DataWarp: Building Applications which Make Progress in an Inconsistent World

Peter Henderson, Robert John Walters, Stephen Crouch, and Qinglai Ni

Declarative Systems and Software Engineering Group,
Department of Electronics and Computer Science,
University of Southampton,
Southampton, UK. SO17 1BJ
{ph, rjw1, stc, qn}@ecs.soton.ac.uk

Abstract. The usual approach to dealing with imperfections in data is to attempt to eliminate them. However, the nature of modern systems means this is often futile. This paper describes an approach which permits applications to operate notwithstanding inconsistent data. Instead of attempting to extract a single, correct view of the world from its data, a DataWarp application constructs a collection of interpretations. It adopts one of these and continues work. Since it acts on assumptions, the DataWarp application considers its recent work to be provisional, expecting eventually most of these actions will become definitive. Should the application decide to adopt an alternative data view, it may then need to void provisional actions before resuming work. We describe the DataWarp architecture, discuss its implementation and describe an experiment in which a DataWarp application in an environment containing inconsistent data achieves better results than its conventional counterpart.

1 Introduction

With the continued fall in the cost of computer hardware and the adoption of the same technologies for distributed computing on intranets within organisations and the global internet [1-4], we are seeing the creation of an everything connected to everything else information utility [5].

In their origins, even the largest of computer systems were comprised of a single process. This process worked with a collection of data which could be assumed to faithfully reflect the state of the real world. In turn, this permits applications to expect rules which apply to the real world to be reflected in the data. As systems have evolved they have developed into networks of communicating components which are assembled into complete applications. Increasingly these networks are adopting asynchronous architectures using a variety of technologies [6-11]. These components may understand that the data they have is incomplete, but they expect the subset that they have to be faithful to some notion of reality. In other words, they suppose that the data they hold provides them with a window into some complete and accurate data store for the whole system which is free from inconsistency as shown in Fig. 1. Here, A and B each have a partial view of the overall state of the system. By exchanging

data, they are able to make their views of the data compatible but their assumption about the data they hold goes further. They both suppose that there exists some definitive value for the true state of the system and that the data they hold is consistent with it. These ideals are both difficult to establish and maintain.

Distribution and replication of data can impart additional resilience and performance to applications [12, 13]. However, as the accumulated mass of data grows it is becoming ever more difficult to maintain the illusion of universal consistency which underpins much of the reasoning applied by applications [14, 15]. Schemes, such as distributed transactions [16, 17] can guarantee this consistency. Their operation requires co-ordination and co-operation between the various locations at which the data is held. The effort associated with this co-ordination is appropriate for some processes, such as a Bank transfer where it is essential that neither end of the transaction can occur without the other. However they are too restrictive to be applied universally [18, 19], especially in enterprise and inter-enterprise systems.

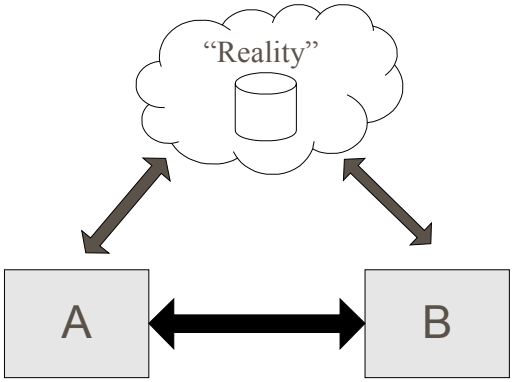

Fig. 1. The view of data of a traditional application

Instead of forcing the data to fit the understanding of our applications, we need to find ways to implement applications in such a way that they can operate in the imperfect data environment in which they necessarily find themselves. If we are to do this, we need to relax the reliance on the assumption of global consistency in data.

This paper introduces DataWarp. A DataWarp application:

- Maintains many *views* of data
- Selects a view based on *assumptions*,
- Is prepared to *alter its assumptions* and take remedial action

2 An Example: MQ Defence

Whilst there are many real enterprise systems which are faced with inconsistent data, these are not ideal subjects for study for two reasons. Firstly, they are so large that describing how they operate is difficult. Secondly, their users and developers have

worked hard to prevent, or ameliorate the effects of, what we perceive as interesting behaviour: being tolerant of *inconsistency*. Recognising this, we have constructed an experimental system which uses the asynchronous technologies of enterprise systems (specifically Message Queue) but is not crafted a priori to work around data inconsistency problems.

