113

Specification-Based Testing of Interactive Systems

Ian MacColl *

Software Verification Research Centre
School of Information Technology
The University of Queensland, 4072
Australia
ianm@cs.uq.edu.au

ABSTRACT Achieving correct user interface software is difficult because such software is complex, highly interactive, modeless, concurrent, graphical, and has user-based real-time requirements. In this paper I propose developing a framework for applying formal methods to testing of user interface software. I survey relevant work in the areas of formal development of user interface software and specification-based testing. I then outline a case study based research plan to extend an existing specification-based testing framework to incorporate multiparadigm user interface specifications.

KEYWORDS Human-Computer Interface, HCI, user interface, UI, formal methods, testing

1. INTRODUCTION

A user interface (UI) is some boundary between a computer system, comprised of hardware and software, and a human user. UI software is a significant component of contemporary systems and graphical user interfaces (GUIs) are now almost universal.

Confidence in the correctness of UI software is usually achieved by prototyping or by testing. Software correctness is commonly described as validation ("Are we building the right software?") and verification ("Are we building the software right?"). Prototyping can be used for validation and to verify that the UI meets usability requirements. Testing can also be used for validation (acceptance test) and to evaluate usability. Traditional software testing (unit, integration and system test) is the major method for verifying the correctness of UI software. Testing UI software is difficult however, because such software is complex, highly interactive, modeless, concurrent, graphical and has user-based real-time requirements.

Industrial strength tools have been developed to automate UI testing. These tools provide facilities such as capture/playback for test development, bitmap- and widget-based test evaluation, and textual and point-and-click scripting. Little support is available, however, for generating test inputs or expected outputs (oracles).

Formal methods are also used to develop correct software, including UI software. Fully formal development is advocated for projects such as life-critical software, where the cost of failure is greater than the additional cost of formal development (although industrial experience indicates that additional costs are offset by other savings).

Formal methods have been combined with software testing to provide specification-based testing techniques that use a (formal) specification to support testing. Little use has been made of these techniques for UI testing to date.

*Ian MacColl is supported by an Australian Postgraduate Award and a Telstra Research Laboratories Postgraduate Fellowship.

Human–Computer Interaction: INTERACT'97 S. Howard, J. Hammond & G. Lindgaard (editors)
Published by Chapman & Hall

This project aims to combine formal methods and software testing to develop specification-based testing techniques for UI software. Case studies will be used to extend an existing technique for specification-based testing to incorporate multi-paradigm UI specifications.

2. LITERATURE SURVEY

This section is a brief survey of work relevant to applying formal methods to testing of UI software, including using formal methods for UI development (§2.1) and for software testing (§2.2).

2.1 HCI and formal methods

Took identifies two paradigms for UI software: linguistic and agent-based (Took, 1990). The linguistic paradigm distinguishes lexical, syntactic and semantic aspects of UIs but formal linguistic notations, such as state-transition diagrams (Parnas, 1969), are not well suited to handling concurrency and attaching semantic information.

Agent-based architectures are an attempt to resolve the limitations of linguistic architectures as a basis for developing GUIs. The agent-based paradigm encapsulates data, functionality, and input and output within an 'agent' such as a button, a screen or an entire UI. The encapsulation is consistent with abstract data type and object-oriented (OO) software development.

In common with OO development, formal approaches to development of agent-based UIs must address both static and dynamic aspects of the system. Model- and property-based notations are well-suited to formalising static aspects whereas behaviour-based notations are preferable for dynamic aspects. Many formal, agent-based approaches use multiple notations of different styles, or extend an existing notation to encompass capabilities of a different style.

Behaviour-based notations, such as Petri Nets, are useful for modeling external behaviour but must be augmented to include static aspects of UIs. Palanque and Bastide, for example, use the Interactive Cooperative Objects (ICO) formalism to specify, verify and implement UIs (Palanque and Bastide, 1994). ICO provides an object-oriented framework for modeling static aspects of a system, and uses Petri Nets for modeling dynamic aspects.

