System Design Validation Using Formal Models

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

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

Abstract

Formal methods are a nice idea, but the size and complexity of real systems means that they are impractical. We propose that a reasonable alternative to attempting to specify and verify the system in its entirety is to build and evaluate an abstract model(s) of aspects of the system that are perceived as important.

Using a model will not provide proof of the system, but it can help to find shortcomings and errors at an early stage. Executing the model should also give a measure of confidence in the final product.

Many systems today are built from communicating components so that the task of the developers is becoming fitting these components together to form the required system. We show how a formal model can be sympathetic to this type of architecture using our tool, RolEnact and explain how this may be related to a COM implementation.

1 Introduction

Nobody would deny that widespread use of formal methods in software development would lead to a great improvement in quality and reliability of systems. However, the present state of the art in formal methods is such that, except for safety critical systems, the cost of using them in time and effort is prohibitive.

Although a complete proof of a system may be impractical, this is not a reason to plough into coding. What is needed is a practical approach to system development which avoids the problems of formal methods, but can still bring with it some of the advantages [2]. One possibility is to build an abstract model of the system which, although it cannot lead to a proof of the system, will allow examination of the operation of at least parts of the system and the identification of fundamental flaws [10].

We believe it is reasonable to build and examine an abstract model of a proposed system. Methods familiar to formal methods like model checking [12] may be applied to this examination of the model as well as less formal evaluation such as manual inspection of an executable version of the model [6]. The advantage of this approach is that the size of the analysis task may be contained by the choice of an appropriate view (or abstraction) of the problem and errors or other shortcomings of the system can be discovered early in the development process [1].

Although it may not look like the final system, such a model could be thought of as a prototype, not of the system as a whole, but of some aspect of the system. Such a model may be used to examine how the system will behave in operation in the same kind of way as a prototype of a user interface might be tried upon the intended users. This will provide the development team with insight into problems in the proposed system and confidence in the final solution chosen. These models need to be easily constructed so that early and interim versions can be discarded with minimal cost.

Although this type of activity has traditionally been unpopular with software engineers, it is well established other areas. For example, using software models to simulate and explore the behaviour of electronic hardware in advance of implementation is well established and there

are mature languages and tools available for this work [7][13].

Once work on the model is completed, it may be dismantled and used as a template for the construction of the real system.

The arrival of software component technologies means that progressively more systems are being developed by integrating existing (legacy) and bought-in components whose behaviour is essentially fixed [5][9][18]. This means that the task of understanding how components behave and interact is more important that ever. RolEnact is a process modelling tool which is suited to the task of examining and understanding such communicating systems. In its latest form it has a graphical interface which enables the easy and fast generation of models allowing even the novice the user to concentrate on the problem under consideration.

2 The Three layer model

Architectures such as COM [3], CORBA [14] and Enterprise JavaBeans [20] are emerging for the construction of large systems using components. Typically, these systems conform to the three layer model [17] where the major components of the system form one layer which is separated from the user interface layer by a third layer concerned with business rules.

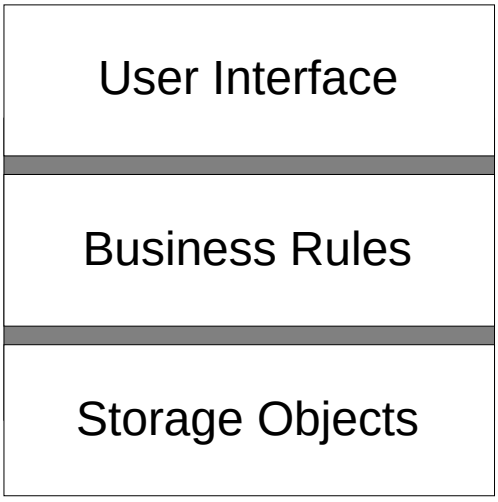

Figure 1: The Three Layer Model

The storage layer includes the back-office systems, such as database management systems, backup systems, and permanent storage. Typically this layer of a system will be provided by a suite of bought-in programs. (Sometimes this layer is split into a local or client layer and a global or server layer.)

The user interface layer comprises the applications and objects visible to the users of the system. These applications are light weight in that they contain little, if any, knowledge or understanding of the underlying task of the system. They are also subject to change.

