BUILDING APPLICATIONS ABLE TO COPE WITH PROBLEMATIC DATA USING A DATAWARP APPROACH

Stephen Crouch, Peter Henderson, Robert John Walters
Electronics and Computer Science, University of Southampton, University Road, Southampton, UK

Keywords: DataWarp, Data Inconsistency, Enterprise information systems, Distributed systems, Dynamic systems

Abstract: As the amount of data systems have to work with increases, it becomes practically impossible to ensure it is consistent, no matter how tough we make our data collection procedures. Currently systems logic is based on the implicit assumption that the data they use is essentially correct and they struggle when this is not the case. To deal with this situation, we need to build applications which are better able to handle inconsistencies. In a series of experiments, we have shown that an application using our "DataWarp" approach to data enjoys a real advantage in one specific environment. Here we describe applying the approach more widely.

1 INTRODUCTION

In their origins, enterprise systems operated in isolation permitting them to impose rules and make assumptions about their data. Systems have become larger, more complex and more interconnected with the same technologies now being used on intranets as for the global internet (Christensen, Curbera et al. 2000; Hunter, Cagle et al. 2000; Snell, Tidwell et al. 2002). Data is no longer held in a single consistent database. Instead it is distributed around a network of cooperating applications in which data is partially replicated at many locations (Nicolle 1999). This pattern is a natural consequence of the way that systems now operate. Some measure of replication of data is deliberate and desirable as it helps to improve performance and resilience (Kemme and Alonso 1998; Wiesmann, Pedone et al. 2000). Today, the data used by enterprise applications is scattered throughout a network of databases and organisations. There is considerable variability in the amount and type of access applications have to data and there is no overall control of its content or accuracy. The situation is further complicated by each entity having its own interpretation of what it means for the data it uses to be consistent.

The immediate reaction in most organisations to problematic data is attempts to drive out the inaccuracies and inconsistencies. Schemes like distributed transactions (Gray 1978; Gray 1981) can guarantee consistency but with the large quantities of data present in enterprise systems today, using them is an enormous task which would be too restrictive to apply universally. The problem is further complicated by rate at which data changes with new and updated information is being added all the time. Even without the popular move to asynchronous communications (Microsoft 2001), new information takes time to propagate. Consequently the accumulated mass of data is unlikely ever to be in a truly consistent state.

We propose an alternative approach which is to accept the impossibility of eradicating all errors and inconsistencies and instead build applications which are able to survive encountering problematic data. One strategy is "DataWarp". We have already shown in a series of experiments that ships in a simulated sea battle using DataWarp enjoy a marked advantage over others (Henderson, Walters et al. 2003). In this paper we extend the approach.

2 MAKING PROGRESS: DATAWARP

There is an implicit assumption applications that the data available provides a window into some complete and consistent world. This model of the

Crouch S., Henderson P. and John Walters R. (2005).
BUILDING APPLICATIONS ABLE TO COPE WITH PROBLEMATIC DATA USING A DATAWARP APPROACH.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 411-414
DOI: 10.5220/0002555104110414

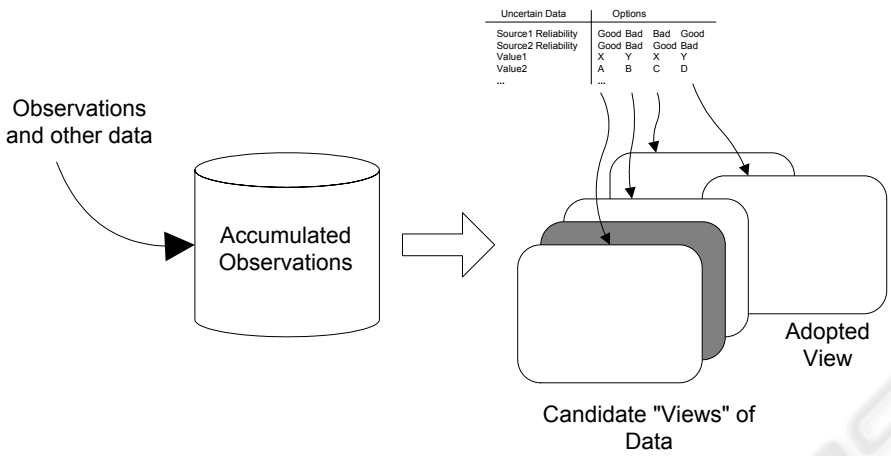

Figure 1: DataWarp in principal

world breaks down in distributed systems. One approach, is to refuse to accept inconsistency and fix up data as necessary. This is a short-sighted approach which is bound to lead to trouble.