The system is a defence simulation in which ships move around a two dimensional grid. It is implemented as a collection of applications which communicate using a commercial message passing middleware product (MSMQ, [7, 20]). Each ship has its own message queue which it is required to read promptly.

Any ship in the system is able to perform a number of actions:

- Move. Ships are artificially constrained by the edges of the grid but otherwise are free to move in any direction at any time.
- Sense local information. Each ship has a sensor which it is able to interrogate. A ship's sensor responds with a message containing a list of ships within its range. Information supplied by sensors is similar to that which a real ship at sea might collect using radar.
- Communicate. A ship may establish a communication channel between itself and any other ship it knows, enabling it to interrogate other ships in a similar way to its sensor. The usual response from a ship to such an enquiry is a message detailing its present view of the entire grid. However, unlike sensors, ships may not respond to all enquiries (see below). Channels operate in both directions, so the ship at either end may use a channel. In establishing a channel, a ship gives away its existence, location and identity.

The range of the sensors is limited and the ships move, so no ship can establish and maintain a complete, up to date view of the grid using its sensor alone since at any time there will be areas which are beyond the range of its sensor. However, by communicating ships can assist each other by sharing information, but at the expense of being prepared to resolve inconsistencies. Consider three stationery ships which have established communications when a fourth moves out of the sensor range of one ship into the sensor range of another. In communication, the third ship will receive reports from the two who have seen the moving ship, placing it in two locations (its true position when last observed) leaving the third ship with a decision to make.

This environment is sufficiently simple to enable us to identify and reason about what is happening whilst also sufficiently realistic to present many of the problems of inter-enterprise distributed computing (not least delays and unreliability) [21].

For this experiment, the ships has have a notion of allegiance and are able to attack one another using missiles which usually, but not always, destroy their target. A ship which fires a missile directs it to a location on the grid which it takes time to reach. Thus a ship may survive an attack if it is lucky or moves far enough whilst the missile is in transit. Each ship is allocated to one of two sides and each navy assists their side's effort to dominate the grid by supplying information to their allies and attacking their enemies. Ships don't respond to enquiries from their enemies so a ship which responds to an enquiry is a friend. A ship which doesn't respond is likely to be an enemy, although it could be a friend who has failed to respond, or whose response is delayed or lost in transit. The information supplied by sensors does not include the allegiance of contacts. Fig. 2 shows an example ship's view of the world. It is aware

of four ships, itself (One), a friend (Two), an enemy out of range (Three) and a ship of unknown allegiance (Four).

In this scenario, there is a further element to the data inconsistency problem faced by the ships. In addition to needing a strategy to resolve conflicting reports about the location of other ships, they now have the urgent need to decide what to do about unidentified ships which are within reach of their weapons (and so probably close enough to launch an attack if they are an enemy), such as a contact which has been observed for the first time by the local sensor or one for which conflicting data has been received.

For the experiment, we pitched two types of ship against each other; ships following the DataWarp philosophy and standards ship using a more traditional behaviour.

