WEB SERVICE TRANSACTION MANAGEMENT

Frans A. Henskens
School of Electrical Engineering & Computer Science, University of Newcastle, Callaghan 2308, Australia

Keywords: Web services, transactions, concurrency control, Internet, component-based software.

Abstract: This paper describes extension of the functionality of conventional web browsers to produce a new enhanced web browser. Each instance of this enhanced browser is part of a federation of browser instances that use a directed graph-based technique to provide transaction and hence concurrency control over access to web services. These *'super browsers'* communicate with web-based services across the Internet, application code that may be obtained from the Internet but then executes as a local program, and with other browser instances.

1 INTRODUCTION

The Internet pervades modern society, providing connected users with access to: huge amounts of textual information; resources such as data files containing computer programs, encoded music and picture data; and email. Most Internet users interact with the Internet using programs called web browsers (e.g. Mozilla (Mozilla Foundation, 2006), Internet Explorer (Microsoft Corporation, 2007b), Safari (Apple Inc., 2007), Opera (Opera Software ASA, 2007)) and mail handlers (e.g. Eudora (Qualcomm Incorporated, 2007), Outlook (Microsoft Corporation, 2007a)). As suggested by their name, mail handlers provide users with the ability to send and receive items of electronic mail. Web browsers allow users to display screens of formatted text and images, use hyperlinks (Nelson, 1965) to move between such pages, interact with active content provided by plug-ins such as Flash (Adobe Systems Incorporated, 2007) and Java (Sun Microsystems Inc., 2007), and interact with external (to the browser) services such as booking systems that provide an interface to underlying data management systems.

Web services allow Internet users to communicate data without the need for intimate knowledge of each others' computer systems or application software. Interaction may be business-to-business, commonly achieved using remote procedure call (Birrel and Nelson, 1984). Increasingly the interaction is between businesses that provide a service or services, and clients who wish to use the service(s). This client-service interaction involves the use of documents in the form of HTML (Raggett et al., 1999) web pages, displayed on the client's computer using a web browser. Standards such as XML (Bray et al., 2006) for data tagging, SOAP (Gudgin et al., 2003) for data transfer, UDDI (OASIS UDI, 2004) for service advertising and WDSL (Christensen et al., 2001) for service description, facilitate use of web services. A common means for client utilisation of a web service involves the client using a web browser to download and display the service provider's web page, which then controls the interaction between the client and the service.

This paper describes extending the functionality of conventional web browsers to produce a new enhanced form of browser. Each instance of this enhanced browser is part of a federation of browser instances that provides transaction-like control over access to web services. These *'super browsers'* communicate with web-based services across the Internet, application code that may be obtained from the Internet but then executes as a local program, and with other browser instances. This architecture provides for heterogeneity of implementation of the new super browser, web services and the user application code.

2 CLIENT-SERVICE INTERACTION

It is interaction with external programs that underpins the motivation for the suggestion of a

A. Henskens F. (2007).
WEB SERVICE TRANSACTION MANAGEMENT.
In *Proceedings of the Second International Conference on Software and Data Technologies - PL/DPS/KE/WsMUSE*, pages 112-119
DOI: 10.5220/0001327901120119

super browser. Increasingly, users of Internet services do so in combination with their use of other services offered by unrelated providers. Additionally, access to such services is typically not gained using a single interface – rather the user is required to separately connect to each service provider, and then use the interface provided by that provider's web site. In this situation it is typical that no 'behind the scenes' service provider to service provider interaction would occur to, for instance, coordinate the client's use of the services.

For example a holiday-maker may wish to organise an itinerary involving booking an airline flight, bus transport to a hotel, accommodation at the hotel, day trip, etc. At issue is the fact that the airline, bus company, hotel and excursion company are typically independent of each other, and implement separate booking systems. The holiday-maker, then, has to separately affect bookings with each company's web-based system. If the user finds that a particular booking is not possible (often through being fully booked), it may be necessary for him or her to undo previous bookings and start the whole process again. Alternately the user may utilise multiple browser windows to simultaneously access each of the service providers, but may find that, in the time taken to peruse other servers, a previously available service has become fully booked. In centralised systems this problem would be solved using a mechanism called a transaction (Härder and Reuter, 1983).