Myers introduces a model for "highly interactive, direct manipulation, graphical user interfaces" that encapsulates interaction into interactor objects (Myers, 1990). The Esprit AMODEUS project has developed two formal models of interactors as encapsulations of a state, events to manipulate the state and a mechanism to present the state to users (Harrison and Duke, 1994). An interactor model developed at CNUCE in Pisa, Italy, uses the process algebra LOTOS, a property-based notation with behavioural capabilities. The CNUCE model is derived from work on graphics systems and input devices.

The second AMODEUS interactor model was developed at University of York, UK. The York model uses model-based notations such as Z and VDM augmented by CSP for behavioural aspects, and is a development of work extending Z to cover both static and dynamic aspects (Sufrin and He, 1990). The AMODEUS project has used interactor models for describing graphics systems and for analysing interactive systems. Current work includes integration of formal descriptions of users and systems in an interactive framework and investigation of scaling up formal development of UI implementations.

Much of the work applying formal methods to HCI emphasises reasoning and analysis rather than implementation. Hussey and Carrington use software patterns (Gamma et al., 1994) to informally transform interactor-based specifications, written in Object-Z (Duke et al., 1995), into widget-based designs (Hussey and Carrington, 1996a, Hussey and Carrington, 1996b). Object-Z enhances the model-based capabilities of Z with encapsulation constructs and includes operators for expressing dynamic aspects such as concurrency and communication between objects.

Partial specifications can be used to accommodate multiple views of a software specification, e.g., UI and functionality views. Jackson advocates partial specifications (in Z) as means to achieve clear and concise specifications with a modular structure to ease maintenance (Jackson, 1995). Zave and Jackson use augmented first order logic as a common representation for multiparadigm specifications (Zave and Jackson, 1996).

This section provided an overview of formal inter-

actor models and object-oriented variants, and introduced partial and multiparadigm specification techniques. The next section is a survey of work on using formal methods to improve testing.

2.2 Formal methods and testing

Software testing is conventionally classified into behavioural and structural testing: behavioural testing emphasises testing against the functional requirements and structural testing emphasises testing based on the internal structure of the software.

The precise definition of behaviour provided by a formal specification is particularly useful for behavioural testing as a basis for deriving test inputs and as a means for determining the output expected from a particular test input.

The application of formal methods to testing, also called specification-based testing (SBT), is an active research topic. Gaudel, for example, summarises program testing based on algebraic specifications, describing methods for selecting a finite subset of an exhaustive test set (Gaudel, 1995). For model-based notations, Hörcher and Peleska show how Z specifications can be used to derive test input data and to automatically evaluate test results (Hörcher and Peleska, 1995), and Stepney reports on a method for deriving tests from Z specifications by a process of systematic abstraction (Stepney, 1995).

The Test Template Framework (TTF) is a formal, abstract model of testing, used to derive a hierarchy of test information, including test inputs and outputs, from a formal specification (Stocks and Carrington, 1996). The framework uses the Z notation and incorporates both traditional and new testing strategies.

It appears that only one group has applied formal methods to UI testing (Yip and Robson, 1991). Three notations are used to specify UI software: state-transition diagrams to express interaction sequences, a model-based notation similar to Z to define the state transitions and an algebraic notation for reasoning. Tests are derived from the state-transition diagrams, and the model-based specifications are used to determine expected outputs. The method for combining the notations appears informal and incomplete. Also, this work has not been updated to take account of subsequent developments in specification-based testing.

This section has outlined recent work on specification-based testing, including UI specification-based testing. The next section concludes this survey.

2.3 Conclusions

Formal methods have been applied to the development of UI software and to improve software testing. The Z notation (Z Base, 1995) is a common theme to much of the work surveyed above. Interactor-based formal specifications map to widget-based designs and implementations. Multiple paradigm and view-based specification techniques can be used to connect static and dynamic aspects of UI specifications and to connect UI and functionality specifications. The Test Template Framework (TTF) provides a framework for testing based on model-based specification notations but does not address sequencing issues such as UI behaviour. There is also no tool support for using the TTF, a requirement for automation of a potentially rapidly changing area such as UI testing.

Issues not addressed in this survey include: relationship of prototyping and specification; formal methods in an iterative, user-centred development context; dialogue separation, refinement and abstraction; and top-down and bottom-up approaches to writing specifications.

3. RESEARCH PLAN

My thesis project is development of a framework to support specification-based testing of UI software.

