with model checking

José Creissac Campos[1], Michael D. Harrison[2], and Karsten Loer[2]

[1] Departamento de Informática, Universidade do Minho
Campus de Gualtar, 4715-187 Braga, Portugal

[2] Department of Computer Science, University of York
Heslington, York YO10 5DD, UK

Abstract. A large proportion of problems found in deployed systems relate to the user interface. This paper presents an approach to the verification of user interface models based on model checking. The approach is intended to be used early in design. The verification is concerned with behavioural aspects of the user interface and requires models that represent both the interactive aspects and also capture important features of the context to allow restrictions of behaviour to those that conform to appropriate human and environmental constraints. A tool suite to support the approach is under development and is described. Future work directions are put forward.

1 Introduction

Large software development projects typically do not benefit from current software engineering practices. It can be argued that the lack of use of these practices has led to poor success rates in these developments. In a study carried out in North America it was concluded that 40% of deployed systems had marginal gains, 40% were rejected, and only 20% of the systems were considered truly successful (cited in [1, chapter 1]) .

Other studies have shown that a large proportion of problems found in deployed systems relate to the user interface. “Although valid figures are difficult to obtain there seems to be general agreement to attribute somewhere in the range of 60-90% of all system failures to erroneous human actions” [2]. However, as Leveson points out [3], classifying a human-computer interaction problem as human error is too simplistic. In most situations it is not clear whether the error lies with the “operator” of the system or with the designers who have created the system.

There is clearly the need for a better integration of human-factors concerns into the software engineering life-cycle. Despite the emphasis being placed on engineering more reliable hardware and software systems, relatively little significance has been attached to human factors issues. Even a recent 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. This situation becomes more extreme with the introduction of ubiquitous technologies where devices are embedded in complex user environments.

In this paper we argue (1) that there should be a closer integration of usability issues into the software development life cycle; (2) that, more generally, there is need to take

Creissac Campos J., D. Harrison M. and Loer K. (2004).
Verifying user interface behaviour with model checking.
In *Proceedings of the 2nd International Workshop on Verification and Validation of Enterprise Information Systems*, pages 87-96
DOI: 10.5220/0002681700870096

a more careful account of context, including human use in the modelling and analysis of the device. Section 2 discusses the gap between software engineering and interactive systems development. Section 3 presents discount usability evaluation methods. Section 4 and 5 propose an approach to usability analysis, and a accompanying tool based on model checking. Section 6 discusses the the need for the use of constraints in the analysis of interactive systems. Section 7 presents some directions for future work, and finally section 8 draws some conclusions.

2 Developing the user interface

Usability concerns are typically not among the major concerns of software engineers during the early stages of design. It is common to find software engineers designing the user interface as a last layer to be placed on top of the system above function or business logic in order 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 passive information transmission layer between the user and the "*application*". Such approaches compromise the quality of the software system that is being produced both in terms of the quality of the user's experience and the quality and maintainability of the implemented code.

User interface layers are, in practice, required to implement control logic. In current applications, the control logic of the user interface is usually tightly coupled with the logic of the functional layer. This happens both in terms of the state and behaviour of user interface objects as they reflect the underlying function. Both layers must be adequately designed so that they can co-operate in order to provide a high quality user interaction.

An inadequate or non-existent design specification leads to a user interface layer that is developed in a more or less *ad-hoc* fashion. This problem is exacerbated by current IDE tools which provide support for the design and construction of the graphical aspects of the user interface, but provide little support for the design of the behavioural aspects. This leads to a situation where the design or, more precisely, the implementation, will require frequent update as problems are found and improvements requested.

3 Discount approaches to usability analysis

In order to address the above issues more effectively, better integration of human factors concerns into the software engineering life cycle is needed. This can be achieved by considering usability issues from early in the development process. Lightweight techniques are 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. Two such methods are Heuristic Evaluation [4] and Cognitive Walkthroughs [5]. Heuristic Evaluation involves systematic inspection of the design by means of guidelines for good practice in the design. It is assumed that there are a number of general characteristics that all usable systems should

exhibit. Cognitive Walkthroughs, on the other hand, aim at analysing how well the interface will guide the user in performing tasks. User tasks must first be identified, and the model of the interface (the device model) must be sufficiently detailed to cover all possible courses of action the user might take. Analysis of how a user would execute the task is performed by asking three simple questions at each stage of the interaction: *Will the correct action be made sufficiently evident to users? Will users connect the correct action's description with what they are trying to achieve? Will users interpret the system's response to the chosen action correctly?* To answer these questions, probable courses of user action must be presumed and documented as so-called user models.

