WEB BROWSER TRANSACTIONALITY

Mark Wallis, David Paul, Frans Henskens and Michael Hannaford
Distributed Computing Research Group
School of Electrical Engineering and Computer Science
University of Newcastle, Callaghan, NSW, Australia

Keywords: Web Browser, Transaction Management.

Abstract: As the complexity of web applications increases new challenges are faced in relation to data integrity and system scalability. Traditional client/server fat applications allow for a high level of transactionality between the client and server, due largely to transactional protocols and tight coupling between components. Transactional functionality within web applications is historically limited to within the web server hosting the application. The scope of the traditional transaction in this context does not extend outside of the web server and its attached services. This paper proposes that web applications can achieve increased system integrity by extending the scope of the transaction to encompass tasks performed by the web browser. An additional layer is introduced to the standard HTTP protocol to facilitate the new functionality, and a simulator is presented as the basis for further research.

1 INTRODUCTION

In Web 1.0 systems, the web browser's role is that of a thin-client (O'Reilly, 2005). As the web evolved to Web 2.0 (McCormack, 2002) and beyond, the role of the web browser has become increasingly complicated; for example, it is now quite common to find application code executing within the web browser (Flanagan, 2002). This increased complexity has allowed engineers to add active content functionality to the web platform. Client-side web application code can dynamically update interfaces with information without having to rely on the classic request/response round trip as was utilised in Web 1.0. This application code can interact with the web browser environment and also with web application components, supported by technology such as AJAX (Garrett, 2005). The tight coupling of the code which executes within the web server and web browser leads to a risk of data corruption due to inherent instability of the communication link between the server and browser. Traditionally, the way to deal with such issues was to implement concepts such as transactionality (Gray and Reuter, 1993).

'Transactions' in this sense relate to bracketing tasks performed by the web application and any utilised services to form atomic units. To date, the focus has been placed on tasks being performed on the server side with the client-side tasks being neglected. The HTTP protocol used to connect the server and client sides of the web application does not have inherent transaction support. In contrast, fat applications that execute in other distributed models, such as CORBA-based systems (Object Management Group, 2004), have transaction support built into their core design.

This paper describes the investigation of extensions to the web paradigm that allow code executing within client browsers to be involved in transactions defined by the web application code at the server side. These enhanced web applications are built around a component-based software architecture where the web server is able to push code out to the web browser for execution. These two pieces of independently executing code essentially give a base architecture where application components within the web server communicate with application components within the web browser.

An extension to the HTTP protocol is presented, along with the design of various software components

Wallis M., Paul D., Henskens F. and Hannaford M.
WEB BROWSER TRANSACTIONALITY.
DOI: 10.5220/0001824300930100
In *Proceedings of the Fifth International Conference on Web Information Systems and Technologies (WEBIST 2009)*, page
ISBN: 978-989-8111-81-4

required to fulfil end-to-end transactionality. Current models for transactions concentrate on server side functionality. This research essentially duplicates the same behaviour to the client side of the system and will eventually allow complete transaction management to occur within the web browser. This will allow tasks to be bracketed into atomic actions across multiple disparate web applications and web services.

Section 2 of this paper introduces a specific problem in the context of web application design. Section 3 then draws parallels between the issues found on the client side of the web application to those identified in previous research with placing web services in transactions. A high level design is detailed and reviewed in section 4 before a worked example is presented in section 5. As a proof-of-concept, a prototype and simulation package has been developed and various results have been collected. These are presented in sections 6, 7 and 8.

2 PROBLEM DESCRIPTION

A classic example of the problem addressed by this paper can be seen in web applications that aggregate multiple web services. Take, for instance, a web application that can book the various components of a holiday for the user. Such a web application implements multiple stages. The first stage books the flights using an aggregated web service contract with various airlines. The second stage allows the user to book their accommodation and a third stage allows them to book a hire car. Such applications provide the benefit of a 'one-stop-shop' and save the user from having to separately locate and initiate interaction with multiple web applications to book the various elements of their holiday. These aggregator applications are typically available only to commercial entities such as travel agents, and not to the general public. Their use involves user interaction at the various workflow stages, for example, to allow selection of service provider, level of service and service options. These aggregator systems create what is essentially a single transaction made up of various atomic actions that are implemented either directly by the aggregator application or through the aggregator application by contracted web services.