The high levels of utilisation of the Internet to conduct both business and pleasure make solving this problem extremely important. The business world desperately needs a way of controlling interaction between services, and high-level analysis of such needs have been the subject of numerous publications, for example (Hanson et al., 2002, Juric et al., 2006, Lazovik et al., 2003). To date, attempts to control such interactions have only applied to business processes for which the activities involve interaction with Web services only, and assume no human interaction. This belies the fact that people often participate in such interactions (IBM and SAP AG, 2005).

In the following sections the notion of transactions is developed, followed by a model for control of web-based transactions. Both the inclusion of human-machine interactions and the provision of a software layer providing support for applications represent significant achievements of the model.

2.1 Transactions

Transactions involve a co-ordinating entity interacting with the various components to ensure that services continue to be available for the life of the transaction, and either all components (services) involved in the transaction successfully complete (i.e. in the above example that all bookings are made) or that none of them complete.

Continuing the previous example, the holiday-maker may check availability of the various components of their holiday before making any bookings. After being satisfied that the desired itinerary is possible, he/she would go back and make the appropriate bookings. This scenario could lead to a second issue, that of control over concurrent access to the resources. In this case, the holiday-maker could attempt to book their flight, only to find that some other person has taken the last seat in the time that has elapsed between determining availability and booking the itinerary. Dealing with this and similar issues is termed concurrency control (Peinl, 1983).

Transaction semantics have long been used to control user interaction with multiple entities in a data store, and also to prevent concurrent users of the store from adversely affecting each other (Farrag, 1989). The theory specifies that all transactions must exhibit the ACID (atomicity, consistency, isolation and durability) properties, the isolation property ensuring that transactions are serializable and therefore that they manage concurrent use of the data in the store (Date, 1999). Traditionally, access to data has been controlled by a data base management system (DBMS) (Date, 1999) implementing either a pessimistic (Gray and Reuter, 1993), optimistic (Kung and Robinson, 1981), or combination-of-the-two approach (Momin and Vidyasankar, 2000) to transaction and hence concurrency control. Thus transaction management has been achieved using a centralised piece of software (the DBMS). With the advent of distributed databases, transaction control has been achieved through the use of messages between DBMS instances that effectively create a single DBMS spanning the networked computers.

Middleware systems such as CORBA (Siegel, 1996) support distributed access to services provided by objects resident at possibly different networked hosts. Transaction-based concurrent access to such objects is supported through the use of a special service implemented in the Object Request Broker (ORB) (Object Management Group, 1998), which also requires specific functionality from the participating server objects (Emmerich, 2000). In both this and the traditional DBMS approaches,

centralised control (in these examples either the DBMS or the ORB) underpins the transaction system. Such centralised control is not available when the services involved in the transaction are heterogeneous in implementation, as is the case for the typical service available on the Internet.

The following section discusses the use of cooperating web browser instances to provide the abstraction of centralised control of transactions involving use of web services.

2.2 Distributed Transaction Control

The notion of using cooperating browser instances to control transactions is similar to that which underpins distributed operating systems (distributed OS) (Sinha, 1996); the operating system (OS) instances are independently executing entities that communicate as necessary to provide the required level of global coordination. The distributed OS, then, is an abstraction provided through the cooperation between, and interaction of, the component OS instances.

Centralised management of user interaction with web services would theoretically be possible. However the huge number of potential service clients and providers, resulting from the geographic spread of the Web and the uptake levels that this spread has encouraged, make it impractical. Such control could be distributed by, for example, creating regional manager entities that work together to provide the abstraction of a single global manager. Any such decomposition would still create significant potential for points of failure and bottleneck.

The alternative approach presented in this paper extends the notion of decomposition of control to the level at which each participating host computer becomes involved in transaction coordination. The key feature of this design is that hosts are only involved on a *need to know* basis. Additionally, the scope of a host's control changes dynamically through the life of the system, as users complete (commit) or abort transactions. Such architecture is possible through the incorporation of a novel transaction representation and management technique, based on Directed Dependency Graphs (DDGs) (Jalili and Henskens, 1995) at each participating host. This technique is described in the following section

2.3 Directed Dependency Graphs

The use of DDGs to control transactions stems from their use in incremental checkpointing of persistent stores (Jalili, 1995). This approach to stability of stores (Brown, 1989) allowed parts of the store to be checkpointed in parallel with user access to other parts of the store. It is important at this stage to note that, while the terms checkpoint and rollback are commonly known as database transaction terms, their use at this time describes the acts of rendering data stable (durable) or reverting the data to a previous stable state respectively.