As an alternative, we have developed DataWarp which is inspired by the attitude towards time of the applications in a virtual time environment (Jefferson 1985; Jefferson 1990; Cleary, Littin et al. 1997) where a global value for the current time is sacrificed in the pursuit of performance. It can lead to complications when applications communicate but the system as a whole produces the same results as a standard implementation.

By applying the same outlook towards data in general as a virtual time does to time, we arrived at an initial design for a DataWarp application, (figure 1). The application keeps all data and uses it to construct candidate views of the world. It then selects a plausible view to use.

When the application finds it must change view, as with a virtual time application which realises it has advanced its clock too far and has to rollback, the DataWarp application has to consider the implications of the view change on its actions and make corrections as necessary.

We have performed a series of experiments using a battleship simulation. We established that DataWarp ships enjoyed a clear advantage by acting immediately compared with the standard ships which couldn't decide what to do (Henderson, Walters et al. 2003).

DataWarp raises a number of issues:

2.1 Choosing a view

Naively, it might be assumed that an application would select the view which it expects to most accurately match the truth but applications may well use other criteria when making a choice. For example:

- Least costly – the view which costs the least to adopt.
- Least dangerous – the view which involves the least risk to the application or its users.
- Most profitable – the view which has the greatest potential for benefit.
- Most probably correct – the naïve option.
- Most easily defended.

Which of these is appropriate will depend on the application.

2.2 Changing view

When an application decides to change view, at least some of its assumptions will change. On making its change, the application needs to consider the implications for recent actions. Some will be unaffected, or still be considered acceptable. Others will now look wrong and will require corrective action. It may be possible to revoke some such "regretted actions", others will require corrective action calculated to compensate for the effect of the previous action. Having changed view, the application resumes work.

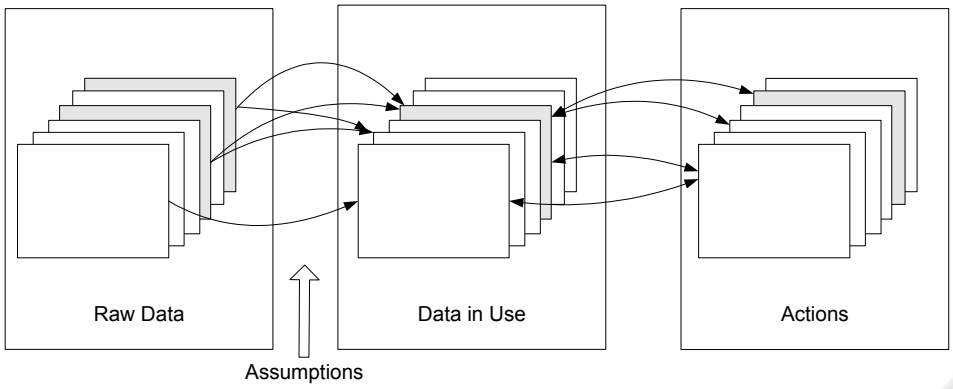

Figure 2: A more sophisticated implementation of DataWarp

3 DEVELOPING DATAWARP

In the first DataWarp applications, such as the battleship simulation, the identification of a view was a selection of exactly one from many candidates which was made at the application level, permitting DataWarp to be added to an existing application by adding a wrapper. Such a wrapper would manage the application's interactions with the outside world, generating and serving up to the application a view of the world which it finds acceptable. The wrapper would deal with all of the issues of holding multiple data values, selecting a view, generating input to the application as is necessary to adjust its internal data and compensating as appropriate after a view change (Henderson, Walters et al. 2001). However, we have concluded that this is not practical. Indeed, we have realised that the behaviour of the DataWarp ship which we deployed in the battleship simulation is more complex than we realised at the time.

In the simulation, a standard ship finding a new contact seeks to identify it before deciding what to do (attack or not). We implemented the DataWarp ships to assume new contacts are hostile. Therefore they attack new contacts regardless. This apparent recklessness is tempered by the way DataWarp ships destroy missiles in flight should realise they are targeted on an ally. However, the behaviour actually depends on the status of the contact in the ships data arises. The difference is that, if the contact is unknown (and assumed to be hostile) the ship not only launches an attack, but it also makes an immediate attempt to communicate with a view to identifying the contact, whereas if the contact is known to be hostile no attempt of communication is made as it could lead to unnecessary disclosure of the attacker's location.

In building DataWarp enabled applications we have found there are two areas where difficulties arise in particular. The first is that compensating actions are highly application and action dependent. The second is that, when faced with the choice of a view, one of the factors real applications want to take into account is the actions which might be influenced by the selection: in situations of uncertain data, applications adopt different positions according to what they are doing. For example, a bank with several possible addresses for a customer sending a formal demand for payment, would probably send a copy to every address to be as sure as possible that the customer received it but the same bank would act differently when issuing an ATM card.