Various issues can arise from including user interaction in the transaction. For instance, the user may reserve a seat on a flight but then fail to complete the transaction, essentially leaving some resources locked. These resources remain locked for a period of time specified by the agreed commit timeout value. Such issues are exacerbated by high levels of concurrent user activity involving the component services. For example, the user who leaves an incomplete workflow may have reserved the last seat on an otherwise fully-booked flight. During the commit timeout period, on the basis of the seat's unavailability, other travellers may have booked their flight with a competitor to the detriment of the provider with the reserved seat.

Systems implementing transactions that include user involvement tend to operate with a hard time limit on the overall interaction. This ensures that resources are not locked indefinitely. A hard time-limit of, say, 20 minutes may leave resources locked for far longer than required, especially if the user has left the session only a few minutes into the interaction and does not plan on returning.

This research addresses issues such as these by providing an additional level of interaction between the web browsers and web server. This interaction involves the web browser itself in the management of the transaction.

3 WEB APPLICATION TRANSACTIONALITY

One of the main problems with transactions in the web environment is that the web is inherently unreliable. Servers or clients can be shut down or network connections broken at any stage and, because each web system is autonomous, no other party can control what happens when the system comes back online. In order to overcome this problem, standards which remove some of the autonomy of the hosts have been defined. These standards specify what should happen in the event of a communication breakdown, so all parties can agree on the expected state of the overall system even when some messages fail. For example, standards such as WS-Coordination (Cabrera et al., 2005) and WS-Transaction (Cox et al., 2004) require participants in a transaction to pass the decision of whether to perform an action to a transaction coordinator, and to guarantee that they will adhere to that decision.

Currently, having the user close a web browser (or tab) while performing a task such as filling in a multi-page form at best leaves the task in an incomplete state, and at worst completes the task with incomplete data. If the client's browser was involved in the management of a transaction that ensured that all form data was completed then, when the user closed the browser, the browser would inform the web application to roll back the transaction. Of course, if an unexpected event such as a network disconnection

or power outage occurred, the browser would not get a chance to send such a message. However, the fact that the client's browser was included as a participant in the transaction management would mean that, after a specified time-out period, the transaction would roll back so it would appear as if the client never started the task.

Allowing the client's web browser to participate in the management of a transaction has the roll-on effect of also involving the users themselves. Interactions between the client and the web browser are inherently unreliable and unpredictable and these problems need to be addressed within the system design. Traditionally, components in a component-based system are able to rely on the middleware to provide a level of guaranteed message deliverly. On the web, there is no such guarantee and, as such, additional measures need to be put in place to ensure end-to-end stability.

Additionally, if future support is built into the browser, it will be possible to allow a client to leave a task before completion and later rejoin the transaction from the point where they left. This would be especially useful for the unexpected client disconnections as, when the network link was re-established or system rebooted, the client's browser could restart and rejoin the transaction. While it would be possible to perform a similar action for simple tasks using technologies such as cookies (Kristol, 2001), the server would see this as a new connection rather than the continuation of an old workflow.

While transactions existing over multiple HTTP request/responses are not a new concept, the definition of a component to manage these transactions at the client side allows for greater end-to-end stability of the system. Various other implementations that use technologies such as Java Applets to heartbeat back to the web application rely on these applets being able to execute a "dying breath" function call to inform the web application that the browser has closed.

4 THE TRANSACTION LAYER

Figure 1 shows a new system design that supports browser-based transactional support. To involve the web browser in a transaction there must exist a channel between the browser and application which can be used for management of the transaction. Initially, when the web application creates a new transaction that involves the web browser, it must notify the web browser of the transaction ID. The standard way for a web application to provide such meta-information to the browser is through the use of a HTTP header name/value pair. As such, a new HTTP header named "WebTransactionID" has been defined. This transaction ID value is additional to any session ID value that the application may already share with the web browser. This allows multiple transactions to be processed within a single user session. When the web browser receives this header it passes the value to a browser component responsible for transactional management within the web browser environment. This object is called the Web Browser Transaction Manager (WBTM).

