A low-latency resilient protocol for e-business transactions

Muhammad Younas*

Distributed Systems and Modelling Research Group,
School of Mathematical and Information Sciences,
Coventry University, Coventry, CV1 5FB, UK
E-mail: m.younas@coventry.ac.uk
*Corresponding author

Barry Eaglestone

Department of Information Studies,
Computational Informatics Research Group,
University of Sheffield,
Sheffield, S10 2TN, UK
E-mail: b.eaglestone@sheffield.ac.uk

Kuo-Ming Chao

Distributed Systems and Modelling Research Group,
School of Mathematical and Information Sciences,
Coventry University,
Coventry, CV1 5FB, UK
E-mail: k.chao@coventry.ac.uk

Abstract: This paper proposes a novel protocol for e-business transaction management, called the low latency resilient (LLR) protocol. LLR applies new correctness criteria based upon enforcing semantic atomicity with increased resilience to failure, and has been implemented as a prototype system for e-business transaction processing. The analysis given shows that LLR avoids resource blocking of e-business systems and significantly reduces response time between such systems. In addition, LLR significantly improves the resiliency of e-business transactions in the event of failures or the unavailability of requested services. This is achieved through the use of e-business transaction within which alternative sub-transactions are specified.

Keywords: transaction management; e-business; SACReD transactions; resiliency; latency of transaction commit protocols; evaluation of commit protocols; performance; resource blocking; calculus of communicating systems (CCS); formal specification.

Reference to this paper should be made as follows: Younas, M., Eaglestone, B. and Chao, K-M. (2004) 'A low-latency resilient protocol for e-business transactions', *Int. J. Web Engineering and Technology*, Vol. 1, No. 3, pp.278–296.

Biographical notes: Dr. Muhammad Younas is a Senior Lecturer in Computer Science in the School of Mathematical and Information Sciences at Coventry University, UK. He has received PhD degree in Computer Science from the University of Sheffield, UK, in 2001. His research interests include web and database systems, transaction processing, agent technology, and mobile computing. He has published various papers in these areas.

Dr. Barry Eaglestone is a Senior Lecturer in Information Systems in the Department of Information Studies at the University of Sheffield, UK. He has been an active researcher in the area of databases since the early 1980s, and has published widely in the research literature on databases and advanced database applications. He has also authored textbooks on relational, object-oriented and web databases. Dr. Eaglestone is Director of the Centre for Health Information Management Research and leads the Sheffield University Data Management and Information Systems Research Group and also the Music Informatics Research Group.

Dr. Kuo-Ming Chao is a Senior Lecturer in Computer Science in the School of Mathematical and Information Sciences at Coventry University, UK. He leads the Distributed System and Modelling Research Group. His main research interests are intelligent agents, web and database technologies and their applications.

1 Introduction

This paper proposes a low-latency resilient (LLR) protocol, for effective management of e-business transactions. E-business facilitates the development, deployment and operation of new electronic business services over the internet. It is therefore *federated*, since it involves interaction across management domains and enterprise networks, and *autonomous*, since different participating parties generally belong to different organisations. Transaction management (TM) is therefore a key requirement, to protect and manage the integrity of transaction outcomes, since these are often financially critical. TM is also essential to consistently manage the critical information resources which underlie many e-business activities. In practical terms, effective e-business TM is necessary from the customers' perspective, to guarantee that services and products obtained from an e-business are consistent with what the customers believe they have requested and the system has delivered. From the perspective of those involved in providing the services and products, TM must ensure that e-business transactions are correctly executed, that the enterprise has correct information about the outcomes of those transactions, and that the information held in an enterprise's databases is maintained to provide a truthful and consistent record of the state of the enterprises. However, TM is problematic because the integrity of transactions and databases must be ensured in the presence of concurrent transactions, system and transaction failures and abandoned or aborted transactions, while at the same time providing acceptable resource utilisation and response time for customers.

LLR addresses those requirements. Specifically, our evaluation of LLR shows a reduction in latency (response time) between participating systems of e-business transactions, compared to current technologies based on the classical two-phase commit (2PC) protocols. Other improvements are avoidance of blocking while still maintaining

systems' autonomy, and enhanced resilience of e-business transactions to failures. The transaction correctness criteria that allow LLR to achieve these improvements are called SACReD (Semantic atomicity, Consistency, Resiliency, Durability) [1] – which better meet the requirements of e-business transactions.

Section 2 critically analyses current approaches to e-business transactions management (TM), thus presenting the motivation for LLR protocol. Sections 3 and 4, respectively, describe the LLR protocol, and evaluate its performance and resiliency, in comparison with existing protocols for e-business TM. Section 5 concludes the paper.

2 Related work

In this preliminary section we present the motivation for our new e-business TM mechanism, through a critical review of related work. Arguably, the requirement for effective TM has heightened importance for e-business, since the environment is more open, non-prescriptive, and vulnerable to failure due to the widespread distribution and unreliable nature of the internet. However, in spite of this, much e-business TM technology remains grounded in the technology developed for 'conventional' database systems [2,3]. In particular, 2PC and its variant Presumed Abort (PA) protocols, are widely used in the commit service of e-business transactions (e.g. [4–7]), even though their limitations are well-known [1,8–10]. The first part of this section therefore focuses on inadequacies, within the e-business context, of 2PC-based protocols and the ACID (Atomicity, Consistency, Isolation, Durability) transaction correctness criteria they enforce. The second part reviews less conventional approaches which address those limitations. Finally, we present the SACReD criteria which better suit the characteristics of e-business transactions.

2.1 Two-phase commit-based protocols

The common model for the 2PC protocols [11,12] executes over a configuration comprising a *coordinator process* at the site from which the transaction is submitted, and *participant processes* at sites that are accessed by the transaction. The coordinator enforces the integrity of a transaction by authorising participants to commit their respective sub-transactions, only after they have unanimously confirmed that they are able to do so. Otherwise, the transaction is aborted. Thus, 2PC enforces the classical ACID criteria for transaction correctness, maintaining a strict atomic and isolation policy. Consequently, transaction must follow the 'all-or-nothing' ethos and intermediate results of partially completed transactions cannot be exposed to other transactions. Variants of 2PC are also used for e-business TM so as to reduce protocol overheads. The most commonly used variant of 2PC is the presumed abort (PA) protocol [5–7]. PA protocol reduces the numbers of messages and forced-write operations by assuming that a sub-transaction is aborted, in the absence of confirmation that it has been committed.