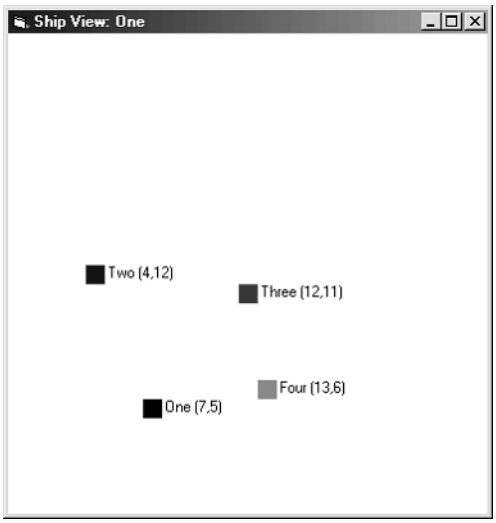

Fig. 2. A view of the world as seen by an MQDefence ship

We use rules as a universal means of describing behaviour at a suitable level of abstraction. Fig. 3 describes a ship *not* using DataWarp. The rules are divided into two sections. The first describes the way the ship reacts to stimuli (the reactive rules), the second describes autonomous behaviour which the ship initiates itself (the proactive rules). This behaviour reflects the usual attitude of applications. For this example, we will concentrate on the issue of the unidentified contact. Faced with a situation such as that shown in Fig. 2, should the unidentified ship Four be within range, the standard ship doesn't know what to do. Before it can proceed it must identify the ship. It cannot do otherwise for fear of attacking a friendly ship. The ship could try asking its ally (named Two in Fig. 2), but it may not know either. The only option certain to give a result is to open a channel. However this is risky: in trying to communicate with an unidentified contact, a ship gives away its own position and allegiance. If the contact is an enemy, it might launch an immediate attack.

```
/* Reactive rules */
Attack any enemy ship within range.
Open a channel to any unidentified ship within range.
A ship which replies on a channel is an ally.
Store data received from sensor.
Store data received along channels.
Respond to enquiries from allies with details of our view.
Reconstruct our view of the grid when the data store changes.

/* Proactive rules */
Move according to algorithm.
Send enquiries on channels.
Read sensor.
```

Fig. 3. Behaviour of a Standard Ship

Fig. 4 outlines the alternative behaviour of a DataWarp ship. This differs from the standard ship in that it resolves problems in its data by constructing a collection of views of the grid and picking one to act upon. In this particular example, the problem is the allegiance of ship Four which must be either an ally or an enemy. This leads to the creation of a set of possible views which can be divided into two subsets: those in which Four is an ally and those in which Four is an enemy. The ship makes its choice by reference to the rules by which it operates. If it elects to assume Four is an enemy, it picks a view from the set in which Four is hostile. The particular view chosen will depend on how this DataWarp ship resolves the positional issues in its data. Should Four come within range, it will be attacked (as an enemy), followed by an attempt at communication. Should Four identify itself as a friend, the DataWarp ship would then know that it chose the wrong view of the gird and abandon it in favour of an alternative in which Four is an ally. In this particular circumstance, the allegiance of Four is now certain so those views in which Four is assumed to be hostile may be discarded. However, in general, views which are presently discounted need to be retained. A side effect of changing its view is that the DataWarp ship will realise that it has launched a missile at an ally and destroy it before it can do any harm. A criticism might be that, should a friendly contact fail to respond in time our ship would destroy an ally. This is indeed a risk. However, the standard ship is worse – if one of its contacts fails to respond, it launches an irretrievable attack.

Should Four be an ally, the DataWarp ship will also realise that it may have misinformed other allies with which it has been in communication whilst the assumption was in force. It will attempt to correct any consequences of this by sending corrections. These corrections have the potential to ripple around if they cause ships that receive them to change views too.

```
/* Reactive rules */
Attack any ship within range known or assumed to be an enemy.
Where a ship is attacked on the assumption of hostility, open a channel
 to that ship.
Destroy a missile in flight to an ally.
A ship which replies on a channel is an ally.
Store data received from sensor.
Store data received along channels.
Respond to enquiries from allies with a details of our view.
Reconstruct candidate views of the grid when the data store changes.
If data affecting assumptions changes, reselect view of grid.
```

```
Send corrections when a change of view changes allegiances of contacts
 advised to allies.

/* Proactive rules */
Move according to algorithm.
Send enquiries on channels.
Read sensor.
Reconsider assumptions.

/* Assumptions */
An unidentified ship is an enemy.
/* Other assumptions and guidelines about resolving positional issues */
```

Fig. 4. Behaviour of a DataWarp Ship

In addition to the experiments performed in the MQDefence environment which is actually distributed and uses MSMQ for messages, we have performed further experiments using a less elaborate simulation written in Java.

To run the experiments, we have needed to add some detail to the definitions given above. This has concerned matters such as the size of the grid, the ranges of sensors and missiles, the time a ship waits for a response to a communication before concluding another is an enemy and the time a missile takes to reach its target. The experiments have been run using a 300x300 sector grid. The number of ships in each navy, as well as the range of sensors and missiles, is a balance. More ships on the grid, and longer ranges for the weapons and sensors mean ships find and destroy each other more easily and lead to shorter experiments.