The WBTM is a component within the web browser environment, similar to components that manage tasks such as HTML rendering and cookie storage. The WBTM's role is to manage the tasks performed by the web browser while within a transaction. Its primary responsibility is to report back to the web application if the transaction needs to be rolled back. To report this, an out-of-band (OOB) connection is established between the web browser and the web application upon creation of a new transaction. The OOB connection is used by the browser to execute a web service to request a roll back of the transaction. All information passed between the WBTM and the web application is tagged with the WebTransactionID provided during the initial HTTP response. This channel is used if the user decides to close the web browser window while the transaction is still active. A heartbeat occurs over this connection to allow the web application to time-out a disconnected web browser. This communication has to occur out-of-band as there is no guarantee that there will be a HTTP request/response pair between the browser and web application at regular enough intervals to not introduce additional delays in the web application knowing the state of the various transactions it is managing.

The final commit of a successful transaction occurs within the web server at which the transaction originated. Once the web application has declared the transaction complete, the final HTTP response is used to inform the WBTM that the transaction is complete. The WBTM then disconnects the OOB connection if no other transactions exist for that web application. If multiple transactions are being performed, for example, in separate web browser tabs, then the OOB channel stays open until all transactions have either been completed or rolled back. Only a single OOB channel is required per web application for each web browser instance.

When an OOB channel closure occurs the web application has the choice to either roll back the transaction or keep it alive for a pre-determined time period. Keeping the transaction alive would allow the user to rejoin the transaction by having the WBTM

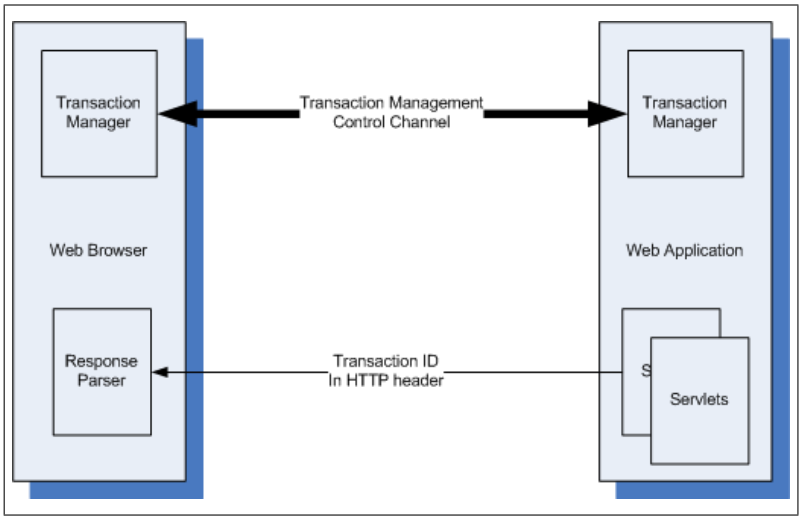

Figure 1: Proposed System Design.

re-establish the OOB connection. This would be useful for instances where the OOB channel breakdown was caused by network segmentation.

The capabilities of the WBTM component can be extended past the realm of sub-management of transactions owned by single web applications to complete management of transactions that involve multiple web applications. Previous research has presented this concept in the realm of a Super Browser (Henskens, 2007) implementation.

5 WORKED EXAMPLE

The sequence diagram shown in Figure 2 gives an example in which multiple HTTP requests occur and the transaction is successfully committed. The WebTransactionID is provided in the initial HTTP response as an HTTP header and is then passed with each subsequent HTTP request/response belonging to that transaction. Once the transaction is committed the HTTP header in the final response is appended with a flag that identifies the transaction as complete. This allows the WBTM to free any resources being used to manage the transaction, such as the OOB channel.

In a situation where the user closes the web browser in the middle of the transaction, the WBTM receives a notification from the web browser environment informing it that the browser page has been closed. It then uses the OOB channel to notify the transaction manager within the web application so it can then roll back the transaction.