The DDG technique uses directed graphs to record the inter-relationships created between programs and the objects they access as the programs execute. These relationships are used when mutated object data is written to non-volatile storage, ensuring that other dependent data is also made stable in an atomic operation. The result is that, on restart after unexpected shutdown (crash) of all or part of the system, the recovered state ise physically and logically self-consistent in spite of the inevitable loss of some data.

The critical observation of the DDG work was that a bi-directional relationship exists between processes and the objects they access. In other words, that a connection between a process and an object has different meaning for checkpoint than it has for rollback. Moreover, the relationship between a process and a previously-mutated (by some other process) object it has read is different from the relationship between a process and an object it has itself mutated. These differences can be represented by including a direction component to each graph edge.

For example, consider processes P_1 and P_2 whose activities are linked through a common object O. If P_1 mutates O, after which O is read by P_2, then according to non DDG-based schemes (which typically use set notation to describe inter-entity relationships) all three entities would checkpoint or roll back as a unit (and all other objects associated with each process would also be affected). In fact P_1 and O could checkpoint independently of P_2, and P_2 could roll back independently of P_1 and O. Only a checkpoint of P_2 must propagate to P_1 and O.

This may be recorded using graph rather than set notation using the directed edges $\rightarrow$ and $\leftrightarrow$. When a process P mutates an object O, the edge $P \leftrightarrow O$ is added (if it does not already exist) to the DDG(s) including P and O. When a process P reads a modified object O, the edge $P \rightarrow O$ is added (if it does not already exist or if an $\leftrightarrow$ edge does not exist) to the DDG(s) including P and O. As implied, when a process belonging to a DDG reads a modified object or modifies an object that belongs to another DDG, the two DDGs are merged using one of the described edges to create a single larger graph. In a distributed system each host maintains that part of the graph containing entities located on

that host. Special 'dummy' graph nodes are used to represent links to continuation of graphs on other hosts, allowing graphs to span networks of hosts.

A DDG shrinks when a set of dependent entities is checkpointed or reverts to its last stable state (rolls back). Once a checkpoint or rollback operation is initiated for an entity *E*, the operation propagates to each entity that is reachable from *E* in the DDG to which *E* belongs, if necessary involving network messages between OS instances. Then, because each involved entity is now stable, all edges attached to them are removed.

At any instant each entity belongs to one and only one dependency graph. To find the set of entities dependent on any entity, it is sufficient to find the location of the entity in its graph and then, subject to the kind of operation, traverse the directed graph starting from the entity. Thus the set of dependent entities may differ for entities in the same DDG.

The following section shows how DDGs can be used to provide browser-based transaction control.

3 DDGS, BROWSERS, AND WEB SERVICES

What is required is a new environment that supports the execution of applications constructed to include and make use of the plethora of heterogeneous services available on the Internet (web services). In essence such applications would be constructed using the component-based approach (Brown, 1996). Accordingly, using the new environment, extant web-based entities providing different services and functionalities may be combined to create a single, all-encompassing, coordinated resource. The required environment is provided by a next-generation web browser (a "super browser") that not only supports current methods of Internet use, but also provides a consistent interface for applications to interact with disparate Internet resources. Importantly, the wrapping applications execute without change on any computer architecture/operating system platforms for which the super browser has been implemented.

Purposes of the super browser are to:

- Support all functionality provided by current web browsers.
- Provide a run-time environment for application programs written to utilise services provided by disparate Internet providers. In this function the super-browser behaves in much the same way as the Java Virtual Machine (Lindholm and Yellin, 1999) does for Java programs, providing a consistent interface to underlying services for application programs, and transparently interacting with those services in the form required by each of them.
- Incorporate mechanisms with which application programs can implement transactions, and hence support predictable concurrent use of the underlying services that form components of the applications.
- Allow application programs to execute without alteration on any host computer for which a version of the super browser is available.

The super browser addresses the problems of:

- Coordinating interactions between multiple Internet users and the services to which the Internet provides access.
- Providing a generic interface for execution of component-based web application software so that the software can be used on any supported computing platform.
- Defining a standard interface to Internet services. Services that comply with that interface will be candidates for inclusion, as components, in the more complex applications that execute as clients of the super browser.