Limitations of 2PC are well known [1,8–10,13]. In particular, a 2PC-based protocol becomes vulnerable to performance deterioration when participant processes enter into the prepared-to-commit state. In this state, a participant has confirmed to the coordinator that it is able to commit its sub-transaction. Sub-transactions are then suspended pending on the decision of the coordination process and are therefore vulnerable to arbitrary and long delays. This can occur, for example, if an internet failure isolates one of the

processes involved in the transaction. Also, in order to ensure atomicity, resources cannot be released by sub-transactions until the commitment or abortion of the whole transaction. Consequently, resources held by the transaction are blocked for use by others during these delays. In fact, even in a hypothetical failure free environment, useless consumption of system resources can increase as a consequence of delays caused by internet related problems, such as overloaded links and servers [14]. Performance problems are also a consequence of characteristics of e-business transactions themselves [10]. These are often long and involve multiple nested transactions ranging over multiple systems, with extensive user interaction. Consequently, 2PC-based e-business TM is prone to resource blocking, in particular caused by slow user response time during heavily interactive transactions.

The above performance problems may be further exacerbated where heterogeneous system domains belong to distinct and possibly competing business organisations. For example, a 2PC-based protocol called TIP (transaction internet protocol [5]) can be used to cause deliberate *blocking* and *jamming* of e-business transactions in order to gain competitive advantage [15]. A participant can block a competitor's system by simulating a failure which, if it had occurred, would prevent participating systems from communicating. It can also cause jamming by deliberately delaying responses, or by aborting sub-transactions. Kemptser et al. [15] also note the potential for a participant to strategically delay completion of a transaction to gain financial advantage. They cite, as an illustration, a system for buying bonds and gold futures, the prices of which generally move in opposite directions when the interest rate changes. A TIP-based transaction is initiated to buy these commodities from two different suppliers who, in principle, must respond together against the prepared message. However, a bond seller may delay a response, pending imminent news on interest rates, so as to profit from any market price falls that are lower than the agreed delivery price.

As a more general criticism of 2PC-based protocols (enforcing ACID criteria) we note that safeguarding atomicity and consistency of e-business transactions in the presence of failures is at the expense of system resilience and transaction throughput. For example, when web failure occurs or a requested service is unavailable, the only solution is to abort the transaction and all of its component transactions. However, this strategy conflicts with the 'common sense' principle of avoidance of aborting transactions and sub-transactions whenever possible, so as to preserve the valuable results.

2.2 Other approaches

Less conventional TM mechanisms have been proposed for e-business to address the above limitations of 2PC-based protocols and ACID criteria. These include an interesting proposal for a business transaction framework (BTF) for web services [10]. This proposal outlines requirements and characteristics of business transactions, and introduces a set of functional criteria for the proposed BTF. It also analyses current standard initiatives such as Business Process Execution Language (BPEL) for web services, WS-Transactions, WS-Coordination, etc. IBM and Microsoft have introduced WS-Transactions [8] and WS-Coordination [16] that aim to address issues related to 2PC-based protocols, such as TIP (as described above [5]). These approaches aim to define a framework for providing transactional coordination of participants of services offered by multiple autonomous businesses that are based on web services technologies. They also provide support for

long-running business activities in addition to the short-lived transactions. Further, [17] describes a framework called WSTx (web services transactions) for web services, and introduces the concept of transactional attitudes. This approach requires web service clients to declare their transactional requirements and web service providers to declare their individual transactional capabilities and semantics.

The above approaches are interesting and possibly have greater potential than the more conventional approaches previously reviewed. However, as yet, they are limited to the specification of frameworks within which TM can be conducted. To our knowledge these approaches do not provide formal treatments of the TM protocols or any rigorous analysis and evaluation in terms of performance and resilience of e-business transactions.

A further alternative is the agent-based approach [18–20], in which multi-agent techniques are applied to transaction management in the e-business applications. For example, [18] incorporates multi-agents to model cooperative transactions in e-commerce. Unlike 2PC protocols, which have a centralised coordinator process, this approach manages transactions using peer-to-peer protocols. Similarly, [20] combines multi-agents and extended transaction models to provide dynamic transaction capabilities to mobile commerce (m-commerce) applications. Further, [19] applies transaction management techniques to provide multi-agents with concurrency and recovery aspects. Agent-based approaches are also interesting, but these provide only the framework within which TM can be conducted.

2.3 Transaction correctness criteria

Much of the above e-business TM is based upon conventional ACID-enforcing database technology, and therefore has well-known limitations. In particular it lacks the resilience required in the web environment to maintain performance in the presence of errors, and is open to abusive competitive sabotage. However, other approaches currently lack the functionality and rigour of database TM. We therefore question whether the problem with current conventional solutions is the validity of the ACID criteria for e-business, rather than the overhead of enforcing them. The over restrictive nature of the ACID test does not fit well within the e-business environment, and can have an adverse impact on system performance. In particular, the blocking of resources, potentially for the duration of long, nested, distributed and highly interactive e-business transactions, is necessary for transaction atomicity but clearly undesirable within an e-business context. Similarly, visibility of intermediate results can be advantageous, where transactions involve cooperation between participants, or where business decisions require only approximate information. Accordingly, we have defined less restrictive transaction correctness criteria for e-business transactions in our previous work [1,21,22], i.e., the SACReD (Semantic atomicity, Consistency, Resiliency, Durability) criteria, of which only semantic atomicity is required (SA).

Semantic atomicity is a weaker requirement than the classical atomicity. Whereas an atomic transaction is guaranteed to complete successfully or not at all, semantic atomicity allows the unilateral commitment of component transactions irrespective of the commitment of their (parent) e-business transaction, with the constraint that information sources must remain consistent after the execution of transaction. Durability, as in the ACID criteria, requires that effects of a committed transaction must be made permanent in the respective databases, even in the case of failures. Resiliency is the ability to commit in spite of failures, and is defined as a desirable, but not mandatory, property.

This is important in e-business environment, as transactions are vulnerable to failures due to the unreliable nature of internet, stronger requirements for local autonomy and consequentially increased likelihood of component transaction failures. Resilience is increased by associating alternative transactions with the component sub-transactions of an e-business transaction.

In our previous work, we have formally specified and verified protocols that enforce SACReD criteria [21,22]. In the remainder of this paper we extend that work, by enhancing the previous protocols to address the specific problems discussed above, relating to blocking, latency, and resilience, within the context of autonomous e-business systems.

3 The LLR protocol for e-business

In this section we describe the LLR protocol for e-business TM. First we establish the basic definitions and concepts (3.1). We then describe the formal specification language, called Calculus of Communicating Systems (CCS), with which the protocol is defined (3.2). Finally we present the formal CCS specification of LLR (3.3).