The pro-active rules are implemented using a simple timer which, on expiry, reads the ship's sensor, reads each of its channels and moves the ship a single division of the grid in a random direction which is weighted in favour of continuing in the same direction as the previous move. All ships use the same moving algorithm and have the same sensor and weapon ranges. For most experiments, the range of the sensors has been set slightly in excess of the range of weapons.

Table 1. Sample results, Standard vs. Standard ships

	Experiment 1	Experiment 2
Ships destroyed: side 1	16	27
Ships destroyed: side 2	11	18
Total	27	45

Initial experiments were performed on a number of machines in the laboratory with six ships on each side, missile and sensor ranges of 45 and 50 respectively. These were permitted to run until one fleet is eliminated, and show that DataWarp ships win approaching 80% of the battles, though they do use more missiles. Even after adjusting the parameters to favour the standard ships, the DataWarp side still wins convincingly, though in some configurations the number of DataWarp ships attacking their allies is significant. Strictly limiting the number of missiles of each ship means that the DataWarp ships are likely to run out – making them vulnerable to attack and

unable to contribute further to the success of their side. However, they still enjoy a significant advantage.

There is a considerable element of chance to where and when ships encounter each other which can affect the outcome. This is clearly a consequence of the moving algorithm, but we retained this algorithm since we wish to avoid the possibility that either type of ship should enjoy an advantage as an accident of the way the ships movement algorithm interacts with the features of the system.

Table 2. Sample results, Standard vs. DataWarp ships

	Experiment 1		Experiment 2	
Standard ships destroyed		130		123
DataWarp ships destroyed by standard ships	24	52	8	43
DataWarp ships destroyed by DataWarp ships	28		35	
Total		182		166

Following on from the initial experiments, we have conducted several extended experiments in which the number of ships on the grid has been maintained by replacing each one which is destroyed with another of the same allegiance at a random location at the edge of the grid. In these experiments there can be no question of either side achieving dominance of the grid by destroying all of their opponents since they are replaced as they are destroyed. However the objective of each of the participants remains unchanged - to assist its side towards that goal by destroying their enemies. Hence the relative number of ships lost by the two sides is a reasonable measure of their relative success. The examples shown are selected from those experiments where the DataWarp ships enjoyed the least advantage. Ironically, even in those experiments the standard ships still didn't manage to sink opponents with much of the improvement in their apparent performance accounted for by DataWarp ships sinking each other. Nevertheless, typically twice as many of the standard ships are destroyed as the DataWarp ships. Table 1 and Table 2 give sample results from some extended runs of the experiment in MQDefence.

3 DataWarp

The traditional approach adopted by applications faced with unreliable data is to subject it to validation procedures to identify and eliminate problematic data. Once past this verification, applications essentially accept data as being absolute and act accordingly. When anomalies arise, the typical action is to report them as exceptions which generally demand external intervention, often from human operators.

DataWarp was initially inspired by TimeWarp [22, 23], which was developed for the implementation of distributed simulation. The processes in a TimeWarp environment do not have a synchronised notion of time. There is no central record of

time in the system at all. However, they still manage to operate in such a way that the results of their computations are unaffected. This ability for a system without a consistent notion of time is extended by DataWarp towards data in general.