Analysis of existing client/server interactions have identified the risks associated with extending a transaction over multiple HTTP request/response calls. These risks include:

1. The user closing their browsing tab during the transaction.
2. The user closing their browser during the transaction.
3. The user losing network connectivity to the server during the transaction.
4. The user's machine crashing during the transaction.

While various techniques exist to mitigate some of these risks, the proposed system allows a centralised method for addressing all four issues. If the user closes the browser tab mid-transaction then the WBTM is made aware of the fact by monitoring events within the web browser. It then uses the OOB connection to roll back the transaction. A deconstructor in the WBTM caters for the user closing the web browser completely during a transaction. The WBTM is then able to notify any web applications that currently have active transactions that they need to be rolled back. If the user loses network connectivity with the web application server during a transaction then the heartbeat running over the OOB connection will expire and the web application waits until a time-out before rolling back the transaction. The same occurs if the user's machine crashes and the OOB connection is not cleanly terminated.

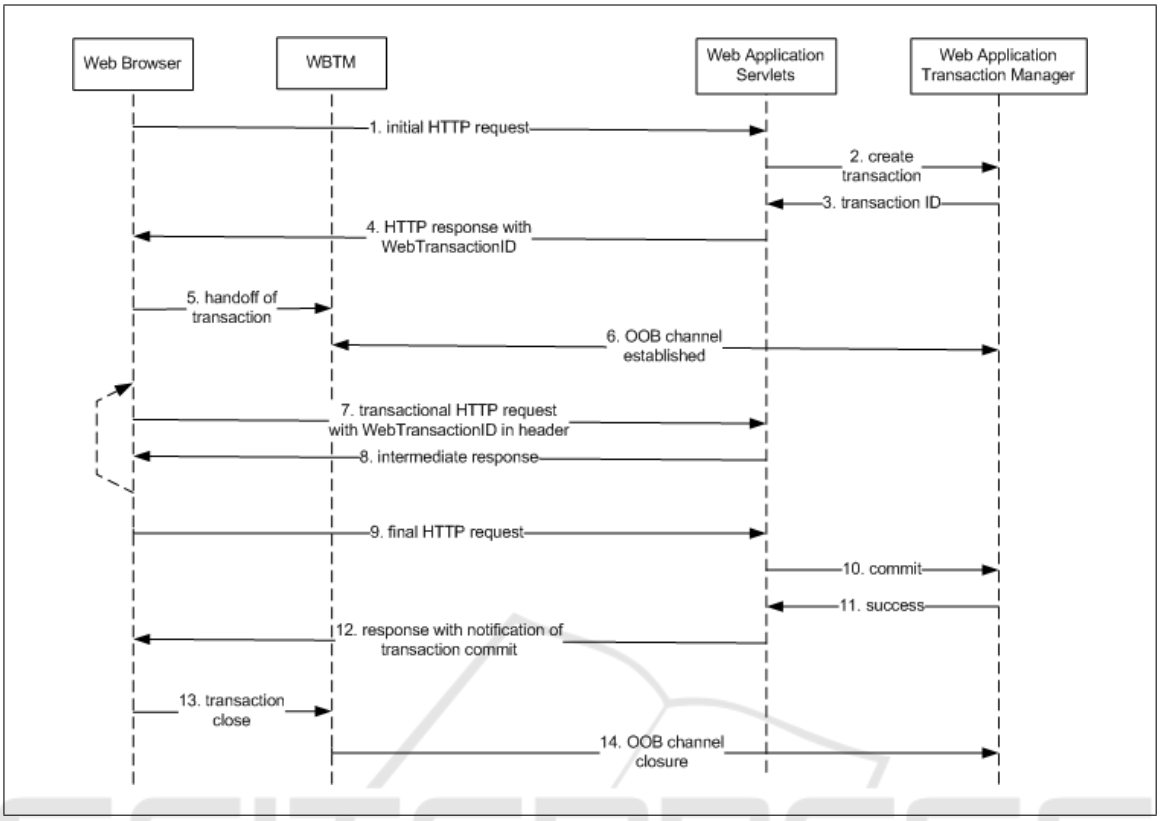

Figure 2: Worked Example Sequence Diagram.

