Component-based Systems as an aid to Design Validation

Peter Henderson
(P.Henderson@ecs.soton.ac.uk)
Declarative Systems and Software Engineering Group
Department of Electronics and Computer Science
University of Southampton
Southampton, UK.
SO17 1BJ

Robert Walters
(R.J.Walters@ecs.soton.ac.uk)
Declarative Systems and Software Engineering Group
Department of Electronics and Computer Science
University of Southampton
Southampton, UK.
SO17 1BJ

Abstract

There is a continuing need for software engineers to better quality systems more quickly. Component based technologies promise to make this possible, but modern systems are too complex for a full analysis of their behaviour to be practical. We propose that a reasonable alternative is analyse abstract models of the essential features of a system. Since these models are abstract, they need contain only details relevant to the aspect of the system under consideration. Consequently they can be small enough to be constructed quickly and analysed thoroughly using formal methods.

Tools are required which are accessible to the novice but remain powerful enough to build models with a formal foundation so that they can be used by system designers who have limited expertise in the use of formal methods. We propose our tool, RolEnact as a candidate for this rôle.

1 Introduction

There is a continuing need for software engineers to build better quality systems more quickly. An obvious route towards this goal has always been to make effective use of existing code. Component technologies such as COM [1, 2], CORBA [3] and Enterprise Java Beans [4] are the primary technologies which make this possible today.

Using these techniques, huge systems can be assembled with a minimum of time and effort by integrating a collection of existing components [5]. Some of the components will be internally generated items for which source code and complete documentation is available, but if the advantages of component based technologies are to be exploited, other components will be "bought-in". The use of components from outside suppliers means that a formal model of the entire system cannot be constructed as the suppliers of components are unlikely to release detailed implementation information.

Validating these systems presents a different type of problem from that of a system built using traditional methods [6, 7]. There may be internally generated code which needs to be verified but for bought-in components this responsibility rests with the suppliers. The task of the designer becomes identifying the components required and fitting them together in such a way that the completed system satisfies the requirements. Verification of a design is concerned with understanding how the components in the system communicate and interact [8]. The designer needs tools and methods which are able support this activity. Constructing executable models is one method which can be used, if the effort required to build the model is reasonable. The task of analysing these models can be kept under control by using an appropriate (abstract) view of the system.

2 Tools for simulation and design verification of component based systems

The new style of constructing software from components requires revised techniques for the evaluation and verification of designs. It is no longer possible to evaluate a design based upon an appreciation of the internal operation of the constituent parts. Instead, designs have to be evaluated based upon an understanding of the

communications (or interactions) between components. The behaviour of the components themselves is, effectively fixed (if not always accurately described). Consequently, the design of other parts of the system, such as any code written to connect components must be able to accommodate any undesirable features in behaviour of components [11].

We observe that in hardware development, building systems from components is much more widespread than in software and extensive use is made of modelling and simulation to evaluate designs. We propose that, if software engineers are to maximise the benefits of component technologies, we should consider adopting a development style more like that used by the hardware community. In particular, we should consider using more modelling and simulation of system designs [12].

Mature tools and widely accepted techniques exist for modelling (simulation) and evaluation of hardware projects [13, 14], but this is not yet the case for software. We do have a selection of tools and methods available which could be used for this type of work from simple diagramming notations and system description languages [15] to complete systems for the construction, execution and evaluation of proposed systems [16]. These tools come from two areas: Formal methods and other design techniques and notations.

Design notations and techniques either suffer from a lack of a suitable formal underpinning or where this exists, it is not enforced. As a consequence, where appropriate simulation tools are available, creating and running simulations of a design requires considerable effort, whilst at the same time driving out the flexibility which makes these notations appealing.

This lack of a formal foundation is not a problem with tools and techniques from formal methods. They have been developed for the construction of software which can be claimed to be "correct" in some formal way. However, this brings with it a need for detail, formality and rigour which is not justified for general purpose software and makes the tools both difficult to use on large systems and inaccessible to users.

The evaluation of designs should extend beyond a simple matching of interface specifications into an evaluation of whether the assembled components interact and communicate in the desired manner - the fact that a pair of components *can* be connected does imply that it makes sense to do it. Executable models [17] can help with this task in a way which cannot be achieved using static methods. However, for these techniques to be useful, the designer must be able to construct, execute and analyse models quickly and easily.

Once constructed, these models can be used for a variety of activities from an informal examination of the operation of the proposed system to verification of the system using model checking techniques [18, 19], as a mechanism to pass on understanding of the system or as a template for the coding of the real implementation.