Discount approaches have the advantage that they do not require substantial planning, and can be used in the early stages before the design is implemented. They operate on some informal representation of the design, for example a storyboard or a draft of the user manual. However, because these methods are largely informal it is more difficult to perform them systematically and therefore ensure coverage. Additionally, the methods rely on application by those who have sufficient expertise to apply them correctly — for example, interpreting any questions that are asked of the design appropriately. This creates problems when technique application takes place in the context of a development and analysis 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. At worst the 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. It requires the specialist analyst to raise the deeper questions. Techniques and tools are required that enable a more thorough analysis of interactive systems in an interdisciplinary context.

4 Integrating usability analysis into software development

We 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. In it, design and verification become part of the same development process, where verification is used to inform design decisions. Four main steps can be identified for this type of approach: 1) selecting what to verify; 2) building an appropriate model; 3) performing the verification; 4) analysing the results.

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.

It must be stressed that these models are not prescriptive but descriptive. That is, they are used not to express *how* the system should be implemented, rather they are used to express *what* system should be implemented. At this stage the system implementation is not of interest, instead focus is on the interaction between the system and its users. The aim of the approach is that it might be applied from the early design stages, when the question is "what should be built?", rather than "how should we build it?".

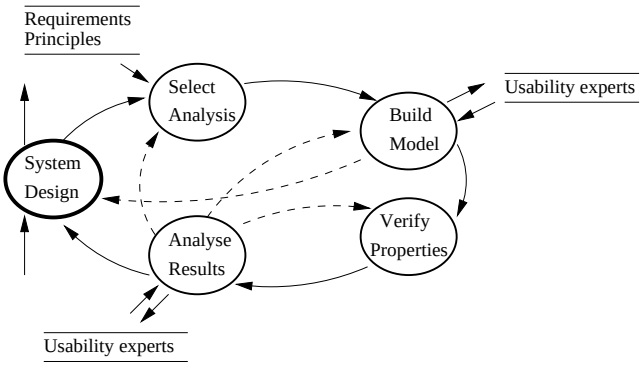

Fig. 1. Integration of verification in development

Once a satisfactory model is reached, the desired usability properties must be expressed and afterwards verified of the model. From an analysis point of view, the most interesting situations are those where the verification fails. In this case, the causes for the failure must be analysed, this will lead to a new iteration of the process where either the model or the property have been modified.

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 [6]: 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) [7]. The resulting MAL specification is similar to a production systems' style specification (see Figure below 3 for an example interactor model).

Properties that are to be checked of the system are written in Computational Tree Logic (CTL) [8]. CTL enables the expression of properties over the behaviour of the model. The properties that can be written/verified deal mainly with which states can or cannot be reached. Typical properties include: X is an invariant (AG(X) — X holds in all states of all behaviours); X is inevitable (AF(X) — for all possible behaviours, X will eventually hold); X is possible (EF(X) — for at least one behaviour X will eventually hold). Different combinations of the operators can be used to express more complex properties.

5 A tool for automated usability analysis

A tool has been developed that enables the translation of interactive systems models into the SMV model checker input language (i2smv) [9]. SMV [10] is a model checker which uses CTL as the logic to express properties. Models are defined as finite state machines, and CTL is used to express properties over the behaviour of the models. 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.

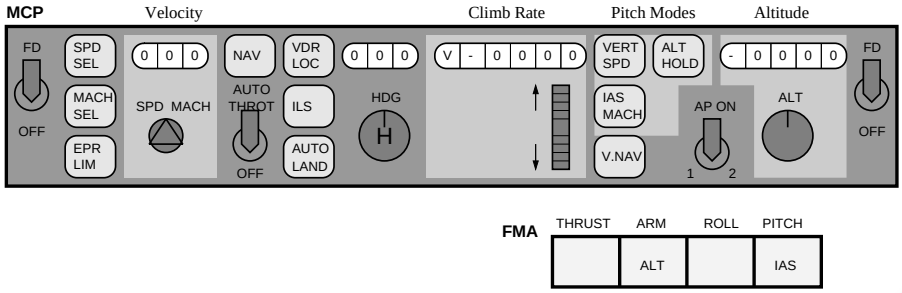

Fig. 2. The Mode Control Panel (adapted from [15])

With the tool it becomes possible to perform an exhaustive search over an interactor model's behaviour in order to establish that certain properties hold (for example: usability principles, access control rules and timing properties — which can all be formulated in terms of safety or liveness properties [11, chapter 5], [12], [13]). From traces that are generated by the model checker, the user behaviour can be extracted and interpreted.