Conventional coordination of interactions between active entities (programs) and the data they access has been achieved using centralised control, for example a DBMS or ORB. In the case of interaction between users and Web-based services, such centralised control is not available. The super browser produces a federation of control agents created by enhancing the capabilities of each of the web browser instances being used to access the services. Each enhanced web browser (super browser) instance will host (act as a virtual machine for) zero or more user-controlled application programs. These programs may be locally stored on host computers and invoke the super browser when they execute (similarly to the way HTML files cause a browser to be activated when they are run), or they may be dynamically downloaded from the Internet (using a hyperlink) as required.

3.1 Role of DDGs

As the hosted application(s) access services on the Internet, each hosting super browser instance uses a locally-maintained DDG to store the inter-entity relationships created by the application's activities.

Super browsers become associated with other remote super browser entities on a 'need to know' basis through their access to common service providers. The associations between browser instances are represented in the sub-graph stored at each instance, with 'dummy' graph nodes used to represent between-browser-instance connections. The result is a globally distributed graph structure comprising one or more disjoint graphs with parts of those graphs being stored on one or more hosts.

While the super browser entities provide the equivalent of a single global access coordinator, they actually dynamically form separate clusters based on the patterns of accesses performed by the user programs they host. These clusters grow as services are accessed, and shrink as transactions complete or abort. This architecture produces a scalable and efficient solution to the problem of coordinating interactions between global users and the huge number of available web services, but does not support the conventional models of transaction control.

3.2 DDGs and Transactions

As described in (Henskens and Ashton, 2007), there are similarities in the information recorded in stability DDGs and the information required for transaction management and concurrency control. However, there are also differences between stability and concurrency control requirements, namely:

- The DDG stability technique maintains dependency information on a per-process basis rather than a per-transaction basis,
- The DDG stability technique records information about dirty-read and write accesses, whereas transaction isolation also requires knowledge of clean-read accesses, and
- Stability checkpoints and transaction commits have different semantics.

A transaction is an abstract concept that includes the user-defined boundaries (BEGIN_TRX and COMMIT_TRX), the required data resources and accesses (that may include mutation) to that data. The super browser entities initiate the activities specified by the transactions they host. This occurs every time a hosting browser makes a request of, and receives a response from, a web service involved in a hosted transaction. Thus, on each communication with a web service, it is necessary for the host browser to record the browser-service dependency, including the identification of the transaction that used the service.

Stability mechanisms provide the abstractions: a durable computational store; a logically-consistent store restart state at all times; concurrency control at process level. Full transaction support requires the abstractions provided by the stability mechanism to be augmented as follows:

1. Support for transaction-based events associated with the programming language key words (e.g. BEGIN-TRX and COMMIT-TRX) used to define the extent of each transaction.
2. The transaction extent defines an atomic unit of work that is isolated from any other concurrent activity.
3. The means for managing concurrency should be flexible enough to cope with run-time determination of the temporal extent and physical granularity of interaction.

A consequence of these requirements is that the transaction management system must have control over the timing of checkpoints that correspond to transaction commits.

The DDG-based transaction manager creates edges between graph nodes representing transactions and accessed entities as follows:

1. A clean-read edge is recorded as "—". T — E indicates that transaction T has queried a web service entity E.
2. A dirty-read edge is recorded as "→". $T \rightarrow E$ records that transaction T has read a web service entity E that had been previously mutated since its most recent checkpoint.
3. A write edge is recorded as "↔". $T \leftrightarrow E$ indicates that transaction T has modified web service entity E since it was last check-pointed.

These edges can be used, with appropriate logic at the time of edge insertion (the completion of a browser request-response) or COMMIT/ABORT-TRX events, to implement an optimistic transaction control mechanism. Moreover, the technique has been shown to perform as well as the better of conventional pessimistic or optimistic transaction management over a wide range of transaction sizes, levels of concurrent activity, distribution of involved objects and object store sizes (Ashton, 2004, Henskens and Ashton, 2007).

Transactions are widely accepted as an appropriate mechanism for management of control over concurrent access to objects in a store (Date, 1999). Thus the use of DDG-based concurrency Control (DCC) represents an excellent choice for general-purpose transaction management and concurrency control.

The super browser uses the Directed graph-based Concurrency Control (DCC) transaction technique to

manage the interactions of component-based applications with web services, thus providing ACID-compliant interaction between concurrent users and the service providers.

Implementation of DCC at browser level requires communication between super browser entities. The current generation of browser entities only communicate with service/page providers and their host computer systems, so browser-to-browser interaction represents a novel enhancement of the way browsers operate.