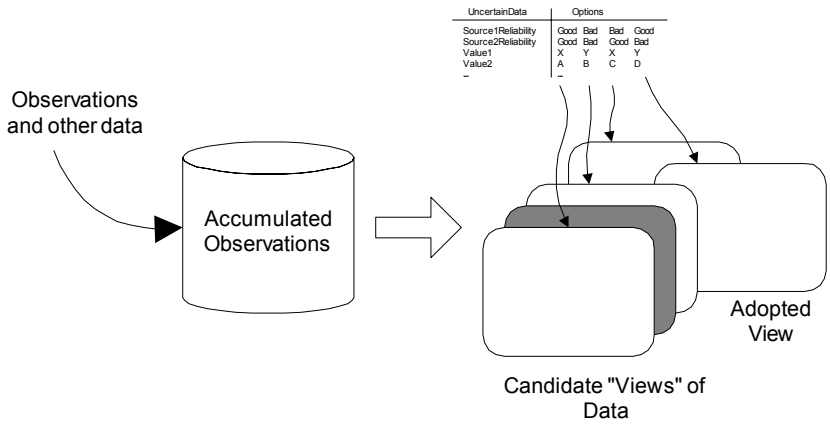

Fig. 5. DataWarp

However, our motivation is different. With TimeWarp, a consistent global notion of the current time is traded for improved performance within a controlled environment. Our applications have no choice about their environment: they operate in the context of the uncontrolled, asynchronous, ever changing and more connected world of enterprise systems. This environment is increasingly polluted with poor data over which no single application exerts control. Accepting that dirty data is inevitable, DataWarp applications adopt local behaviours towards data which are similar in spirit to those of a TimeWarp process towards time.

In place of the usual behaviour, in which an application develops a single view of the world, when faced with inconsistencies in data, a DataWarp application constructs a collection of alternative data views which it regards as candidates for the view of the world it will use. With the passage of time and the arrival of further data, the application maintains this view-set, adding more when additional inconsistency arises and removing views where new data permits them to be discounted.

DataWarp applications act promptly on input as it is received but retain a history of state, input and actions so that when errors come to light, they are able to re-consider and, where necessary undo actions or make compensations. Imposing an obligation onto applications to retain a history of actions may appear onerous, but in fact most commercial applications already record this information in audit trails.

There are real differences between a TimeWarp process and a DataWarp application which concern:

- The environment in which they operate
- The nature of the data which could be subject to amendment
- Identifying the need to rollback
- The actions which may be required to achieve a rollback

TimeWarp is concerned with time - a single valued data item which (in some general sense) always progresses. The processes of a TimeWarp system may be thought of as being distributed along a time-line. The extent to which the processes are distributed along this line will depend on the pattern of communication between them. Since there is no co-ordination of time in the system, so long as the processes operate in isolation their clocks will tend to drift apart. The effect of communication depends on the relationship between the local times in the processes concerned. A side-effect of a message which causes the receiving process to rollback is the near synchronisation of the clocks of the sending and receiving processes. More communication between processes is likely to cause the times on their local clocks to bunch together. Additionally, despite temporary variations, in the long term the collection of processes as a whole progress along this line from the past into the future.

In contrast, DataWarp is concerned with many unordered data items. Its applications might be considered as being distributed about a multi-dimensional space instead of along a line. In common with TimeWarp, it is reasonable to expect that applications which communicate will tend to approach each other in this space, but there is no equivalent of the relentless progress of time from the past to the future so, faced with two incompatible pieces of data there is no simple universal way for a DataWarp application to choose between them. Where a TimeWarp application is able to identify situations where rollbacks are necessary by simple comparisons between the timestamp on a message and the value on its local clock, a DataWarp application has no such a simple test. Instead, it has to identify inconsistencies in the data it has received (and acted upon) using application specific consistency rules. However, where the traditional application resorts to raising an exception demanding some external intervention, the DataWarp application will be able to cope. In the example above the DataWarp ship, on discovering faulty data and having to change its view destroys any missile which it has fired at an ally and sends corrections to the information it has supplied to any of its allies.

The details of when and how DataWarp applications identify which particular data items are at fault, and how far to rollback has to be application specific. During execution, the DataWarp application accumulates a collection of data items about which it has made assumptions. These assumptions may be directed towards using values which are the most likely to be true but according to the circumstances, other strategies are also appropriate. For example, the application may elect to use values which are most easily defended (should that become necessary) or the least likely to cause damage in the event that they have to be changed. When the complexity of the uncertainty being managed warrants it, technologies like belief revision (or truth maintenance) [24, 25] have to be employed.

The fact that a DataWarp application works with many pieces of data does bring one particular advantage: when an error is discovered and a roll-back is required, the application does not need to revert fully to its state before any of the erroneous actions were taken. Instead the application need only address those actions which may have been affected by the data concerned. For example, on discovering an error in its record of a customer's address, there is no need for an online store to rollback all of its actions. It only needs to consider rolling back actions relating to that particular

customer. The remainder can be allowed to stand, making the rollback a less onerous task.