A formal verification of the LLR protocol is given in [21,22]. Also, LLR has been validated through the development of a prototype e-business TM system, which operates over heterogeneous and autonomous web servers and database systems. The implementation uses Java IDL [23], a CORBA-based software tool.

3.1 Basic definitions and concepts

E-business operation can range over a wide range of new electronic business services over the internet. Accordingly, we define an e-business transaction (denoted as T_{eb}) as the execution of an application which can be divided into well defined units that provide semantically correct transitions between consistent states of the shared databases. An e-business transaction can therefore be decomposed into component transactions, denoted CT_i. A component transaction can be compensated if its effects on the database can be semantically undone by executing a compensating transaction. It can be replaced if there is an associated alternative transaction.

The configuration and general approach of LLR is based on 2PC, with two main modifications to address the problems reviewed in the previous section. Firstly, we introduce flexible component transactions, by allowing alternative transactions to be specified, which can be executed in place of an aborted component transaction. This enhances flexibility and resilience to failures by increasing the number of component transaction sequences that terminate in a commit. Secondly, we relax the atomicity constraint, thus ensuring only semantic atomicity. To this end, we introduce compensating component transactions (as in [2,3]), i.e., ones which semantically reverse the corresponding transaction, and allow component transaction that can be compensated to autonomously commit. Thus, we avoid the blocking effect of 2PC. However, a consequence of relaxing atomicity is the loss of the isolation property, since intermediate transitory result can become exposed to other transactions.

LLR executes within a configuration comprising a process coordinator, referred to as the business transaction coordinator (BTC), which executes at the site where the transaction is submitted, and a set of participant processes, called business transaction

agents (BTAs), which execute at the various sites that are accessed by the transaction. The BTC and BTAs serve as middleware component systems for e-business transactions.

3.2 Calculus of communicating systems

The formal specification of LLR uses the Calculus of Communicating Systems (CCS), an algebra for specifying and reasoning about concurrent and possibly distributed systems [24,25]. CCS algebra provides a set of terms, operators and axioms with which elements of a system can be specified as expressions. The system's behaviour can then be analysed by manipulating those expressions. The basic element in CCS is a uniquely named agent (or process) that exhibits a specified behaviour defined by a set of actions. CCS operators (listed in Table 1) are used to specify the occurrence of those actions. For example, a single agent (of a commit protocol) may be defined to perform a set of actions {decision . send-message}, to make a decision and then send a message to another agent. In particular, the occurrence of actions is specified using the prefix operator, denoted as '**.**'.

Replicas of an agent, can be defined using the CCS relabelling operator, denoted as [ ]. This allows the name of one or more actions to be changed. For example, the following relation

$$P \stackrel{def}{=} Q\ [decision_p/decision_q,\ send\text{-}message_p/send\text{-}message_q]$$

defines an agent P, which is structurally identical to Q but uses the names $decision_p$ and $send\text{-}message_p$ wherever Q uses the names $decision_q$, and $send\text{-}message_q$. Relabelling is very useful when several replicates of an agent are used to define complex behaviour.

Table 1 Description of CCS notation

Notation	*Description*
.	Prefix operator, which represents the occurrence of actions
|	Represents the concurrent execution or synchronisation of agents
+	Describes the choice between two actions
[ ]	Re-labelling operator
$\sum$ Ei	Summation (describes choice between expressions Ei)
$\stackrel{def}{=}$	Defines the behaviour of an agent

3.3 Formal specification of the LLR protocol

In the following we use CCS to formally specify the *business transaction coordinator* (BTC) and *business transaction agents* (BTAs) of the LLR protocol. The CCS specifications are supplemented by informal explanatory notes.

Business transaction coordinator (*BTC*). The BTC has the responsibility of overseeing the execution of an e-business transaction, the component transactions of which are delegated to individual BTAs. Its behaviour, when a new transaction is initiated, is as follows:

1 $BTC \overset{def}{=} newTrans(T_{eb}) \cdot BTC'(T_{eb})$

2 $BTC'(T_{eb}) \overset{def}{=} \overline{s\text{-}write}\,(begin\text{-}of\text{-}T_{eb}) \,.\, Wait\,(0, rec, T_{eb})$

3 where rec = {1, ..., k} is a set of indices of BTAs having sent their votes.

4 $Wait\,(n, rec, T_{eb}) \overset{def}{=} (n = no\text{-}of\text{-}ST \,.\, BTC\text{-}commit\,(T_{eb}))+$

5 $\sum_{k=1}^{no-of-ST} vote_k\,(vote) \cdot ((k \in rec \,.\, ignore(vote))+$

6 $(vote_a \cdot \overline{s\text{-}write}\,(local\text{-}aborted_i) \cdot BTC\text{-}abort(CT_i, T_{eb}))+$

7 $((vote_c \cdot \overline{s\text{-}write}\,(local\text{-}commited_i)) \cdot Wait\,(n + 1, rec, T_{eb})$

In the above procedure, initially, a new e-business transaction, T_{eb}, is assigned to the BTC (line 1). The start of T_{eb} is then recorded by BTC in a log file using a simple write operation and BTC enters into a wait state, pending messages from the BTAs concerning completion of component transactions (lines 2–3). When a vote is received from BTA_k (lines 4–7), the BTC tests if the vote received is a duplicate, i.e., $k \in rec$, in which case it is ignored (line 5). If $k \notin rec$, then BTC acts according to BTC-commit (T_{eb}) to commit T_{eb} (line 4), or BTC-abort (CT_i, T_{eb}) to abort T_{eb} (if CT_i is not replaceable) (line 6). Note that the possibility of duplicated vote arises in the situation when a BTA fails while sending a vote to the BTC. Thus the recovery process (detailed in [21,22]) forces the BTA to re-send the commit/abort vote to BTC.

The BTC commit procedure is as follows:

$$BTC\text{-}commit\,(T_{eb}) \overset{def}{=} \overline{f\text{-}write}\,(commit\text{-}decision) \,.\, send\text{-}commit\,(0, no\text{-}of\text{-}CT)$$

$$send\text{-}commit\,(i, no\text{-}of\text{-}CT) \overset{def}{=} (i = no\text{-}of\text{-}CT \cdot Global\text{-}commit) + \sum_{k=1}^{no-of-CT} \overline{g-commit}_k \cdot send\text{-}commit\,(i + 1, no\text{-}of\text{-}CT)$$

$$Global\text{-}commit \overset{def}{=} \overline{Terminate}\,(T_{eb}) \cdot BTC$$

In the above commit procedure, BTC forcibly writes the commit decision and sends global commit messages to all BTAs. It then terminates T_{eb} according to Global-commit and starts processing a new e-business transaction. Importantly, BTC does not require acknowledgements from BTAs regarding the commit of CT_i as they are already (unilaterally) committed.