A number of case studies have been carried out in order to access the validity of the approach and the tool. In [14] the user interface of ECOM, an Audio-Visual Communication System, is analysed. It is demonstrated how the integration of two different mechanisms for the control of accessibilities in the system creates an user interface that is difficult to use.

In [9] the Mode Control Panel (MCP) of the MD-88 aircraft (see Figure 2) is analysed with respect to its predictability. More specifically, it was decided to analyse the task of setting and acquiring a target altitude. Figure 3 shows an extract of the interactor model developed for this purpose. The model covers the lighter coloured areas in Figure 2. It resorts to two auxiliary interactors: `plane` and `dial`. Interactor `plane` is used to model the relevant physical features of the aircraft (altitude, velocity and climb rate). Interactor `dial` is used to model the several dials and knobs present at the interface. Other relevant features such as the pitch mode and the state of the altitude capture are modelled as attributes. Actions are defined for each event that can take place at the user interface (both by user initiative - those annotated with vis — or by system initiative). Finally, axioms are used to model the behaviour of the system. It is not the purpose here to present a complete account of the case study (see [9] for further details). It suffices to say that the first axiom defines the effect of changing the climb rate on the system, the second the effect of the enterAC action (arming the altitude capture), the third and fourth axioms set permission and obligation conditions on this last action, and the fifth axiom presented expresses an invariant over the state of the model. The complete model has sixteen axioms.

In [9] the analysis was concerned with the capability of the system to acquire the target altitude in response to the user setting the altitude capture mode. Note that the system is analysed from the perspective of its interaction with the user. The model does not try to express how the system is implemented. Rather, as with more traditional

```
interactor MCP
includes
  aircraft via plane
  dial(ClimbRate) via crDial
  dial(Velocity) via asDial
  dial(Altitude) via ALTDial
attributes
  vis pitchMode: PitchModes
  vis ALT: boolean
actions
  vis enterVS, enterIAS, enterAH, toggleALT
  enterAC
axioms
  [crDial.set(t)] pitchMode′=VERT_SPD ∧ ALT′=ALT
  [enterAC] pitchMode′=ALT_CAP ∧ ¬ALT′
  per(enterAC) → (ALT ∧ |ALTDial.needle - plane.altitude|≤2)
  (ALT ∧ |ALTDial.needle - plane.altitude|≤2) → obl(enterAC)
  pitchMode=VERT_SPD → plane.climbRate=crDial.needle
  ...
```

Fig. 3. Extract from the MCP model

discount usability evaluation methods, it captures what information and behaviour it presents to users. A first attempt at expressing the desired property in CTL was:

```
AG((plane.altitude < ALTDial.needle & ALT) ->
    AF(pitchMode=ALT_HLD & plane.altitude=ALTDial.needle))
```

This reads: it always (AG) happens that if the plane is below the altitude set on the MCP and the altitude capture is on then (AF) the altitude will always be reached and the pitch mode be changed to altitude hold.

This model was translated into SMV using the i2smv tool, after checking a trace was produced showing that the property did not hold. This trace identified a user behaviour that would lead the aircraft to abandon the altitude capture mode (for example, by explicitly changing the pitch). Since the particular situation where the pilot would want the target altitude to be acquired was of interest it was decided to redefine the property so that those behaviours would not be reported:

```
AG((plane.altitude < ALTDial.needle & ALT) ->
    AF((pitchMode=ALT_HLD & plane.altitude=ALTDial.needle)
       | (plane.climbRate = -1))
```

The attempt to check the property still failed, and the resulting trace highlighted a situation where an implicit mode change in the state of the system could create an altitude bust incident (the aircraft climbing above the intended altitude). In short, there is a intermediate mode (used for smooth transition between altitude capture and altitude hold) where the altitude capture is no longer armed and the plane is still acquiring the target

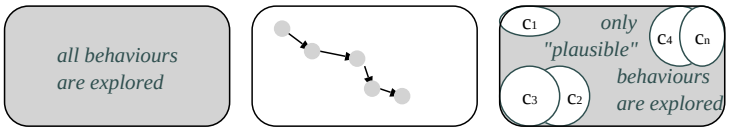

Fig. 4. Illustration of the input spaces that are explored in different analysis strategies.

altitude. Actions by the pilot during this period can cause the plane to miss the target altitude.

In an additional case study [16], the user interface of a simple mobile phone was analysed and usability problems were found, that relate to the interference of the processing of phone calls with the processing of SMS (Short Message System)[3] messages.