The business rules layer connects the other two and is the layer in which we are interested. It is the rules encapsulated in this layer which define how the system operates.

The design of one of these systems presents different problems from those encountered with a conventional software development project. The task is less to do with writing functions and constructing objects and more concerned with how (bought-in) components will behave and interact. It is important to understand the communications between the various parts of the system in order to ensure that new components have the correct behaviour and interfaces. As with all systems, the earlier errors and bugs are discovered, the more easily and cheaply they may corrected [1].

3 Why model with formal systems?

Formal methods promise to provide the means to build robust systems whose behaviour can be described as "correct". However, aside from applications where safety is an issue, they have failed to make an impact. One factor which contributes to this is that, in all but the simplest of systems, the task of forming proofs and exhaustive searches is daunting.

However, provided they are constructed using suitable abstractions from the detailed problem, models can be a reasonable size. Indeed one of the criteria for selecting an abstraction to be used will be that it produces a model of manageable size. This allows the models to be exercised extensively, if not exhaustively enabling the location and correction of faults in the architecture of the system before development of the final system starts.

Building a model of a proposed system using a formal modelling tool will give the developer a better understanding of how the system should operate. Executing the model gives the developer the opportunity to experiment with the architecture, iron out bugs and confirm that the proposed solution meets the requirements. Errors and bugs discovered at this stage can be corrected with a minimum of cost and delay to the project.

Once the model of the system has been built and verified, the development of the real system should proceed more smoothly than it would have otherwise. This is because flaws in the high level design will have already been

discovered and corrected. Without the verification exercise, these shortcomings (which often are the unexpected consequence of interactions between components in the system rather than errant behaviour of components themselves) will not be evident until enough of the system has been completed for the initial versions to be set up and tested.

4 RolEnact

RolEnact is a process modelling tool built using the Enact [8] modelling language. RolEnact is described fully elsewhere [16] so here we give just an outline. An improved version of RolEnact/RaDraw is now available which may be downloaded as a single package from: http://www.ecs.soton.ac.uk/~ph/RolEnact.

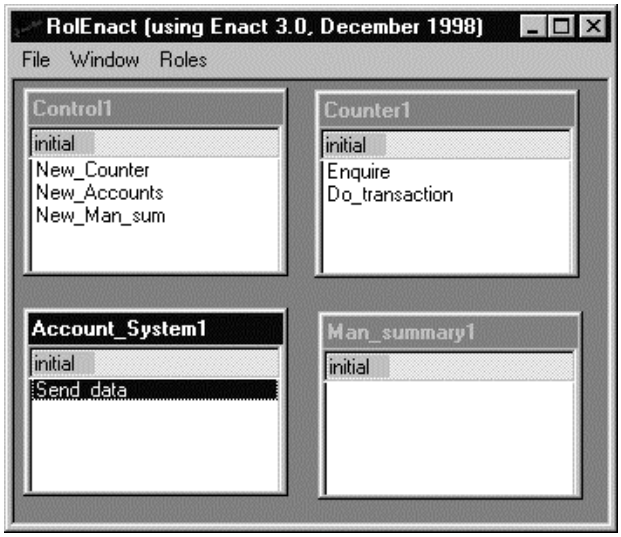

Figure 2: A running RolEnact model of a simple version of the Bank example

A RolEnact model consists of a number of Roles each of which has state and appears on the screen as a separate window within the RolEnact application. The model in Figure 1 has four Roles running: one instance each of the Control, Counter, Account_system and Man_summary Roles. The title bar of a Role's window shows its name (which is derived from its type by suffixing an instance number since there may be more than one instance of each Role). The first (coloured) line in a Role's window indicates its state and is followed by a list of the events which the Role is currently able to initiate.

All Roles commence in an "initial" state. They are able to change state by means of events. An event may involve one, two or more Roles. Each event is initiated by a Role in the system and has a "before" state and an "after" state for each of the Roles involved. Before an event may take place, each of the Roles involved in the event must be in the "before" appropriate state. After the event, each of the Roles changes to the appropriate "after" state.