The BTC aborts T_{eb} according to the following procedure:

$$BTC\text{-}abort(CT_i,T_{eb}) \overset{def}{=} (replaceable\,(CT_i) \cdot Wait\,(n, rec, T_{eb})) + \overline{f\text{-}write}\,(abort\text{-}decision) \cdot send\text{-}abort\,(0, no\text{-}of\text{-}CT)$$

Note that, in the above abort procedure, if a replaceable component transaction, CT_i, is aborted, then an alternative component transaction is initiated. BTC then waits for each

BTA's decision regarding commitment or abortion of the alternative component transaction. If CT_i cannot be replaced then BTC forcibly writes the abort decision and proceeds accordingly to send-abort(i, no-of-CT) as follows:

$$\text{send-abort (i, no-of-CT)} \stackrel{\text{def}}{=} (\text{i = no-of-CT} \,.\, \text{Global-abort}) + \sum_{k=1}^{\text{no-of-CT}} \overline{\text{g-abort}_k} \,.\, \text{send-abort (i + 1, no-of-CT)}$$

$$\text{Global-abort} \stackrel{\text{def}}{=} \overline{\text{Terminate}}\ (T_{eb}) \,.\, \text{BTC}$$

In the above, BTC sends global-abort messages to all the BTAs, and terminates the transaction.

Business transaction agent (*BTA*): In the LLR protocol, BTAs exhibit similar characteristics. Thus, we model a single generic BTA and then make use of the CCS re-labelling operator [] to define individual BTAs.

Each BTA manages the execution of its component transaction, CT_i as follows:

$$\text{BTA}(CT_i) \stackrel{\text{def}}{=} \overline{\text{s-write}}\ (\text{begin-of-}\ CT_i) \,.\, \text{WTA-process}\ (CT_i) \qquad (1)$$

$$\text{BTA-process}(CT_i) \stackrel{\text{def}}{=} \text{execute}(CT_i) \,.\, \text{Voting}\ (CT_i) + (\text{g-abort} \,.\, \text{abort}\ (CT_i) \,.\, \overline{\text{s-write}}\ (CT_i\text{-global-abort}) \,.\, \overline{\text{Terminate}}\ (CT_i) \,.\, 0)$$

Initially, when a new CT_i is assigned to a BTA, it records the start of CT_i in the log file using simple write operation and then starts the processing (equation (1)).

After executing CT_i, BTA sends a vote to BTC. Also, BTA can receive a global-abort message from BTC amid the processing of CT_i (equation (2)). If that occurs, BTA must abort and terminate CT_i. This situation can arise when BTC receives an abort vote from another BTA, in which case BTC must send global abort messages to all BTAs.

The BTA voting decision follows the following procedure:

$$\text{Voting}(CT_i) \stackrel{\text{def}}{=} (\text{commit}(CT_i) \,.\, \text{local-committed}\ (CT_i)) + \text{local-aborted}\ (CT_i)$$

$$\text{local-aborted}\ (CT_i) \stackrel{\text{def}}{=} \overline{\text{f-write}}\ (\text{abort-decision}) \,.\, \overline{\text{vote}}\ (\text{abort}) \,.\, \overline{\text{Terminate}}\ (CT_i) \,.\, 0$$

Note that BTA forcibly writes the abort of CT_i, sends an abort vote to BTC, and declares CT_i as local-aborted. It then terminates CT_i , and stops processing.

A BTA can locally commit, as follows:

$$\text{local-committed}(CT_i) \stackrel{\text{def}}{=} \overline{\text{f-write}}\ (\text{commit-decision}) \,.\, \overline{\text{vote}}\ (\text{commit}) \,.\, \text{Wait(BTC-decision)}$$

In the above, the BTA forcibly writes a commit decision, sends a commit vote to BTC and then waits for BTC's decision, as follows:

$$\text{Wait(BTC-decision)} \stackrel{\text{def}}{=} (\text{g-commit} \,.\, \text{global-commit}(CT_i)) + (\text{g-abort} \,.\, \text{global-abort}\ (CT_i))$$

$$\text{global-commit}\ (CT_i) \overset{def}{=} \overline{\text{s-write}}\ (CT_i\text{-global-commit}) \,.\, \overline{\text{Terminate}}\ (CT_i) \,.\, 0$$

Note that, after receiving the global commit decision, the BTA simply writes the global commit of CT_i and changes the status of CT_i from local-committed to global-committed.

In the case of a global abort decision from the BTC, the BTA will execute the following procedure:

$$\text{global-abort}\ (CT_i) \overset{def}{=} \text{local-committed}\ (CT_i) \,.\, \text{compensate}(\text{comp}(CT_i))$$

$$\overline{\text{s-write}}\ (CT_i\text{-compensated}) \,.\, \overline{\text{Terminate}}\ (CT_i) \,.\, 0$$

In the above, if CT_i is locally-committed and BTA receives an abort message, BTA must execute the compensating transaction for CT_i. This is logged by BTA by simply writing the compensation decision and then marking the end of CT_i. It is to be noted that BTA does not send commit and abort acknowledgements to BTC, as CT_i has either unilaterally committed or aborted. LLR therefore removes the need for acknowledgement messages, in a similar manner to the presumed abort and commit protocols.

Now we define the behaviours of individual BTAs as instances of the above generic BTA using the relabelling function [] of CSS, as follows:

$$BTA_k\ (k = 1, ..., n) \overset{def}{=} BTA[process_k/process, vote_k/vote, compensate_k/compensate,$$
$$g\text{-}commit_k/g\text{-commit}, g\text{-}abort_k/g\text{-abort}]$$

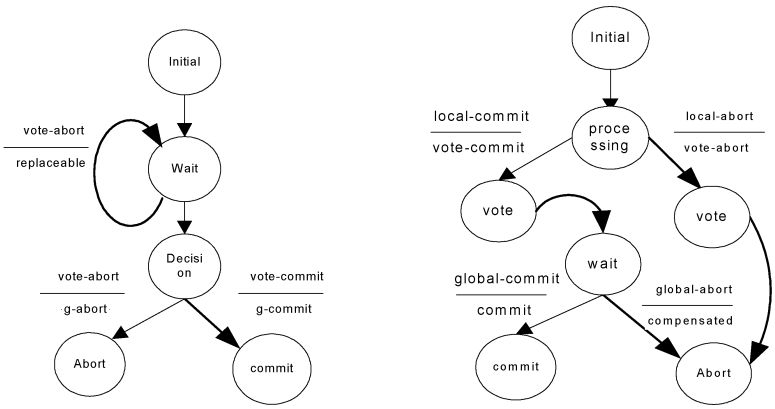
In the above, a label pair such as $process_k$/process defines the mapping of *process* to $process_k$, *vote* to $vote_k$, and so on. This defines the behaviour of n WTAs which are involved in the execution of component transactions CT_i.

