RICES: Reasoning about Information Consistency across Enterprise Solutions

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

Introduction

RICES is a four year project based at the University of Southampton working in collaboration with ICL and DERA. It is concerned with understanding and seeking solutions to the problems that accrue when large systems are integrated. The particular focus of this project is how to manage inconsistent data.

Background

In a recent interview [19], John Taylor, Director General of Research Councils, described the UK's progress towards the information utility, which is becoming as essential as the utilities of water, gas and electricity. He gave a very positive view of the UK's participation in this global phenomenon, but warned "As a mass of such information [from many sources] is gathered, some will inevitably be wrong. The consequences of this will be unpredictable, unpleasant and in many cases invisible. This data-pollution will become a key cyber-green environmental issue."

In everyday life we are bombarded with information from various sources. Based on this information, we form our view of the world and decide upon our actions. Much of this information is incomplete and contains inaccuracies. However, we are good at maintaining the data that we retain and (generally) are able to function normally despite shortcomings in the data we receive - we apply "common sense". RICES is a project which is looking at how we might be able to replicate this type of behaviour when similar problems are encountered in computer systems.

To some extent these problems may be addressed by transaction processing (TP) systems, but TP systems are typically too tightly coupled to allow easy evolution. They implement strong consistency through transactions and protocols such as two-phase-commit. We have to question the appropriateness of two-phase-commit and other synchronous transaction models, partly because of the consequences for reliability and performance but, in particular in the context of the information utility, because of the involvement of human participants.

Hence the move to more loosely coupled application systems integration, away from Corba [20] and distributed objects [2] and back to message based or process interaction based interaction with technologies such as MQM [6] (supported by both IBM, with MQ Series, and Microsoft, with MSMQ). Such approaches introduce asynchrony and concurrency, which introduce a temporal dimension into reasoning about information consistency. Enterprise Application Integration (EAI) tools provide *mechanism*(s) for integration but there is little formal underpinning – reasoning about validity of a proposed solution is largely left to the intuition of the designer. Hence the need to be able to

- Categorise component application system properties in terms of information storage & access
- Reason about adding new components to existing configurations
- Reason about migration of information (and functions)
- Reason about consistent derivations of information
- Reason about the consistency properties visible to an observer with access to several components

Nature of the Solution to be Researched

System Level Reasoning

Systems Level Reasoning is the key to quality in large systems. All engineers do this, whatever their basic technological skills. They use a wide range of diagramming and modelling techniques and adapt the most appropriate ones to their own purposes. In presenting a proposed solution, integrated from existing systems, they reason that their solution is valid based on relatively informal arguments [1, 5, 8, 12, 14-16, 21]. The reason for this informality is because very large systems, especially those which are integrated across enterprises from existing (continuing to operate) systems, do not lend themselves to formalisation. Languages like UML and its derivatives [6, 7], still popular in Information System Design, are not really up to the task of formulating an argument about the ability of a large system to deliver a required property despite the inconsistency of its information sources.

Recent advances in model checking have demonstrated reasonable success at formulating arguments at a system level for hardware [4], for telecommunications systems [13], for distributed systems [17], for embedded systems [22] and for safety-critical systems [14]. Model checkers have been shown to be able to generate counter-examples to arguments of the form: all possible sequences of actions from a given state will satisfy a given temporal property. Most dynamic properties of systems can be cast in this form. Model checkers have been shown to scale-up reasonably to systems of systems, especially in telecommunications. Model checking has also proved itself in industrial application, the reason being that engineers find that the benefits justify the investment. That is, although not trivial to drive, model checkers do provide important early feedback on dynamic properties of systems, and in particular on proposed changes to systems. What has not yet been shown, however, is how such techniques can be extended to embrace partially-consistent and inconsistent data.

There are examples of reasoning about strong consistency in distributed systems, especially for cache-coherence [18] and for asynchronous communication [15]. Also, means of reasoning about system-level properties have been pioneered in the areas of security and authentication [16]. Loosely-coupled architectures have been formulated by Cardelli [3]. We have presented ways of formulating descriptions of large systems, viewed as components interfacing through services [11] and developed laws for reasoning informally about loosely-coupled systems [10]. Recently we have shown how these techniques anticipate new, loosely coupled architectures for inter-enterprise solutions [9].

The view of systems used in [10] used the following definitions.

- A component supplies services (eg methods, message handlers, event handlers) which may be used by other components.
- A system is a collection of components which cooperate by each using the services supplied by others.
- A dynamic system is a system which can be reconfigured (new components added, old ones removed) without having to stop.