Early versions of RolEnact required the modeller to write Enact code. Later versions introduced a streamlined version of the code which could be translated into the executable code using a conversion tool. More recently, we have been working on a tool, RaDraw to generate RolEnact models from a graphical interface which is able to draw a RAD-like [15] representation of the model. A Role Activity Diagram is similar in concept to sequence diagrams in UML [4].

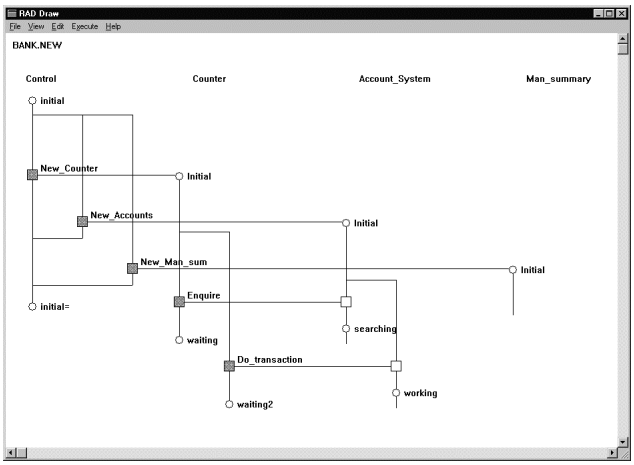

Figure 3: The RolEnact model of the Bank example during construction using RaDraw

Figure 3 shows part of the Bank model during construction using RaDraw. In the diagram, each Role of the model occupies a column. The states of a Role are indicated by small circles which are labelled with the name of the state. Events in which a Role takes part are shown as small squares linked to their "before" and "after" states by vertical lines. Where an event affects more than one Role, the boxes for the event in each of the Roles are linked by an horizontal line. The shaded square of an event indicates the Role which initiates the event. The latest Event to be added to the model is "Do_transaction", an event which may be initiated by the Counter Role when it and the Account_System are in state "initial". Firing this event causes the Counter to change to state "waiting2" and the Accounts_system to move to state "working".

In addition to drawing a diagram showing the model, RaDraw is able to display the model as RolEnact code. The menus of the tool provide dialog boxes with which the modeller is able to add new Roles, States of Roles and Events to the model as well as calling RolEnact to run the model.

Roles are equivalent to finite state machines, so that a RolEnact model amounts to a collection of finite state machines with synchronised state transitions, hence although easier to use RolEnact has an expressive power equivalent to that of CSP [11].

5 Mapping RolEnact to COM

COM [3][17] is Microsoft's Component Object Model and underpins OLE and ActiveX. It defines an architecture for the assembly of systems from components and how those components will communicate [19].

In order to use a RolEnact model to assist with the task of implementing a real system, the essential features of the model must relate to features that will exist in the real system. A Role in a RolEnact model corresponds to a component (or application) in the implementation. There are detail differences between the concepts of Role and component, but for the purpose of building an abstract model, these differences are unimportant.

The tables below describe the effects of the four types of event that exist in RolEanct and how these events may be interpreted in a system implemented using COM.

In a RolEnact model:

Creation	Make a new instance of a Role
Selection	Locate and communicate with *any* instance of a type of Role
Interaction	Communicate with a known instance of a Role
Action	Unilateral change of state

Figure 4: Summary of RolEnact Events

There are subtle differences between the meanings of types of event in RolEnact and the features of COM, but at the high levels of abstraction at which these models are built, these are not important and the model provides a good template for the construction of a real system.

In a COM implementation of a system derived from a RolEnact model:

Creation	Make a new object provided by a COM server
Selection	Send an event to/make a procedure call to any existing object on another COM server
Interaction	Send an event to/make a procedure call to a known instance of an object
Action	Internal operation of this process

Figure 5: Summary of Relationship between Events in a RolEnact Model and features of an Implementation using COM.

6 Benefits of using modelling during design

In the types of system that we are considering, the applications in the storage layer are likely to be large data management systems. Often these systems will be commercial packages or legacy systems. Either way, the developer building a new system will have to use the interfaces provided by these systems as they are.

The user interface layer is quite different from the storage layer in that these applications will be light weight and the subject of constant change as user requirements develop. However, there may be many instances of these applications installed around an organisation, so it is essential that as new applications (and new versions of existing applications) are deployed, the existing applications continue to work. Thus, although these applications are quite different from those in the storage layer, they present the developer with similar problems, at least in the short term.