Now that we have specified the behaviours of WTC and WTAs, the LLR protocol is defined as follows:

$$LLR \overset{def}{=} (BTC \mid BTA_1 \mid BTA_2 \mid BTA_3, ..., BTA_k)$$

The state transition diagrams for BTC and BTAs are given in Figure 1 [21], in which edge labels represent (above the line) the reason for the transition, i.e., the message (or an action taken), and (below the line) the message that is sent as a result.

Figure 1 State transition diagram of LLR protocol



4 Evaluation of the protocols

This section evaluates LLR protocol in comparison to the existing protocols in terms of latency, resource blocking, and resilience to failures.

The time complexity analysis (as in [3,11,26]), of LLR, 2PC and PA, evaluates their relative performance in terms of system resource blocking (or latency), i.e., the delay required to release the system resources. System resources are held by e-business transactions, for example, when a participant waits, in a prepared-to-commit state, to receive a decision from the coordinator. Latency is calculated as the sum of the delays incurred by the participating systems, i.e.:

- The time (P) each participant takes to process a component transaction, excluding the time required for forced-write operations and the message arrival which are treated separately.
- The time (M) a coordinator and each participant take to exchange messages.
- The time (W) a coordinator and each participant take in force-writing a commit or abort decision. Note that forced-write operations significantly affect the performance of the protocols. This is because the communication between participating e-business systems is suspended until the required information is written to the persistent storages such as log files of the web servers or database systems.

We assume (as in [11,26]) that the coordinator sends (and receives) messages to (from) all the participants concurrently, and also that participants process the component transactions and perform the write operations in parallel. Accordingly, the latency for each of the protocols is calculated in the following subsections.

4.1 Latency of the protocols

4.1.1 2PC protocol

Commit case. In this case the time required to release the system resources is calculated as $3 \times M + 2 \times W + P$. Each participant receives two messages (preparation and decision) and sends one message (local commit vote) to the coordinator in order to release the systems resources held by a component transaction. In addition, the coordinator and each participant take time W to respectively force write the global commit/abort decision and the prepare-to-commit state to a log file. Further, each participant takes a delay of P to process a component transaction.

We now consider the two situations in which an abort can occur.

Abort case-1. In this case, it is assumed that at least one of the participants is in the prepare-to-commit state. This is because the participants, having no prepare-to-commit states, can unilaterally abort and terminate their component transactions and will not enter into the second phase of the protocol. In this case, 2PC takes the delay of $3 \times M + 2 \times W + P$ to release the system resources. If at least one participant is in the prepare-to-commit state, then the same number of messages (as that of commit case) are communicated between the coordinator and the participants to release the system resources.

Abort case-2. In this case, it is assumed that none of the participants is in the prepare-to-commit state. Consequently the participants can unilaterally abort and terminate their component transactions and will not enter into the second phase of the protocol. Thus 2PC incurs a delay of W + P to release the system resources. In this case participants are not required to wait for the coordinator messages if they are going to abort the transaction. Each participant takes the delay of P, which is the time required to process the component transaction.

4.1.2 PA protocol

Commit case. PA behaves similar to 2PC in the commit case. PA therefore takes the delay of $3 \times M + 2W + P$ to release the system resources.

Abort case-1. For the above reason, it is assumed that at least one of the participants is in the prepare-to-commit state. Thus the delay required to release the system resources is $3 \times M + W + P$. In this case, PA reduces the delay by one W, as the coordinator does not force-write the abort decision to persistent storage.

Abort case-2. If none of the participant is in the prepared-to-commit case then PA takes the delay of W + P to release the system resources (as is the abort case-2 of 2PC).

4.1.3 LLR protocol

Commit case. The LLR protocol allows the unilateral commitment of component transactions. Participants therefore release the system resources as soon as they complete executing their component transactions. They do not wait for the global decision from the coordinator to release the system resources. LLR therefore takes the delay of W + P to release the system resources; where P is the time required to process a component transaction and W is the time required to force-write the commit decision. LLR does not need any message to be communicated between the participants and coordinator to release the system resources.

Abort case. In the abort case, LLR releases the system resources after the delay of $2 \times W + 2 \times P$. In this case, LLR needs additional P and W to execute the compensating transactions, if any of the component transaction is unilaterally committed. This is because, system resources will be held twice; once by the original component transaction and then by the compensating transaction. No delays are incurred on account of messages, as in the commit case.

4.2 Case studies

We now compare the relative performance of 2PC, PA and LLR within five test cases. These cases use the estimated values of M, P, and W, as shown in Tables 2 and 3. Determining the exact values of M, P, and W depend on various factors. For example, the message delay, M, depends on the type and location of participating systems, the type of internet links (slow or fast), etc. According to [26] a message takes 50 ms on average to be sent on the internet across the USA. Our own measurement, using the ping utility of RTT (round trip time) for internet access across the world demonstrate predictable increases in RTT for intercontinental access. For example, average RTT was found to be 90 ms between web servers located in Coventry, UK, and Pennsylvania, USA, and

300 ms between Singapore and Coventry. We therefore assume 100 ms and 300 ms, respectively as minimum and maximum average RTT values for M. Similarly, the processing time of a component transaction, P, depends on various factors such as system load, the nature of the application, etc. For example, some applications involve a larger number of I/O operations and longer CPU time than others [27]. We therefore take 50 and 100 ms, respectively as minimum and maximum values for P. Further, the values of W (forced-write delay) are also estimated. The values assumed for W are 15 ms and 35 ms, in line with estimates in [27].

Note that the above values of M, P, and W are estimated values and these vary according to the nature of the environment in which e-business transactions operate.