Based on these definitions we stated laws for dynamic (loosely-coupled) systems, as follows.

- A component, added to a system, may not disrupt the behaviour of that system.
- A component, using the services of another, does so at its own risk and must protect itself from damage.
- A component offering a service does so at its own risk and must protect itself from misuse.

Both ICL and DERA need to plan major system evolutions, ICL on behalf of its customers and DERA on behalf of MoD. Both find themselves in an increasingly heterogeneous world, where they are experiencing first-hand the consequences of the everything-connected-to-everything information utility, in particular only partially-consistent information. In this project we are working together to develop the means whereby reasoning about proposed solutions can reduce the cost of system evolution by mitigating the consequences of this data pollution.

An Example:

In the initial stages of the project, we are building and analysing a collection of models to explore the nature of the problems which might be encountered by large integrated systems. Some of these models are to be based upon case studies provided by our collaborators in the project. One of these models (which we will call the "distributed agreement problem") is described below.

A system is comprised of a collection of entities such as shown in Figure 1. All nodes have some ability to communicate with others. Some (shaded in the diagram) also have some capability to sense data directly. Each of the nodes in the system will have a need to form some sort of view of the state of the "real" world. Nodes which sense some parameter of the "real world" may be concerned only with the value it is able to measure and simply pass on data about that value without any form of processing. Others will receive data from various sources (including its own sensors and derived data from other nodes) and attempt to reconcile this data into a single consistent view. Using executable models, we are investigating how data introduced into such a system affects the "view" of the nodes in the system. In a first system, the nodes may take one of a few states depending on an evaluation of the data they have. When running each node would be to either re-evaluate its data and decide whether to change state (or not), send some piece of data to another node in the model or receive a piece of data from another node. The purpose is to see how the system behaves when the system receives inconsistent data and to understand what features are needed to ensure that the system remains responsive to change without becoming unstable.

A concrete example of this type of behaviour might be how a large organisation which records address information about people. Some locations in its system (nodes in the model) will hold address information from people who are customers, others will record addresses of suppliers, staff, potential customers, government bodies, etc.. This information will be collected from a variety of sources (of differing reliability) including customer orders, customer enquiries, public records, etc.. as well as recorded and used for a variety of purposes like sending catalogues, billing customers, dispatching goods, paying suppliers, etc.. As the system operates, there will be interactions between the organisation in which information about addresses may become available and there will also be interactions between parts of the organisation where address information could be passed around within the organisation. The consequence of this will be that conflicts will arise which may need to be resolved. In some circumstances resolution may be easy, such as the new data filtering around the organisation after a member of staff who has moved house. Others are not so simple. For example; when it is discovered that a customer has different addresses recorded by our sales team and our accounts office, is this because they have moved or does this customer have an accounts office at a location separate from other parts of its operations?

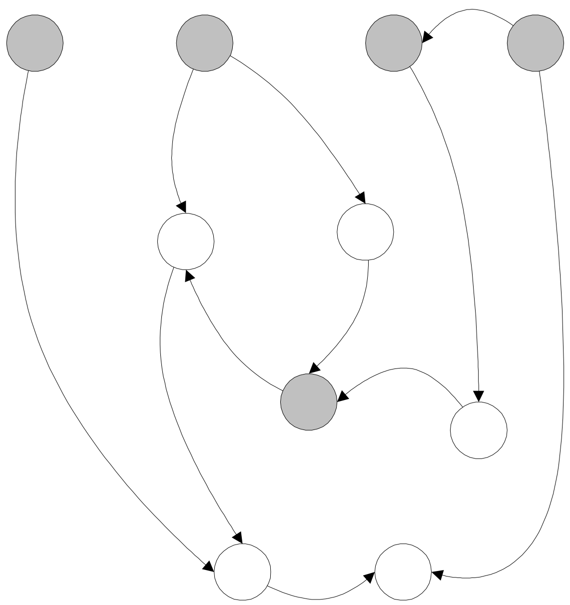

Figure 1

Work in Progress

One of the observations we have made from our initial modelling exercises is that although maintenance (or management) of consistency in data is undoubtedly important and it is easy to see how inconsistency causes problems in real systems, it is hard to reproduce these problems in modest models. One of two things tends to happen: either the model appears contrived or a simple, "obvious" solution to the problem appears. Where solutions appear, typically they are intimately related to the particular example in the model rather than the more general problem which the model had been built to investigate.