6 PROTOTYPE

A prototype has been developed to demonstrate involvement of web browsers in transaction management. Based upon Java (Gosling et al., 2005) and the Tomcat (Apache Software Foundation, 2008a) servlet engine, the development of the prototype consisted of two main tasks: enhancements to the web server, and enhancements to the web browser.

The software enhancements at the web server involve an additional layer in the web servlet engine that allows the web developer to implement a 'transactional servlet'. The prototype implements this transactional support using an additional servlet class inheritance layer in the Tomcat web application server. The additional layer takes care of processing the web developer's request to begin and commit a transaction, as well as the required protocol enhancements to pass the transaction ID to the web browser using HTTP headers as described in section 4. Additionally, a new web service is defined and exposed from the web server. This service manages the OOB connection for each client transaction. The prototype uses the AXIS SOAP (Apache Software Foundation, 2008b) library to implement a basic web service that supports two commands - heartbeat() and rollback().

The web browser component of the prototype was implemented as a software extension to the Firefox (Mozilla, 2008) web browser. This extension has three major functions:

1. Monitor all HTTP responses and identify any transaction-related HTTP headers.
2. Implement the transaction manager within the web browser.
3. Manage the OOB communication channel between the web browser and web application.

The purpose of the prototype is to experiment with and fine tune the various methods and associated event bindings required by the proposed system enhancements. The following messages were identified and fed into a custom developed simulation package to gather the comparative performance metrics described in section 7.

- Message 1: The begin transaction message from server to browser.
- Message 2: The heartbeat message from browser to server.

- Message 3: The complete transaction message from server to browser.
- Message 4: The rollback transaction message from server to browser.
- Message 5: The rollback transaction message from browser to server.

Additionally, the following new system events were identified:

- Event 1: Server-generated web browser heartbeat timeout.
- Event 2: Server-generated transaction commit timeout.

7 SIMULATION

Using the messages and events identified during the prototyping stage a simulation program was developed to collect metrics on the of the new browser-based transaction support. The simulation program is a combination of custom web server components and Grinder (Grinder Open Source Project, 2008). The Grinder software package was used to simulate the behaviour of the web browser sessions of multiple users. The web server hosted two servlets for simulation, an application servlet and the OOB web service. The Grinder software package was configured with various user profiles and simulated user activity by sending pre-defined HTTP requests to the web server servlets, based on the web browser activity defined in section 6. Every second the web server collected statistical information on how many resources had been locked by the simulated web browsers.

Since the analysis in this paper is concerned with the contention of locked resources, it suffices to restrict requests in this prototype to a single type of resource rather than the heterogeneous resources required for a holiday booking application. In effect, this places all resources in a single pool instead of having a separate pool for each of the different resource types. The system behaves identically with a single pool as it would with multiple pools, though the differences between the various resource types have been abstracted over.

A random set of user behaviours was defined over a time period of 10 minutes. The random behaviours were generated according to the distribution:

- 80% completed the transaction within 60-180 seconds
- 10% exited the transaction ungracefully after 10-100 seconds of activity
- 10% timed out from the transaction after 10-100 seconds of activity

A feature of the simulator is that identical user behaviour data can be used for each scenario: the system flow without web browser transaction support and the flow where the system implemented the transaction support enhancements. In the absence of transactional support a commit timeout of 5 minutes was used. For the other scenario, a heartbeat interval of 30 seconds was used. Statistical resource usage information was then analysed to define trends and investigate the effect of the new browser-based support for transactions.

8 RESULTS

The first analysis involved the level of resource locking that occurred at the web server. Figure 3 depicts the difference in the number of resources locked in a system system and a system that supports the browser-based transaction extensions. This shows that the level of resource locking is decreased when using the new browser-based transaction system, which leads to greater availability of the server's resources. Thus, the experiment demonstrates the benefit of involving the web browser in the management of transactions.