3.3 Inter Browser Communication

Super browser entities must communicate in order that commit, abort and roll-back actions can be implemented. The latter two actions occur under user program instruction, and the former as a result of transaction manager determination that the transaction cannot succeed because of concurrent activity involving the web services (Ashton, 2004). A browser that is either instructed to commit or abort, or that determines a need to roll back, uses its locally stored graph segment to communicate with involved web service providers instructing them to take appropriate action. It uses the 'dummy' graph nodes that provide connection to remote browser entities to traverse the graph to those entities, causing the operation to propagate through the distributed system.

Many browser entities are hosted on computers that sit behind routers or firewalls, for example as parts of local area networks connected to the Internet through a broadband connection such as ADSL or cable. Such browsers are able to operate successfully in the usual request-response circumstance using techniques such as NAPT. Servers sitting behind routers are only accessible (except in a response situation) through definition of a port forward defined in the router (Comer, 2004).

At the commencement of a transaction, the initial relationship between the controlling browser and the involved web services is of the client/server nature supported by techniques such as NAPT. Implementation of DCC-based concurrency communication changes the relationship between the participating browsers and service providers to peer-to-peer, requiring initiation of requests from the Internet side of routers to entities behind those routers, and without port forwarding implemented in those routers. This is made possible by storing of IP and port information for browser entities, obtained from the initial request messages, in the graph 'dummy' node(s) representing those entities in the DDGs. A comprehensive description of this aspect of DCC-based transaction management will be presented in a future publication.

3.4 Application Program Interface

The super browser provides a run-time environment for client application programs. Initially this takes the form of extension of the Java Virtual Machine (JVM) that, in conventional browsers, supports execution of Java applets. Extensions to the Java Application Programmer Interface (API) provides for bracketing of transactions using the usual notations (BEGIN_TRX, COMMIT_TRX and ABORT_TRX). Additionally, the API supports identification of the web services that form components of the application, and subsequent access to those services.

Future work will investigate support for fully compiled client software written, for example, in object-oriented languages like C++. For such programs the super browser will appear to be more like an extension of the operating system (in the same way as is middleware) than as a user-level application program.

4 CONCLUSION

Web services allow business to business interaction across the Internet, and are increasingly being used for web page based interaction between businesses and their clients.

It is currently difficult or impossible to construct applications, or web pages, that provide transaction-like semantics to use of web services offered by unrelated service providers. This is particularly the case if there is a requirement for human interaction with the transaction.

Directed dependency graphs, previously used to underpin stability and later transaction-based concurrency control in persistent object stores, can be used to provide transactions involving clients and disparate web services. Termed DCC, this technique, when implemented in a federation of enhanced web browsers, provides an efficient, distributed and scalable form of transaction management.

With the support provided by these so-called super browsers, programmers can build component-based client application programs incorporating multiple remotely provided web services. Clients can enjoy transaction semantics in their use of those services. Moreover, the application software can either execute using a virtual machine (e.g. JVM) provided by the browser, or as an independent

program for which the super browser behaves as middleware with respect utilisation of web services.

Features of this new approach to control of interaction with web services include:

- Extension of the extant web service interaction models to include support for human interaction.
- A software platform (the super browser) providing support for programming and execution of applications that implement the extended web service interaction model.
- An Application Programmer Interface (API) for the interaction between web services and the super browser.
- An API for the interaction between the component-based application programs and the super browser.
- A browser-to-browser communication protocol that supports transactions and transaction-based concurrency control.

A super browser implementation, together with sample user applications and web services are currently under development, and will be used to prove and demonstrate these technologies. The results will form the subject of a future paper.

REFERENCES

Adobe Systems Incorporated, 2007. Macromedia Flash. http://www.adobe.com/products/flash/flashpro/.

Apple Inc., 2007. Safari RSS. http://www.apple.com/macosx/features/safari/.

Ashton, M. G., 2004. Management of Data, Access and Concurrency in a Persistent Object Store. *Ph.D, Computer Science & Software Engineering.* University of Newcastle.

Birrel, A. D. & Nelson, B. J., 1984. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems,* 2(1)**,** 39-59.

Bray, T., Paoli, J., Sperberg-Mcqueen, C. M., Maler, E. & Yergeau, F., 2006. Extensible Markup Language (XML) 1.0 (Fourth Edition). http://www.w3.org/TR/2006/REC-xml-20060816.