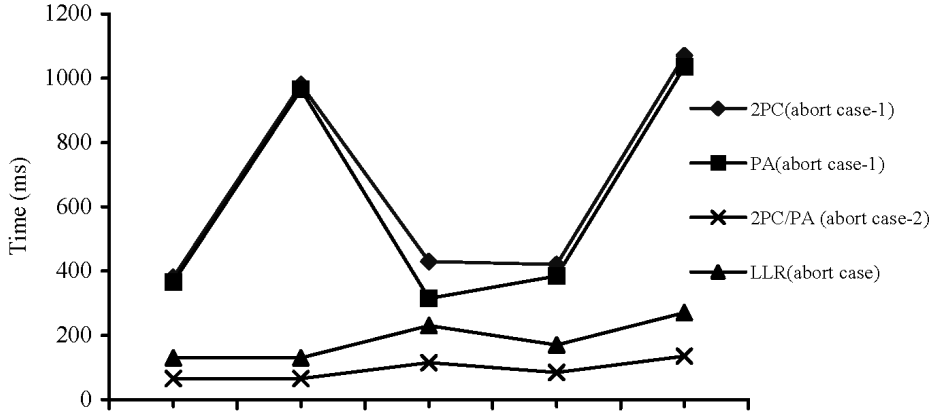
4.2.1 Abort case

The computed abort case latency (delays), incurred by 2PC, PA, and LLR, to release the system resources are respectively listed and shown graphically in Table 2 and Figure 2.

Table 2 Total delay required to release system resources (abort case)

Case	*M* (ms)	*W* (ms)	*P* (ms)	*2PC (abort case-1)* (*3M + 2W + P*)	*PA (abort case-1)* (*3M + W + P*)	*2PC/PA (abort case-2)* (*W + P*)	*LLR* (*2W + 2P*)
1	100	15	50	380	365	65	130
2	300	15	50	980	965	65	130
3	100	15	100	430	315	115	230
4	100	35	50	420	385	85	170
5	300	35	100	1070	1035	135	270

Figure 2 Delay required to release the system resources in abort case



From Figure 2, it is clear that in the abort case-2 (unilateral abort), 2PC and PA incur shorter delays to release the system resources, compared to LLR. This is because LLR incurs an extra delay to process compensating transactions, when an e-business transaction is globally aborted. It is to be noted that such a situation (i.e., abort case-2) happens very rarely, where all participating systems of 2PC or PA do not enter the prepare-to-commit state and that they abort their component transactions. The most

common is the abort case-1. In the abort case-1, both 2PC and PA perform poorer than LLR, as they are not allowed to abort the component transactions unilaterally in a prepare-to-commit state. The participants must therefore wait for the coordinator's decision to release the system resources. Consequently, 2PC and PA incur greater delays compared to LLR, as they are required to exchange messages between the coordinator and participants.

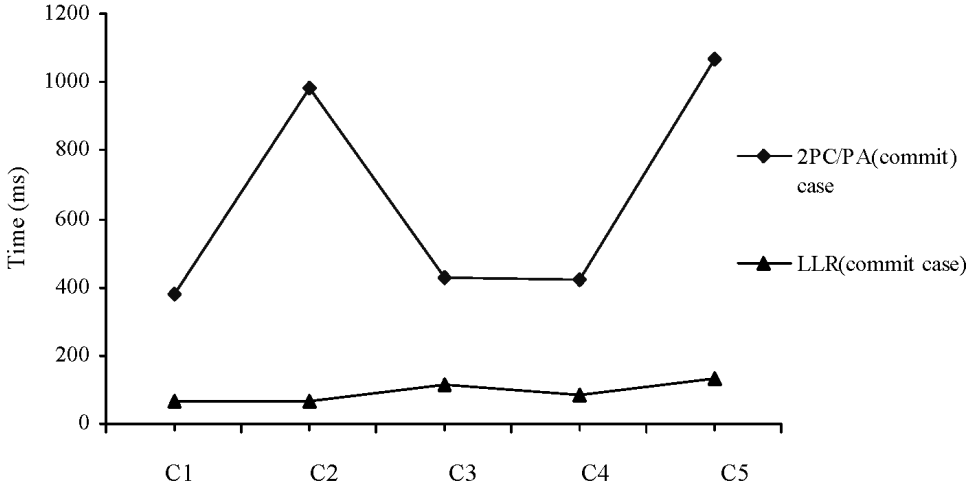
4.2.2 Commit case

The delay required by the protocols 2PC, PA, and LLR, to release the system resources in the commit case, are respectively listed and shown graphically in Table 3 and Figure 3.

Table 3 Total delay required to release the system resources (commit case)

Case	*M* (ms)	*W* (ms)	*P* (ms)	*2PC/PA* (*3M* + *2W* + *P*)	*LLR* (*W* + *P*)
1	100	15	50	380	65
2	300	15	50	980	65
3	100	15	100	430	115
4	100	35	50	420	85
5	300	35	100	1070	135

Figure 3 Delay required to release the system resources in commit cases



It is clear from Figure 3 that LLR significantly reduces the latency required to release the system resources as compared to 2PC and PA protocols in the commit case. This significant difference arises mainly because of the number of messages communicated between the coordinator and the participants to release the system resources. As described earlier, 2PC and PA require $3 \times M$ messages while LLR does not require any messages to be communicated between the coordinator and participants. Consequently LLR reduces the delay required to release the system resources. It is to be noted that in LLR, the coordinator does not send a prepare message to the participants, as in implicit yes vote (IYV) protocol [28]. In addition, participants release the system

resources as soon as they complete the execution of the component transactions. The participants therefore do not wait for the coordinator's decision to locally commit/abort their component transactions.

From the above evaluation it is shown that the LLR protocol incurs low latency and avoid resource blocking as compared to the existing protocols for e-business transactions.

4.3 Resiliency evaluation

The evaluation of the resiliency of the LLR protocol in comparison to current 2PC-based approaches is based on the following commit and abort probabilities [29] of the e-business transactions:

- *Actual commit/abort probability*: this refers to the commit/abort probability of the e-business transaction where no alternative transactions are involved.
- *Total commit/abort probability*: this refers to the commit/abort probability of the e-business transaction where alternative transactions are involved.
- *Existential probability*: this refers to the probability of the existence of alternative transactions. It determines, on the average, the ratio of alternative transactions in the overall e-business transaction.

The fundamental principle on which the evaluation is based is that, given the abort and existential probabilities, resiliency of a protocol is determined by the total commit probability of an e-business transaction. Specifically, an increase in resiliency indicates an increase in the probability that the e-business transaction will commit.

We conduct two case studies to evaluate the resiliency of LLR protocol in comparison to the current protocols, using different existential probabilities of 0.3 and 0.5 – showing that there exist 30% and 50% alternative transactions in an e-business transaction. In each case, different values of the actual commit/abort probabilities are also used in conjunction with a particular existential probability. Within the context of these cases various situations are considered. For example, the worst case scenario where the abort probability of e-business transaction is very high, and also the best case scenario where the commit chances are high. To model the best case scenario, the actual commit probability is kept high showing that abort chances of e-business transaction are low. The commit probability is then gradually reduced such that the abort probability of e-business transaction reaches the highest level showing the worst case scenario.