Following this experience, we have started to build some example systems which are much larger than our usual models. These systems will be limited implementations of the type of system used by our industrial partners. The first system is an internet banking system. Our banking system has a number of banks which are able to provide basic services to users, together with applications which simulate the activity of the clearing systems of real banking and interfaces which enable users to manipulate the system.

Communications between the various applications which form the "Banking" part of our system are handled by Microsoft Message Queue (MSMQ) which provides reliable asynchronous delivery of messages. This is the type of environment in which we expect the real implementations of the type of applications we wish to study to operate.

The objective of building these systems is to give us working environments with which we can experiment. These need to be both large enough for the data inconsistency issues to emerge and small enough for us to manage and reason about. In the case of our banking example, we could have constructed a working system more quickly if we had used commercial components. However, we wish to use these systems as test beds to examine the consequences of a variety of problematic behaviour by applications within the system (and possibly the infrastructure within which it operates) and to understand how applications might be built which are able to continue in the face of errors, omissions and inconsistencies. We felt we could not use commercial accounting or banking components because their developers will have incorporated features specifically to control or prevent the types of event which we wish to examine. For example, we have constructed a version of our application which represents the clearing system of a real banking system which sometimes fails to deliver entries to the correct destination. It is difficult to imagine how we could have generated such behaviour in a system built from commercial components.

Conclusion

RICES is a project aimed at understanding the problems encountered in the integration of large enterprise systems. In particular the project hopes to form an understanding of the nature of the errors and inconsistencies which occur in the data of these systems and to find some strategies which will enable the acceptable operation of these systems despite shortcomings in the information they have to use.

References:

[1] E. Bensley, “Evolvable Real-time C3 Systems,” *1st International Conference on Engineering of Complex Systems*, 1995.

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

[3] L. Cardelli, “Abstractons for Mobile Computation,” Microsoft Research Technical Report MSR-TR-98-34, 1998.

[4] E.M. Clarke, O. Grumberg, and D.E. Long, “Model Checking and Abstraction,” *ACM Transactions on Programming Languages and Systems*, pp. 1512-1542, 1994.

[5] M. Cusumano and D. Yoffe, *Competing on Internet Time - Lessons from Netscape and its battle with Microsoft*: The Free Press (Simon and Schuster), 1998.

[6] A. Dickman, *Designing Applications with MSMQ - Message Queuing for Developers*: Addison Wesley, 1998.

[7] E. Gamma and et al, *Design Patterns - Elements of Object Oriented Software Architecture*: Addison Wesley, 1995.

[8] M. Heimdahl and N. Leveson, “Completeness and Consistency in heirarchical state-based requirements,” *IEEE Transactions on Software Engineering*, 1996.

[9] P. Henderson, “Evolution of Dynamic Systems - architectures for reuse in the context of constant change,” *Submitted to International Conference on Software Re-Use (ICSR2000)*, 2000.

[10] P. Henderson, “Laws for Dynamic Systems,” *International Conference on Software Re-Use (ICSR 98)*, Victoria, Canada, 1998, pp.

[11] P. Henderson and G.D. Pratten, “POSD - A Notation of Presenting Complex Systems of Processes,” *First IEEE International Conference on Engineering of Complex Systems*, 1995.

[12] C.A.R. Hoare, “The role of formal techniques: past, current and future or how did software get so reliable without proof?,” *18th International Conference on Software Engineering (ICSE-18)*, Berlin, 1996, pp. 233-234.

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

[14] D. Jackson and J. Chapin, “Simplifying Air Traffic Control: An Excercise in Software Design,” 1999.

[15] R. Kurki-Suoni, “Component and Interface Refinementin Closed-Systems Specifications,” *FM '99*, 1999.

[16] B. Lampson, “Authentication in Distributed Systems,” in *Distributed Systems*, S. J. Mullender, Ed., 1993.

[17] J. Magee and J. Kramer, *Concurrency: State models and Java Programs*: John Wiley and Sons, 1999.

[18] S.D. Mulender, *Distributed Systems*: Addison Wesley, 1993.

[19] L. Nicolle, “John Taylor - The Bulletin Interview,” *The Computer Bulletin*, 1999.

[20] Object Management Group, “Common Object Request Broker: Architecture Specification,”.

[21] M.A. Ould, “Designing a re-engineering proof process architecture,” *Business Process management Journal*, vol. 3, 1997.

[22] S. Team, “SACRES - Safety Critical Realitime Embedded Systems,” , Final Priject Report.