Brown, A. L., 1989. Persistent Object Stores. *Faculty of Mathematics and Computational Science.* St Andrews, Ph.D, Universities Of St Andrews and Glasgow.

Brown, A. W., 1996. *Component-Based Software Engineering,* Wiley.

Christensen, E., Curbera, F., Meredith, G. & Weerawarana, S., 2001. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

Comer, D., 2004. *Computer Networks and Internets,* Pearson/Prentice Hall.

Date, C. J., 1999. *An Introduction to Database Systems,* Reading, MA, USA, Addison-Wesley Publishing Co.

Emmerich, W., 2000. *Engineering Distributed Objects,* Wiley.

Farrag, A. A., Ozsu, M. T., 1989. Using Semantic Knowledge of Transactions to Increase Concurrency. *ACM Transactions on Database Systems,* 14**,** 503 - 525.

Gray, J. & Reuter, A., 1993. *Transaction Processing: Concepts and Techniques,* San Mateo, CA, Morgan Kauffmann Publishers.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J. & Nielsen, H. F., 2003. Simple Object Access Protocol Version 1.2. http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

Hanson, J., Nandi, P. & Levine, D., 2002. Conversation-enabled Web services for agents and e-business. *International Conference on Internet Computing.*

Härder, T. & Reuter, A., 1983. Principles of Transaction-oriented Database Recovery. *ACM Computing Surveys,* 15(4)**,** 287 - 317.

Henskens, F. A. & Ashton, M. G., 2007. Graph-based Optimistic Transaction Management. *Journal of Object Technology*.

Ibm & Sap Ag, 2005. WS-BPEL Extension for People.

Jalili, R., 1995. A Failure Transparent Distributed Persistent Store. *Ph.D, Basser Department of Computer Science.* Sydney, University of Sydney.

Jalili, R. & Henskens, F. A., 1995. Reducing the Extent of Cascadable Operations in Stable Distributed Stores. *18th Australian Computer Science Conference*. Adelaide, Australia.

Juric, M. B., Mathew, B. & Sarang, P., 2006. Business Process Execution Language for Web Services Version 2nd Edition. PACKT Publishing.

Kung, H. T. & Robinson, J. T., 1981. On Optimistic Methods for Concurrency Control. *ACM Transactions on Database Systems,* 6(2)**,** 213-226.

Lazovik, A., Aiello, M. & Papazoglou, M. P., 2003. Planning and Monitoring the Execution of Web Service Requests. *Service-Oriented Computing - ICSOC 2003.* Springer-Verlag, Lecture Notes in Computer Science.

Lindholm, T. & Yellin, F., 1999. *The Java Virtual Machine Specification,* Sun Microsystems.

Microsoft Corporation, 2007a. Microsoft Outlook. http://office.microsoft.com/en-au/outlook/default.aspx.

Microsoft Corporation, 2007b. Windows Internet Explorer. http://www.microsoft.com/windows/ie/default.mspx.

Momin, K. A. & Vidyasankar, K., 2000. Flexible integration of optimistic and pessimistic concurrency control in mobile environments. *Lecture Notes in Computer Science,* 1884**,** 346-353.

Mozilla Foundation, 2006. About Mozilla. http://www.mozilla.org/about/.

Nelson, T. H., 1965. A File Structure for the Complex, the Changing and the Intermediate. *20th National Conference.* ACM.

Oasis Udi, 2004. Introduction to UDDI: Important Features and Functional Concepts. http://www.uddi.org/whitepapers.html.

Object Management Group, 1998. *The Common Object Request Broker: Architecture and Specification*, 492 Old Connecticut Path, Framinkham, MA 01701, U.S.A.

Opera Software Asa, 2007. The Opera Web Browser. http://www.opera.com/.

Peinl, P., Reuter, A., 1983. Empirical Comparison of Database Concurrency Control Schemes. *9th International Conference on Very Large Databases.* Florence, Italy, Morgan Kaufmann.

Qualcomm Incorporated, 2007. Eudora. http://www.eudora.com/.

Raggett, D., Le Hors, A. & Jacobs, I., 1999. HTML 4.01 Specification. http://www.w3.org/TR/html401/.

Siegel, J., 1996. *CORBA Programming*, Wiley.

Sinha, P. K., 1996. *Distributed Operating Systems: Concepts and Design*, Wiley-IEEE Press.

Sun Microsystems Inc., 2007. Java Plug-In Technology. http://java.sun.com/products/plugin/.