There is one further complication which the DataWarp application has to be able to accommodate. TimeWarp processes operate in a controlled environment. This environment is populated with TimeWarp-aware processes, enabling a process which needs to perform a rollback to retract actions and communications since the destination processes will be equipped to accept the retraction (and instigate their own rollback, if required). However, the DataWarp application does not have this luxury. It has to be able to handle the consequences of wishing to retract messages already sent where the receiver may not be prepared to accept message retractions or even understand them. The DataWarp application needing to perform a rollback handles this by examining the actions it needs to retract and dividing them into two categories according to whether any evidence of the action has yet been disclosed to the outside world. Those actions which have been disclosed are described as having hardened. Those actions which have not yet hardened can always be reversed because they affect only state internal to the application. Of the remainder, some aspect of the action has been seen by another application. Where it is known that the other application will accept message retractions, perhaps because the application advertises this facility to potential users, then this is the preferred option. Otherwise, some kind of compensating action will have to be performed. According to the nature of the action, and possibly how long ago the action was initially performed, the application may well have to accept that the effect of the compensating action may fall short of completely eliminating the effects of the original action.

In deciding how to act upon data as it arrives, there is a judgement for the application to make about how quickly to allow its actions to become externally visible and so liable to harden (when a third party sees them) since hardened actions are more difficult to retract. An application applying DataWarp to its processing may gain an advantage over its competitors because, since by processing work optimistically it is able to respond more quickly than its traditional peers. Alternatively, at least for some transactions, it may feel that it is appropriate to conceal the effects of some transactions for a short time in order to increase the probability that, should a rollback be necessary it will not have to deal with hardened actions. Consider the situation faced by an online bookshop receiving an order. In a traditional view of operation, the shop will process the order as a sequence of actions, starting with checking its stock, followed by processing the payment, sending the book and noting the sale to order replacement stock. The DataWarp shop can process this order differently. Instead of carrying out the actions in sequence, it can set all of them in motion as soon as the order arrives: it assumes the book is in stock and the clients payment will be honoured. If, for example, the book is not in stock then the order needs to be rolled back and its processing re-started. In this case, in place of sending the book the shop may send a communication to the client informing them of the situation and requesting confirmation that they still want to buy and cancel the request for payment (or make a refund). The automated re-order action on the sale may not now be appropriate either. It might be replaced as a customer specific order or cancelled completely pending further contact from the client.

4 Conclusion

As computer systems become larger and more widespread, they are collecting huge amounts of data. Many systems already have so much data that they struggle to keep it up to date and consistent. The continuing trend of connecting systems into even larger systems is making this problem more difficult and the situation is unlikely to improve. The situation is further complicated by the mobile systems which only maintain intermittent contact with our connected world and the asynchronous architectures which are being increasingly used. The traditional approach to managing problems arising from inconsistencies in data is to avoid the problem by enforcing consistency using strategies such as distributed transaction processing. However, the volume of data and the complexity of the interconnections between the systems which process is increasing whilst at the same time, the data environments are becoming less controlled and more varied. Together these mean that the task of maintaining consistency is becoming overwhelming. Contemporary systems need to be able to succeed despite having to work with data which they know contains errors and inconsistencies. They need to be *inconsistency tolerant*.

We have performed a collection of experiments both in an experimental environment built using a commercial message passing middleware product and in a simulation environment which shows that an application adopting a DataWarp approach enjoys a considerable advantage when faced with inconsistent data.

In DataWarp, applications proceed provisionally with their work but are prepared to revoke actions in the event that the data which motivated them turns out to be incorrect and re-commence operations with the new, (hopefully) better data. As time passes, these provisional actions become more nearly permanent. Eventually they can be regarded as definitive. In common with the attitude of TimeWarp processes towards time, the DataWarp applications do not concern themselves with maintaining a view of the world which is consistent with others using the same data unless or until they are forced to do so by interaction. When they do acquire additional data they decide whether if they need to adopt a different data view.