My work to date has involved a literature search to select this topic (alternatives were in the areas of formal methods, OO and functional programming, and software testing). I have also started a case study specifying the Parker Brothers game Boggle in Z.

I propose to conduct three case studies to extend the TTF to support multiparadigm UI specifications. This will require accommodating behaviour-based notations such as CSP and Statecharts, and identifying UI-specific testing strategies. I will then evaluate the extended TTF in a formal experiment comparing TTF-derived UI testing with non-TTF-derived UI testing.

References

Duke, R., Rose, G., and Smith, G. (1995). Object-Z: A specification language advocated for the description of standards. *Computer Standards & Interfaces*, 17:511–533.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Gaudel, M.-C. (1995). Testing can be formal, too. In *TAPSOFT '95: Sixth International Joint Conference on Theory and Practice of Software Development*, volume 915 of *Lecture Notes in Computer Science*, pages 82–96. Aarhus, Denmark.

Harrison, M. and Thimbleby, H., editors (1990). *Formal Methods in Human-Computer Interaction*. Cambridge University Press.

Harrison, M. D. and Duke, D. J. (1994). A review of formalisms for describing interactive behaviour. In Tayler, R. N. and Coutaz, J., editors, *Software Engineering and Human-Computer Interaction. ICSE '94 Workshop on SE-HCI: Joint Research Issues*, volume 896 of *Lecture Notes in Computer Science*, pages 49–75, Sorrento, Italy. Springer-Verlag.

Hörcher, H.-M. and Peleska, J. (1995). Using formal specifications to support software testing. *Software Quality Journal*, 4(4):309–327.

Hussey, A. and Carrington, D. (1996a). Using Object-Z to compare the MVC and PAC architectures. In Roast, C. R. and Siddiqi, J. I., editors, *Proceedings of BCS-FACS Workshop: Formal Aspects of the Human Computer Interface*, pages 45–60, Sheffield, UK.

Hussey, A. and Carrington, D. (1996b). Using Object-Z to specify a web browser interface. In Grundy, J. and Apperley, M., editors, *Proceedings OzCHI*, pages 236–243, Hamilton, New Zealand.

Jackson, D. (1995). Structuring Z specifications with views. *ACM Transactions on Software Engineering and Methodology*, 4(4):365–389.

Myers, B. A. (1990). A new model for handling input. *ACM Transactions on Information Systems*, 8(3):289–320.

Palanque, P. and Bastide, R. (1994). Petri Net based design of user-driven interfaces using the Interactive Cooperative Objects formalism. In Paterno, F., editor, *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems*, pages 383–300, Bocca di Magra, Italy.

Parnas, D. L. (1969). On the use of transition diagrams in the design of a user interface. In *Proceedings of the 1969 National ACM Conference*, pages 739–743. Association for Computing Machinery.

Stepney, S. (1995). Testing as abstraction. In Bowen, J. P. and Hinchey, M. G., editors, *ZUM '95: The Z Formal Specification Notation*, volume 967 of *Lecture Notes in Computer Science*, pages 137–151, Limerick, Ireland. Springer-Verlag. Proceedings of the Ninth International Conference of Z Users.

Stocks, P. and Carrington, D. (1996). A framework for specification-based testing. *IEEE Transactions on Software Engineering*, 22(11):777–793.

Sufrin, B. and He, J. (1990). Specification, analysis and refinement of interactive processes. In (Harrison and Thimbleby, 1990), chapter 6, pages 153–200.

Took, R. (1990). Putting design into practice: Formal specification and the user interface. In (Harrison and Thimbleby, 1990), chapter 3, pages 63–96.

Yip, S. W. L. and Robson, D. J. (1991). Graphical user interfaces validation: A problem analysis and a strategy to solution. In Shriver, B. D., editor, *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, volume II, pages 91–101.

Z Base (1995). Z notation: Version 1.2. `ftp://ftp.comlab.ox.ac.uk/pub/Zforum/ZSTAN/versions/` (15 Jan 97).

Zave, P. and Jackson, M. (1996). Where do operations come from? A multiparadigm specification technique. *IEEE Transactions on Software Engineering*, 22(7):508–528.