3 RolEnact

RolEnact [9, 10] is a tool for constructing executable models. A RolEnact model consists of a collection of communicating elements called Roles each of which has state and a number of events which, when they occur cause changes of state. For an event to occur, each of the Roles involved must be in the required "before" state. After an event, each of the Roles involved changes to a specified "after" state. In a running model, each Role has a unique name (derived from the type of the Role) and window in which the state of the Role and a list of the actions which the Role is presently able to initiate are displayed. To run the model, the modeller selects an event and makes it happen by "double clicking" on the name of that event in the window of the appropriate Role. The model then changes the states of each of the Roles involved in that event and then re-evaluates the lists of events that each Role is able to initiate.

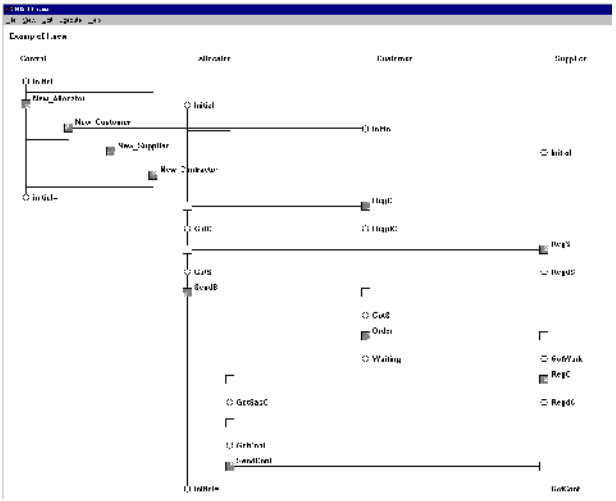

Figure 1: A RolEnact model displayed as a diagram

RolEnact has a simple interface which allows the model to be viewed as a list of events or as a "RAD-like" [20] diagram (which is similar in concept to a UML sequence diagram [15]). Roles and events may be added to and deleted from the model using a series of dialog boxes. Figure 2 shows the dialog box for adding a new

event showing how the type of event, the Roles and the before and after states are selected from those available.

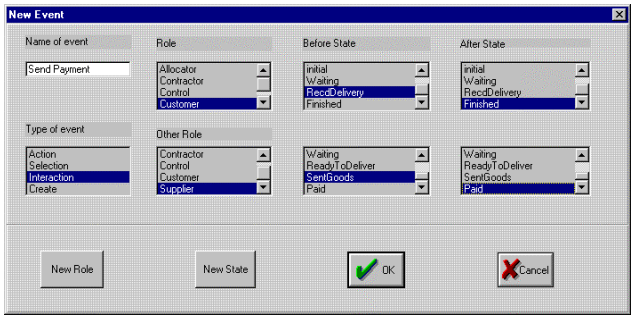

Figure 2: Dialog box for adding a new event to a model

The modeller may execute the model at any time. By executing a model, the modeller is able to step through sequences of events and experiment to see how the system behaves. Figure 3 shows the example system running.

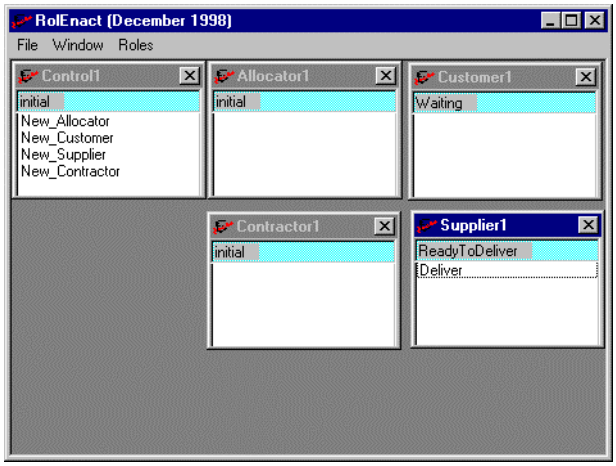

Figure 3: A RolEnact model running

The diagrams in conjunction with the dialogue box guided method for the construction of models is particularly appealing to the inexperienced modeller for whom it is important to be able to begin to build models and execute them with a minimum of effort.

The more experienced modeller may choose to write code as described in [10].

RolEnact is free and may be downloaded from http://www.ecs.soton.ac.uk/~ph/RolEnact.

4 Components and RolEnact

Roles interacting in a model map to the interacting components of the target system. The various actions and interactions possible by components are reduced to four types of event in RolEnact.