The simulation results were also analysed from the the point of view of user perception of resource availability. Figure 4 depicts the users' perception of resource availability with the standard system versus the browser-based transactional model. Users in this simulation were attempting to obtain between 1 and 4 resources from a pool of 80 available resources. If resources were not available then the client's request failed. Clients, once rejected, sought the resource elsewhere. Figure 4 also depicts the number of un-booked resources (as opposed to reserved resources) over time. It was found that, for the standard model, clients believed there were no more resources available at time 289, when in fact there were 33 resources reserved but not yet firmly booked. These un-booked resources did not become available until well after the users had left. Under the browser based transactional model this value was reduced to 9 un-booked resources at time 449. In addition, with the browser-based transactional model, those 9 resources were then fully booked within the next 64 seconds giving full resource utilisation due to the fast rollback times the new model provides. These results show a 41% increase in the number of resources booked and demonstrates improved system utility from the viewpoint of both resource providers, who dispense

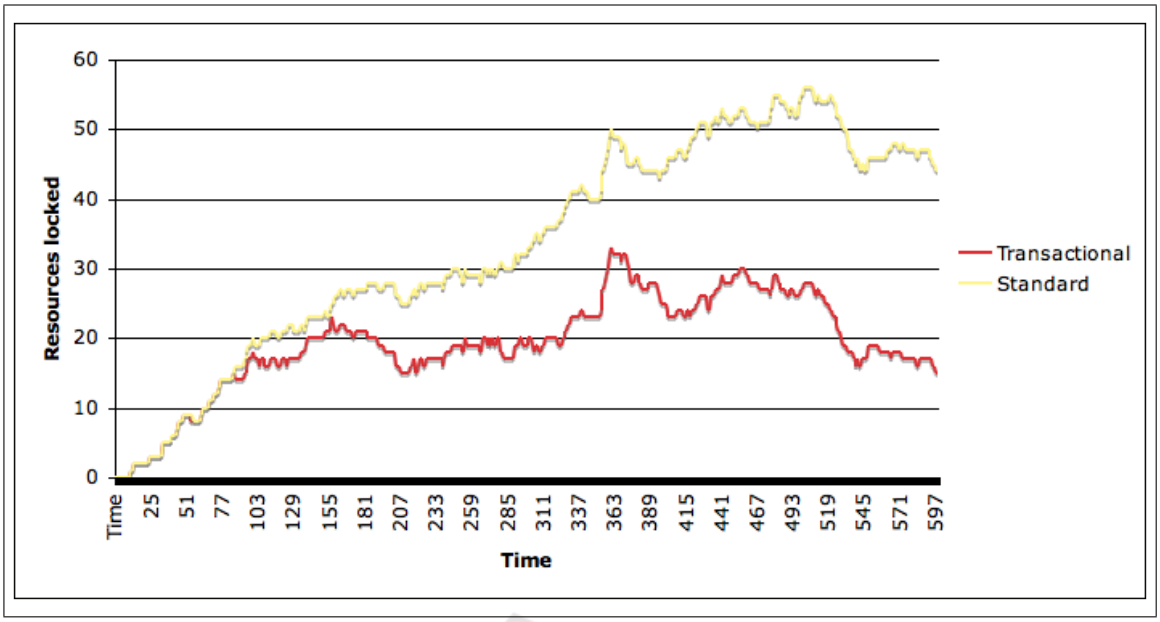

Figure 3: Resources locked with and without transaction support.

a higher percentage of product, and resource consumers, for whom more of the product is ultimately available.

9 CONCLUSIONS

This paper presents research extending management of distributed transactions to include participation of web browsers. This extension provides greater end-to-end stability through reliance of software components executing both within the web application server and the web browser. This concept is extendable to systems that use tools similar to the 'Super Browsers' (Henskens, 2007), which incorporates extended runtime environments for more complex code execution within the web browser. The increased power of the runtime environment within super browsers opens up possibilities of complex code executing at the client side in a completely autonomous manner. The technology presented in this paper supports a different approach to the way in which web transactions are treated by the overall system. This is realised by viewing the system as a component-based architecture with various system components distributed across the web server and web browser.