We realise now that our original DataWarp vision of an application evaluating all reasonable views and selecting one with which to operate is too simple. A DataWarp application needs a more mature and complex attitude to the selection of view which admits the possibility of having more than one current view.

In a real application, DataWarp needs to be applied nearer to the level of individual actions. Some inconsistencies may be resolved before they become an issue. For the remainder, the application will select which data value(s) to use according to the action under consideration.

We now propose a revised structure for a DataWarp application. The application keeps a record of actions as before but in place of the single collection of candidate views from which it selects one to use, the application keeps a second database in which it records data values as it uses them. Links are maintained from this database to values which were used in connection with each action, whether the data was used as input to the action itself or in the decision process which lead to the action being taken. This "used" database in turn holds links into the raw data which were considered when arriving at the used value (figure 2).

Incoming data which creates ambiguity in the data-store causes the application to identify "used" values which are derived from data value concerned

and reconsider whether they need to be changed. Should any of the "used" values need to be changed, those actions which depend on them will need to be reconsidered. Actions which are now considered to be inappropriate must be revoked or compensated and replaced. In this way the application reproduces the DataWarp behaviour whereby it is able to work notwithstanding inconsistent data in its environment, by making assumptions and acting upon them whilst being prepared to retrace and replace its steps if necessary.

4 CONCLUSION

With the continued fall in the cost of computer hardware, computer systems continue to expand and hold more data. There is also a continuing trend for systems to be connected forming even larger systems which already hold so much data that they struggle to keep it consistent. The situation is unlikely to improve. The traditional reaction to managing inconsistency problems has been to adopt strategies which prevent them. However, the task of maintaining consistency is now overwhelming and it is inevitable that applications will encounter data problems. Therefore we need to build applications which can succeed in the presence of data which contains shortcomings.

Applications cannot afford to suspend or abandon every action where it encounters uncertainty or inconsistency in data since this limits their ability to make progress. Instead they need to be tolerant of data inconsistencies.

This paper proposes DataWarp which permits an application to progress notwithstanding problems in its data. The essence of the approach is that, the application "makes the best" of the data available. Provided the algorithms and heuristics used by the application are sound and reasonable, most of the actions the application takes will become definitive. Where the chosen value turns out to be wrong, the application has to put things right.

REFERENCES

Christensen, E., Curbera, F., et al. (2000). Web Services Description Language (WSDL).

Cleary, J. G., Littin, R. J., et al. (1997). Applying Time Warp to CPU Design. In *Proceedings of 4th International Conference on High Performance Comuputing (HiPC '97)*, Bangalore, India, IEEE.pp.290-295

Gray, J. N. (1978). Notes on Database Operating Systems. *Operating Systems: An Advanced Course*. R. Bayer, R. Graham and G. Segmuller, Springer. 60 pp. 391-481.

Gray, J. N. (1981). The Transaction Concept: Virtues and Limitations. In *Proceedings of 7th International Conference on Very Large Data Bases*, Cannes, France.pp.144-154

Henderson, P., Walters, R. J., et al. (2001). Inconsistency Tolerance across Enterprise Solutions. In *Proceedings of 8th IEEE Workshop in Future Trends of Distributed Computer Systems (FTDCS01)*, Bologna, Italy.pp.164-169

Henderson, P., Walters, R. J., et al. (2003). DataWarp: Building Applications which make Progress in and Inconsistent World. In *Proceedings of 4th IFIP WG 6.1 International Conference, Distributed Applications and Interoperable Systems (DAIS 2003)*, Paris, Springer.pp.167-178

Hunter, D., Cagle, C., et al. (2000). *Beginning XML*, Wrox Press Inc.

Jefferson, D. R. (1985). "Virtual Time." *ACM Transactions on Programming Languages and Systems* 7(3): pp.404-425.

Jefferson, D. R. (1990). Virtual Time II: Storage Management in Distributed Simulation. In *Proceedings of 9th Annual ACM Symposium on Principles of Distributed Computing*, Quebec City, Quebec, Canada, ACM.pp.78-89

Kemme, B. and Alonso, G. (1998). A Suite of Database Replication Protocols based on Group Communication Primitives. In *Proceedings of 18th International Conference on Distributed Systems (ICDCS)*, Amsterdam, The Netherlands.pp.156-163

Microsoft (2001). Microsoft Message Queuing Services, Microsoft.

Nicolle, L. (1999). John Taylor - The Bulletin Interview. The Computer Bulletin, British Computer Society.

Snell, J., Tidwell, D., et al. (2002). *Programming Web Services with SOAP*, O'Reilly & Associates Inc.

Wiesmann, M., Pedone, F., et al. (2000). Database Replication Techniques: a three parameter classification. In *Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000)*, Nurenberg, Germany, IEEE Computer Society Press