The total commit/abort probabilities, of an e-business transaction, computed using different existential probabilities, are shown in Tables 4 and 5. In these tables the following abbreviations are used:

CP is the actual commit probability of the e-business transaction where no alternative transactions are involved.

AP is the actual abort probability of the e-business transaction where no alternative transactions are involved.

P (Alt-ST) This is the probability of the existence of alternative transactions in the overall e-business transaction.

CP (Alt-ST) refers to the commit probability of alternative transactions.

AP (Alt-ST) refers to the abort probability of alternative transactions.

TAP is the total abort probability of the e-business transaction where alternative transactions are involved.

TCP is the total commit probability of the e-business transaction where alternative transactions are involved.

Table 4 Calculation of total commit probabilities (existential probability = 0.3)

CP	*AP*	*P (Alt-ST)*	*CP (Alt-ST) = CP × AP × P (Alt-ST)*	*AP (Alt-ST) = AP × AP × P (Alt-ST)*	*TAP = 1 – CP + CP (Alt-St)*	*TCP = 1 – TAP*
1.0	0.0	0.3	0.000	0.000	0.000	1.000
0.9	0.1	0.3	0.027	0.003	0.073	0.927
0.8	0.2	0.3	0.048	0.012	0.152	0.848
0.7	0.3	0.3	0.063	0.027	0.237	0.763
0.6	0.4	0.3	0.072	0.048	0.328	0.672
0.5	0.5	0.3	0.075	0.075	0.425	0.575
0.4	0.6	0.3	0.072	0.108	0.528	0.472
0.3	0.7	0.3	0.063	0.147	0.637	0.363
0.2	0.8	0.3	0.048	0.192	0.752	0.248
0.1	0.9	0.3	0.027	0.243	0.873	0.127
0.0	1.0	0.3	0.000	0.300	1.000	0.000

Table 5 Calculation of total commit probabilities (existential probability = 0.5)

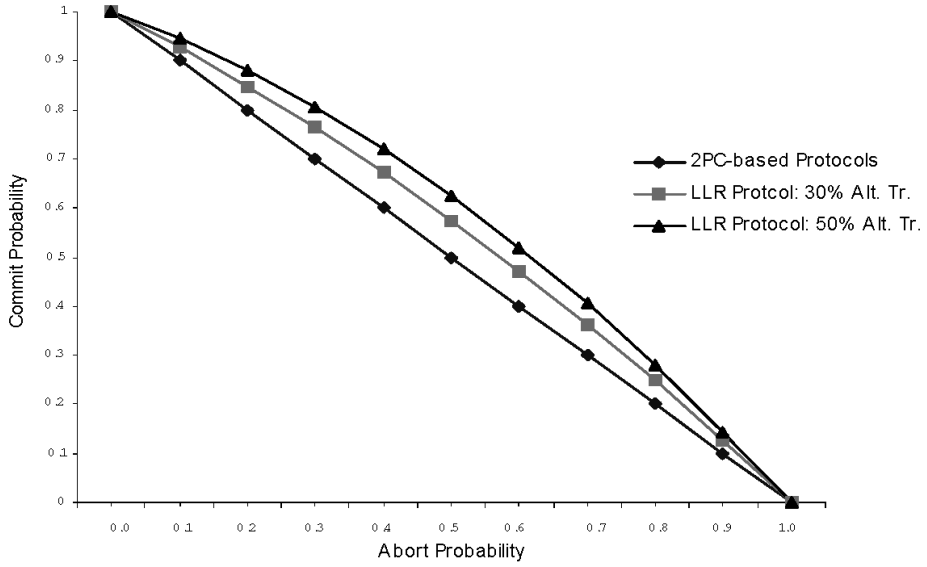
CP	*AP*	*P (Alt-ST)*	*CP (Alt-ST) =} CP × AP × P (Alt-ST)*	*AP (Alt-ST) = AP × AP × P (Alt-ST)*	*TAP = 1 – CP + CP (Alt-St)*	*TCP = 1 – TAP*
1.0	0.0	0.5	0.000	0.000	0.000	1.000
0.9	0.1	0.5	0.045	0.005	0.055	0.945
0.8	0.2	0.5	0.080	0.020	0.120	0.880
0.7	0.3	0.5	0.105	0.045	0.195	0.805
0.6	0.4	0.5	0.120	0.080	0.280	0.720
0.5	0.5	0.5	0.125	0.125	0.375	0.625
0.4	0.6	0.5	0.120	0.180	0.480	0.520
0.3	0.7	0.5	0.105	0.245	0.595	0.405
0.2	0.8	0.5	0.080	0.320	0.720	0.280
0.1	0.9	0.5	0.045	0.405	0.855	0.145
0.0	1.0	0.5	0.000	0.800	1.000	0.000

Note that, if the actual commit probability is 1.0 (or 100%), then each transaction will certainly commit. Consequently, the commit probability of alternative transactions is zero. Similarly, if the actual abort probability is 1.0, then a transaction will certainly be aborted. In that case the commit probability of the e-business transaction is zero. This is the situation where a transaction is aborted due to failures, such as unavailability of the requested services, or site or communication failures.

Figure 4 graphically represents the total commit probability of e-business transactions calculated in the aforementioned cases. It also shows the 2PC-based protocols which do not support alternative transactions.

The graph in Figure 4 clearly indicates that LLR protocol with alternative transactions enhances the resiliency of the e-business transactions. The graph shows that the greater the commit probability of the e-business transaction, the greater the resiliency. Comparing the resiliency of the protocols under consideration, it is observed that resiliency increases with the increase in existential probability; where the average number of alternative transactions in a particular e-business transaction is high. For example, with the existential probability of 0.3, the maximum gain achieved in total commit probability is 0.075 as shown in Table 4. Similarly, the alternative transactions with existential probability of 0.5 increase the commit probability by 0.125 (maximum gain) as shown in Table 5. The above cases prove that LLR increases the resiliency of the e-business transactions as compared to current approaches based on 2PC-based protocols which do not support alternative transactions.

Figure 4 Evaluation of the resiliency of LLR and 2PC-based protocols



5 Conclusion

For conventional database TM to be adapted for e-business, it is necessary to improve its performance within a federated web-based environment, and particularly to remove the potential for performance manipulation for advantage over competitor systems. Also, the validity of the transaction correctness criteria needs to be considered within the e-business context. Our proposed LLR e-business TM protocol addresses these problems. Specifically, it enforces the SACReD criteria, a relaxed version of the ACID criteria, which we argue is more appropriate for e-business. Within these looser constraints, we have increased resilience through the use of alternative component transactions, and reduced delays and resource blocking, through the introduction of unilateral commit and compensation of component transactions. Our performance and resiliency evaluation has demonstrated significant improvements over existing e-business transaction protocols.