RolEnact is able to model the behaviours of these existing components and affords the developer the opportunity to experiment with alternative solutions to the problems posed when they have to be assembled into a single working system. The effort involved in building and evaluating a model is small. Consequently, the investment required to evaluate possible system architectures is also small. When a model highlights problems with a proposed solution, the cost associated with amending the architecture or adopting an alternative approach is reasonable. The decision about how to implement the system may be based upon an evaluation of several competing solutions.

Simple examination of the diagram of the model highlights issues concerning which Roles need to interact, and running the model provides insight into the behaviour of the model and confidence that the proposed architecture is acceptable.

7 Conclusion

The arrival of component technologies has accelerated the shift towards building software systems using existing and bought-in components. The behaviour of these com-

ponents is essentially fixed so that the development task is becoming one of finding ways to fit these components together.

Confidence about the behaviour of these component based systems is important, but they are typically so large that applying formal methods would be impractical.

Hence, we propose that a reasonable alternative is to build an abstract model of the system using a formal modelling tool. The model can then be used to perform some "light weight" analysis of the system, which although not rigorous is preferable to none at all. Building such a model also gives the developer a template for the construction of the real system as well as documenting how the developers intended the system to operate.

Our tool, RolEnact is one possible choice for this type of work and we have shown how a RolEnact model might be related to a real system to be developed using COM.

References

[1] Barjaktarovic, M.; et al.: *Formal Specification and Verification of Communication Protocols Using Automated Tools*. Proceedings of the First IEEE International Conference on Engineering of Complex Systems, IEEE Computer Society Press, 1995.

[2] Beitzer, B.: *Cleanroom process model: A critical examination*. IEEE Software, 1997.

[3] Box, D.: *Essential COM*. Addison Wesley 1998.

[4] Fowler, M.: *UML Distilled – Applying the standard Object Modelling language*. Addison Wesley, 1997.

[5] Garlan, D.; et al: *Architectural Mismatch, or, why it's hard to build systems out of existing parts*. ICSE 1995.

[6] Gravell, A.; Henderson, P.: *Executing formal specifications need not be harmful*. Software Engineering Journal, Vol. 11, No.2, IEE 1996.

[7] Hashmi, M. M.; Bruce, A. C.: *Design and Use of a System-Level Specification and Verification Methodology*. Proceedings of Euro-DAC '95, Brighton, Great Britain, 1995.

[8] Henderson, P.: *Enact User Manual*. Available at http://dsse.ecs.soton.ac.uk/~ph.

[9] Henderson, P.: *Laws for Dynamic Systems*. International Conference on Software Re-Use (ICSR 98), Victoria, Canada, June 1998.

[10] Hoare C. A. R.: *How did Software get to be so reliable without proof*. Keynote address at the 18th International Conference on Software Engineering. IEEE Computer Society Press, 1996.

[11] Hoare, C. A. R.: *Communicating Sequential Processes*. Prentice-Hall 1985.

[12] Holtzmann, G. J.: *The Model Checker SPIN*. IEEE Transactions on Software Engineering, Vol. 23, No. 5, 1997.

[13] *IEEE Standard VHDL Language Reference Manual*. IEEE Std 1076-1993. IEEE, New York, 1994.

[14] Object Management Group: *Common Object Request Broker: Architecture Specification*. Available at http://www.omg.com.

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

[16] Phalp, K.; Henderson, P.; Abeysinghe G.; Walters, R. J.: *RolEnact - Role Based Enactable Models of Business Processes*. Information And Software Technology, Vol. 40 No.3 pp 123-133, 1998.

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

[18] Shaw, M.; Garlan, D.: *Software Architecture – Perspectives on an emerging discipline*. Prentice Hall, 1996.

[19] Sullivan K.; Knight J. C.: *Experience Assessing an Architectural Approach to Large Scale Reuse*. Proceedings of ICSE-18, IEEE Computer Society Press, 1996.

[20] Thomas A.: *Enterprise JavaBeans Technology White Paper*. Prepared for Sun Microsystems, Inc. by Patricia Seybold Group. Available at: http://java.sun.com:8081/products/ejb/white_paper.html, 1998