6 Context

An important role of the dialogue between the software engineer and the human factors or domain expert is to impose rules or constraints to further limit the possible counter-examples so that “interesting” traces can be discovered. In the case of the MCP model this constraint was achieved both by introducing a further model (see [9]) of the physical environment (in this case notions of altitude) and by adding properties to the initial CTL property to further constrain the possible circumstances of interest.

In typical analysis methods for interactive systems it is either assumed that the possible behaviour, as reflected in typical use of the device, is limited to those represented by a task — a description of what the user will do — or, alternatively, the device is explored by investigating all possible behaviours limited only by the physical device constraints. The approach described provides the additional opportunity to identify a representation of environmental or behavioural constraints that can be used to analyse sets of plausible behaviours and thereby provide opportunity to explore unexpected behaviours such as might occur through work arounds.

In other related work [11], model checking techniques were applied to analyse models of interactive systems, using the behaviour of the device under design, as well as the detailed behaviour of relevant environmental components. User models were either omitted, to achieve an exhaustive exploration of the space of user inputs; or the user was described by a normative task specification if the system was to be analysed with a particular way of performing a user task in mind [13].

Three possibilities for the analysis of models of interactive systems may be appropriate:

1. No assumptions about the user are made at all: as a result the model checker will explore all possible (combinations of) user inputs, see Figure 4a. The advantage

[3] SMS is a service for sending text messages to GSM (Global System for Mobile communication) mobile phones, which has become very popular in Europe.

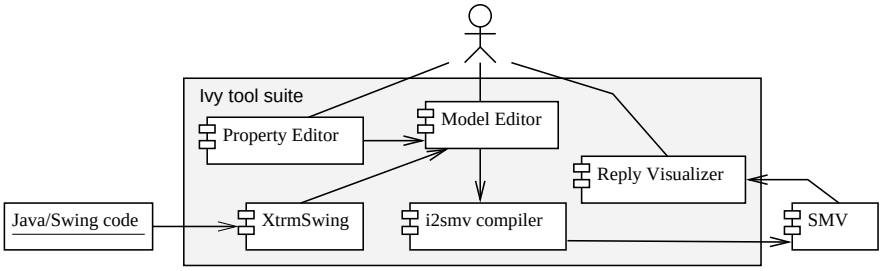

Fig. 5. IVY architecture

of this approach is that behaviours are considered that were not anticipated by the designer.

2. An instance of a particular user behaviour is modelled explicitly (see Figure 4b). This behaviour may be derived from a task description. The aim here is to focus on all possible responses of a device to a given user behaviour. The disadvantage of this approach is that it is not very flexible. It is hard to capture "similar" behaviours; for example, behaviour where the user performs additional actions that are not relevant for a given goal.
3. A set of general assumptions are made by the designer about the behaviour of under-specified components. Such assumptions can be interpreted as constraints. They "impose structure, restriction[s], or limitation[s]" on the model, and thereby force the model checker to explore only a sub-set of the input space during the analysis (see Figure 4c).

The advantage of this approach is that it is not necessary to engage in detailed modelling of user behaviour and user tasks, as such an approach would have at least two disadvantages. While user task models help focus the analysis of interactive system behaviour in relation to a particular task, the restrictive view they provide of the user and the world might rule out desirable design alternatives [17]. A second disadvantage, the description of complex decisions (for example, as might be required in dynamic function scheduling [18]) requires the development of complex user models. In the context of the model checking analysis, the addition of user models contributes to the state explosion problem.

7 Future Work

At this moment `i2smv` is a working prototype. It enabled the analysis of models during the case studies, but it does not yet adequately support the analysis process. In order to provide support, a tool is needed that can be used throughout the process, from modelling to interpretation of results. We are developing a tool suite with that purpose in mind. The architecture of the proposed tool suite is presented in Figure 5. In order for a tool to be effective it must support the different activities identified in section 4. The base architecture consists of a models' editor, the i2smv compiler, and a trace visualiser tool.

In order for this style of approach to be viable, it must itself be usable. Two of the most problematic aspects are building the models and expressing the properties. To address these issues we are considering two lines of work: developing a properties editor module, and performing reverse engineering of the user interface code. The properties editor module follows the ideas from [11]. Its main responsibility is to help the software engineer in expressing properties of the model. Regarding reverse engineering, we have started exploring the derivation of interactor models from Java/Swing code.

Traditional model based usability analysis techniques typically resort to some sort of graphical description of user interfaces. This can create problems when analysing software that resort to other presentation modalities other than graphical ones. For example, software for visually impaired users. More abstract modelling notations such as interactors can be of advantage since they move away from concrete details of the actual presentation. This is an area to be explore by using the approach during the development of a browser for visually impaired users.