References

1 Younas, M., Eaglestone, B. and Holton, R. (2000) 'A review of multidatabase transactions on the web: from the ACID to the SACReD', *Proceedings of the 17th British National Conference on Databases* (*BNCOD*), Exeter, UK, Springer LNCS, pp.140–152.

2 Elmagarmid, A.K. (Eds.) (1992) *Database Transaction Models for Advanced Applications*, Morgan Kaufmann Publisher.

3 Gray, J. and Reuter, A. (1993) *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann Publishers.

4 Billard, D. (1998) 'Transactional services for the internet', *Proceeding of the International Workshop on Web and Database* (*WebDB'98*), Valencia, Spain, pp.14–33.

5 Lyon, J., Evans, K. and Klein, J. (1998) *Transaction Internet Protocol: Version 3.0*, Internet-Draft, April, http://www.ietf.org/ids.by.wg/tip.html.

6 Little, M., Shrivastava, S., Caughey, S. and Ingham, D. (1997) 'Constructing reliable web applications using atomic actions', *WWW6/Computer Networks and ISDN Systems*, Vol. 29, Nos. 8–13, pp.1281–1290.

7 Little, M. and Shrivastava, S. (1998) 'Java transactions for the internet', *Distributed Systems Engineering*, Vol. 5, No. 4, pp.156–167.

8 Cabrera, F., Copeland, G., Cox, B., Freund, T., Klein, J., Storey, T. and Thatte, S. (2002) *Web Services Transaction (WS-Transaction)*, http://www-106.ibm.com/developerworks/library/ws-transpec/.

9 Kappel, G., Rausch-Schott, S. and Retschitzegger, W. (1999) 'Transaction support for dataweb applications – a requirement's perspective', *Proceeding of Fifth American Conference on Information Systems* (*AMCIS'99*), Milwaukee, Winconsin, USA, pp.877–879.

10 Papazoglou, M. (2003) 'Web services and business transactions', *World Wide Web*, Vol. 6, No. 1, pp.49–91.

11 Berstein, P., Hadzilacos, V. and Goodman, N. (1987) *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, USA.

12 Ozsu, T. and Valduriez, P. (1991) *Principles of Distributed Database Systems*, Prentice-Hall.

13 Ehmayer, G., Kappel, G. and Reich, S. (1997) 'Connecting databases to the web: a taxonomy of gateways', *Proceedings of 8th International Conference on Database and Expert Systems Applications* (*DEXA 1997*), Toulouse, France, Springer LNCS, Vol. 1308, pp.1–15.

14 Barker, R., Slothouber, L., Tracy, M. and Ware, S. (1997) *Professional Web Site Optimization*, Wrox Press Inc.

15 Kempster, T., Stirling, C. and Thanisch, P. (1999) 'A critical analysis of the transaction internet protocol', *Proceeding of the 2nd International Conference on Telecommunication and Electronic Commerce* (*ICTEC 1999*), Nashville, USA.

16 Cabrera, F., Copeland, G., Freund, T., Klein, J., Langworthy, D., Orchard , D., Shewchuk, J. and Storey, T. (2003) *Web Services Coordination (WS-Coordination)*, http://www-106.ibm.com/developerworks/library/ws-coor/.

17 Mikalsen, T., Tai, S. and Rouvellou, I. (2002) 'Transactional attitudes: reliable composition of autonomous web services', *Proceedings of the Workshop on Dependable Middleware-based Systems (WDMS 2002) at the Dependable Systems & Network Conference (DSN 2002)*, Bethesda, MD, USA.

18 Chen, Q. and Dayal, U. (2000) 'Multi-agent cooperative transactions for e-commerce', *Proceedings of the 7th International Conference on Cooperative Information Systems* (*CoopIS 2000*), Eilat, Israel, pp.311–322.

19 Nagi, K. (1999) 'Transactional agents: a robust approach for scheduling orders in a competitive just-in-time manufacturing environment', *Proceedings of the Workshop on MAS in Logistic and Economical Perspectives of Agents on Conceptulisation*, Germany.

20 Younas, M., Chao K-M. and Anane, R. (2003) 'M-commerce transaction management with multiagent support', *Proceedings of 17th International Conference on Advanced Information Networking and Applications* (*AINA 2003*), IEEE CS Press, Xi'an, China, pp.284–287.

21 Younas, M. and Eaglestone, B. (2002) 'A formal verification strategy for crash recovery in web-database applications', *Proceeding of 3rd International Conference of Web Information System Engineering* (*WISE 2002*) *Workshops*, IEEE CS Press, Singapore, pp.113–119.

22 Younas, M., Eaglestone, B. and Holton, R. (2000) 'A formal treatment of a SACReD protocol for multidatabase web transactions', *Proceedings of the 11th International Conference on Database and Expert Applications* (*DEXA 2000*), Greenwich, London, Springer LNCS, pp.899–908.

23 Lewis, G., Barber, S. and Siegel, E. (1998) *Programming with JavaTM IDL: Developing Web Applications with Java and CORBA*, John Wiley & Sons Inc.

24 Bruns, G. (1997) *Distributed Systems Analysis with CCS*, C.A.R. Hoare Series Editor, Prentice Hall.

25 Milner, M. (1989) *Communication and Concurrency*, C.A.R. Hoare Series Editor, Prentice Hall.

26 Zhang, Z., Perrizo, W. and Shi, V. (1999) 'Atomic commitment in database systems over wide area active networks', *IEEE International Conference on Data Engineering* (*ICDE 1999*), Sydney, Australia.

27 Spiro, P.M., Joshi, A.M. and Rengaranjan, T.K. (1991) 'Designing an optimised transaction commit protocol', *Compaq Digital Technical Journal*, Vol. 3, No. 1, Winter, http://www.research.compaq.com/wrl/DECarchives/DTJ/DTJ100/.

28 Al-Houmaily, Y. and Chrysanthis, P. (1995) 'Two-phase commit in gigabit-networked distributed databases', *Proceeding of 8th ISCA International Conference on Parallel & Distributed Computing Systems*, pp.554–560.

29 Gupta, R., Haritsa, J.R. and Ramamritham, K. (1997) 'Revisiting commit processing in distributed database systems', *ACM-SIGMOD Record*, Vol. 26. No. 2, pp.486–497.