- Actions: activities that do not involve another Role - unilateral activity by a component.
- Interactions: events shared by two Roles - communication between connected components.
- Creations: creation or a new instance of a Role - the instantiation (or location and identification) of another component.
- Selections: establishing new connections between Roles as a side-effect of an interaction - the location of (and communication with) an instance of a required type of component.

Armed with a completed RolEnact model of a proposed system, the developer has a template for the interactions between the various parts of the system. This can be used to design the interfaces of the components. The final details including whether interactions are to be COM events, Remote Method invocations or some other form of communication and whether each represents a series of events or just one is for the developer to decide.

The purpose of the model is to define the system and the interactions which will occur in the final system *in abstract terms* so it is appropriate that the implementation details are not included.

5 Conclusion

It is undisputed that building fault free software systems is a considerable problem for the software industry to solve. At present, the best solution where reliability and correctness is essential is formal methods. However, these are costly and difficult to use in their present state of development, and they cannot be readily applied to component based systems because they need an intimate knowledge of every part of a system. We propose that, a reasonable alternative is to construct models of parts and views of a system which are perceived as important which may then be validated. This approach does not provide proof of correctness but can help the developer to reason about proposed designs, find errors and build confidence in the system.

An executable model provides a template for the construction of the real system without dictating the details as well as a tool for the rapid explanation of the underlying principals of the system to new developers joining the team.

However, to build these executable models, there is a need for a new style of tool. These new tools should provide the developer with facilities for the construction, execution and verification of models. At the same time, these tools must be easy to use by the expert and novice alike. We propose RolEnact as one potential candidate for this type of task.

6 References

[1] D. Box and G. Booch, *Essential COM*: Addison Wesley, 1998.

[2] R. Sessions, *COM and DCOM - Microsoft's Vision for Distributed Computing*: Wiley Computer Publishing, 1998.

[3] Object Management Group, "Common Object Request Broker: Architecture Specification," .

[4] A. Thomas, "Enterprise JavaBean Technology," Patricia Seybold Group, White Paper prepared for Sun Microsystems Inc December 1998.

[5] R. J. Allen and D. Garlan, "A Fomal Basis for Architectural Connection," *ACM Transactions on Software Engineering and Methodolgy*, 1997.

[6] M. Barjaktarovic and et al, "Formal Specification and Verification of Communication Protocols Using Automated Tools," presented at First IEEE International Conference on Engineering of Complex Systems, 1995.

[7] D. Garlan and et al, "Architectural Mismatch, or, why it's hard to build systems out of existing parts," presented at ICSE, 1995.

[8] K. Sullivan and J. C. Knight, "Experience Assessing an Architectural Approach to Large Scale Reuse," presented at 18th International Conference on Software Engineering (ICSE-18), 1996.

[9] P. Henderson and R. J. Walters, "System Design Validation Using Formal Methods," presented at Tenth IEEE International Workshop on Rapid System Prototyping (RSP99), Clearwater, Florida, 1999.

[10] K. T. Phalp, P. Henderson, G. Abeysinghe, and R. J. Walters, "RolEnact - Role Based Enactable Models of Business Processes," *Information And Software Technology*, vol. 40, pp. 123-133, 1998.

[11] P. Henderson, "Laws for Dynamic Systems," presented at International Conference on Software Re-Use (ICSR 98), Victoria, Canada, 1998.

[12] C. A. R. Hoare, "How did Software get to be so reliable without proof," presented at 18th International Conference on Software Engineering (ICSE-18), 1996.

[13] M. M. K. Hashmi, "Extending VHDL for Interface and System Specification," presented at Fall VIUF - VHDL for Power Users, 1998.

[14] IEEE, "IEEE Standard VHDL Language Reference Manual," . New York: IEEE, 1994.

[15] M. Fowler, K. Scott, and G. Booch, *UML Distilled - Applying the standard Object Modelling language*: Addison Wesley, 1997.

[16] M. J. Butler, "An Approach to the Design of Distributed Systems with B AMN," presented at 10th International Conference of Z Users (ZUM'97), Reading, 1997.

[17] A. Gravell and P. Henderson, "Executing formal specifications need not be harmful," *Software Engineering Journal*, vol. 11, 1996.

[18] E. M. Clarke and e. al, "Model Checking and Abstraction," *ACM Transaction on Programming Languages and Systems*, 1994.

[19] G. J. Holtzmann, "The Model Checker SPIN," *IEEE Transactions on Software Engineering*, vol. 23, 1997.

[20] M. A. Ould, *Business Processes - Modelling and Analysis for Re-engineering and Improvement*: John Wiley and Sons, 1995.