8 Conclusions

In this paper it has been argued that a closer integration of usability issues into the software development life cycle would be a positive contribution to effective user interface design. Studies have shown that for large software development projects current software engineering practices are still far from guaranteeing satisfactory solutions. Part of the problem lies in the (lack of) design of the interaction between human and computer.

An approach for the integration of usability analysis into software development has been presented based on software model checking. The approach, based on analysing what systems should be implemented, differs from approaches such as [19, 20] where the goal is to analyse how the systems should be implemented. There are other approaches to the model checking of interactive system models, see section 6 in [9] for an overview. One of the main differences between those approaches and this work relates to the role properties play in verification. In all of the other approaches properties are seen as expressing some fact about the system which, if verified, should guarantee the system to be usable. Properties are typically statically determined, not for negotiation, whereas here properties capture potential properties of the user in the context of the system. The traces that are generated can be used to elicit scenarios representing examples of usage which thereby aid the development of modifications to the property or indeed new properties.

A prototype to support model checking based analysis of interactive systems in the context of the approach has also been presented. Case studies have been carried out that indicate the approach can be useful at early design stages to identify potential usability problems. The development of a tool suite to support the proposed approach which is based around the prototype has begun. Other directions for future work have also been put forward.

References

1. Preece, J., et al.: Human-Computer Interaction. Addison-Wesley (1994)

2. Hollnagel, E.: Human reliability analysis: context and control. Academic Press, London (1993)
3. Leveson, N.: Safeware: System Safety and Computers. Addison-Wesley Publishing Company, Inc. (1995)
4. Nielsen, J.: Usability Engineering. Academic Press, Inc (1993)
5. Lewis, C., Polson, P., Wharton, C., Rieman, J.: Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. In: CHI '90 Proceedings, New York, ACM Press (1990) 235–242
6. Duke, D.J., Harrison, M.D.: Abstract interaction objects. Computer Graphics Forum **12** (1993) 25–36
7. Ryan, M., Fiadeiro, J., Maibaum, T.: Sharing actions and attributes in modal action logic. In Ito, T., Meyer, A.R., eds.: Theoretical Aspects of Computer Software. Volume 526 of Lecture Notes in Computer Science. Springer-Verlag (1991) 569–593
8. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Transactions on Programming Languages and Systems **8** (1986) 244–263
9. Campos, J.C., Harrison, M.D.: Model checking interactor specifications. Automated Software Engineering **8** (2001) 275–310
10. McMillan, K.L.: Symbolic Model Checking. Kluwer Academic Publishers (1993)
11. Loer, K.: Model-based Automated Analysis for Dependable Interactive Systems. PhD thesis, Department of Computer Science, University of York (2003)
12. Schaad, A.: A Framework for Organisational Control Principles. PhD thesis, Department of Computer Science, The University of York, UK (2003)
13. Loer, K., Harrison, M.: Towards usable and relevant model checking techniques for the analysis of dependable interactive systems. In Emmerich, W., Wile, D., eds.: Proceedings of Automated Systems Engineering: ASE'02, IEEE Press (2002)
14. Campos, J.C., Harrison, M.D.: Using automated reasoning in the design of an audio-visual communication system. In Duke, D.J., Puerta, A., eds.: Design, Specification and Verification of Interactive Systems '99. Springer Computer Science. Springer-Verlag/Wien (1999) 167–188
15. Honeywell Inc.: SAS MD-80: Flight Management System Guide. Honeywell Inc., Sperry Commercial Flight Systems Group, Air Transport Systems Division, P.O. Box 21111, Phoenix, Arizona 85036, USA. (1988) Pub. No. C28-3642-22-01.
16. Campos, J.C.: Using task knowledge to guide interactor specifications analysis. In Jorge, J.A., Nunes, N.J., e Cunha, J.F., eds.: Interactive Systems. Volume 2844 of Lecture Notes in Computer Science., Springer (2003) 171–186
17. Vicente, K.: Cognitive Work Analysis. Lawrence Erlbaum Associates (1999)
18. Hildebrandt, M., Harrison, M.: The temporal dimension of dynamic function allocation. In S. Bagnara, S. Pozzi, A.R., Wright, P., eds.: 11th European Conference on Cognitive Ergonomics (ECCE 11), Istituto di Scienze e Tecnologie della Cognizione Consiglio Nazionale delle Ricerche (2002) 283–292
19. Havelund, K., Pressburger, T.: Model checking java programs using java pathfinder. International Journal on Software Tools for Technology Transfer **2** (2000) 366–381
20. Holzmann, G.J., Smith, M.H.: Automating software feature verification. Bell Labs Technical Journal **5** (2000) 72–87