In summary, DataWarp is an architecture for building applications which are inconsistency tolerant. A DataWarp application:

- Maintains many *views* of data
- Selects a view based on *assumptions*,
- Is prepared to *alter its assumptions* and take remedial action

References

1. Universal Description Discovery and Integration (UDDI)), Technical White Paper, see http://www.uddi.org (2000)
2. Christensen, E., et al.: Web Services Description Language (WSDL), see http://msdn.microsoft (2000)
3. Hunter, D., et al.: Beginning XML, Wrox Press Inc (2000)
4. Snell, J., D. Tidwell, and P. Kulchenko: Programming Web Services with SOAP. First Edition, O'Reilly & Associates Inc. (2002)

5. Nicolle, L.: John Taylor - The Bulletin Interview, British Computer Society, The Computer Bulletin. (1999)
6. Microsoft: Legacy File Integration Using Microsoft® BizTalk Server 2000, see http://www.microsoft.com/biztalk/techinfo/LegacyFileIntegrationWP.doc, Microsoft (2000)
7. Microsoft: Microsoft Message Queuing Services, see http://www.microsoft.com/ http://www.microsoft.com/ntserver/appservice/techdetails/overview/msmqrevguide.asp, Microsoft (2001)
8. Object Management Group: Common Object Request Broker: Architecture Specification, see http://www.omg.com
9. Sun Microsystems: Enterprise Java Beans, see http://www.sun.com
10. Szyperski, C.: Component Software, Longman (1998)
11. Thomas, A.: Enterprise JavaBeans Technology, Patricia Seybold Group. (1998)
12. Kemme, B. and G. Alonso: A Suite of Database Replication Protocols based on Group Communication Primitives. Proceedings of 18th International Con-ference on Distributed Systems (ICDCS), Amsterdam, The Netherlands (1998)
13. Wiesmann, M., et al.: Database Replication Techniques: a three parameter classification. Proceedings of 19th IEEE Symposium on Reliable Distributed Systems (SRDS2000), Nurenberg, Germany, IEEE Computer Society Press (2000)
14. Sircar, S. and A. Kott: Enterprise Architecture Analysis Using an Architecture Description Language. Proceedings of DARPA Symposium on Advances in Enterprise Control, Minneapolis (2000)
15. Dayal, U., M. Hsu, and R. Ladin: Business Process Coordination: State of the Art, Trends, and Open Issues. The VLDB Journal. Vol. (2001) 3-13.
16. Gray, J.N.: The Transaction Concept: Virtues and Limitations. Proceedings of 7th International Conference on Very Large Data Bases, Cannes, France (1981)
17. Gray, J.N.: Notes on Database Operating Systems, in Operating Systems: An Advanced Course. R. Bayer, R. Graham, and G. Segmuller, Editors, Springer, (1978) 391-481
18. Henderson, P., R.J. Walters, and S. Crouch: Inconsistency Tolerance across Enterprise Solutions. Proceedings of 8th IEEE Workshop in Future Trends of Distributed Computer Systems (FTDCS01), Bologna, Italy (2001)
19. Henderson, P., R.J. Walters, and S. Crouch: RICES: Reasoning about Infor-mation Consistency across Enterprise Solutions, in Systems Engineering for Business Process Change: New Directions, Springer-Verlag London Limited, London, (2002) 367-371
20. IBM: MQSeries Family, see http://www-4.ibm.com/software/ts/mqseries/ (2001)
21. Henderson, P.: Reasoning about Asynchronous Behaviour in Distributed Sys-tems. Proceedings of The 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'02), Greenbelt, Maryland (2002)
22. Jefferson, D.R.: Virtual Time. ACM Transactions on Programming Lan-guages and Systems. Vol. 7(3). (1985) 404-425.
23. Jefferson, D.R.: Virtual Time II: Storage Management in Distributed Simula-tion. Proceedings of 9th Annual ACM Symposium on Principles of Distrib-uted Computing, Quebec City, Quebec, Canada, ACM (1990)
24. Friedman, N. and J.Y. Halpern: Belief Revision: A Critique. Journal of Logic, Language and Information. Vol. 8(4). (1999) 401-420.
25. Shaprio, S.C.: Belief Revision and Truth Maintenance Systems: An Overview and a Proposal, Department of Computer Science and Engineering and Center for Multisource Information Fusion and Center for Cognitive Science, State University of New York at Buffalo, Buffalo (1998)