It is expected that browser involvement will lead to efficient dynamic user interaction with transactions. This will be particularly useful, for example, when a user wishes to aggregate services from multiple service providers in a way that is inherently transactional.

The system presented in this paper allows a web application to push transaction management code to a web browser, where it executes independently in the web browser's runtime environment. The additional responsibility placed on the web browser represents a move towards an increasingly peer-to-peer-based Web architecture, where the role of client and server are blurred. The challenges of moving to a peer-to-peer design are well established (Subramanian and Goodman, 2005) and must be addressed, especially in relation to connectivity between peers (Wallis et al., 2007).

The example demonstrates one situation in which browser-based transaction management provides enhanced system functionality. Ongoing research is investigating novel technologies such as dynamic, client defined, transactions involving autonomous web services (Paul et al., 2008). These technologies, which previously required the existence of dedicated middleware such as a CORBA ORB, are made possible by the underlying research presented in this paper.

The future research will open up new possibilities and address issues such as how to inform users that previously unavailable resources

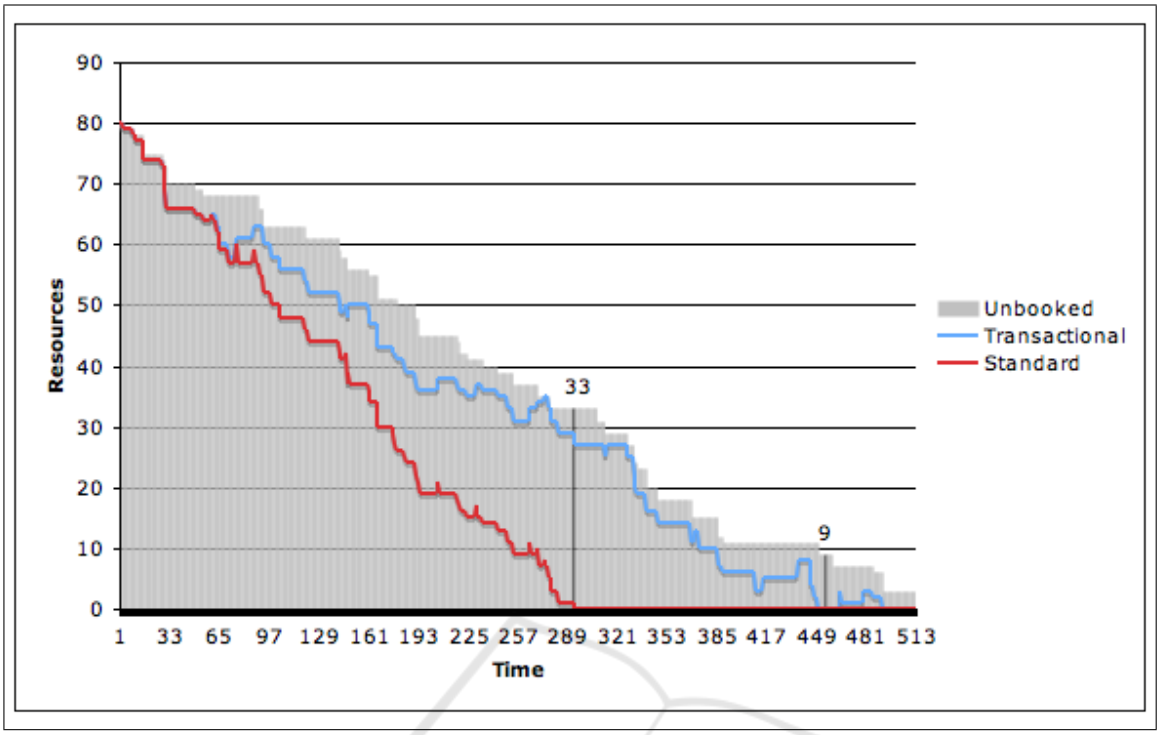

Figure 4: Users competing for unique resources.

REFERENCES

Apache Software Foundation, Apache tomcat. http://tomcat.apache.org.

Apache Software Foundation, Web services - axis. http://ws.apache.org/axis/.

Cabrera, L. F., Copeland, G., Feingold, M., Freund, R. W., Freund, T., Johnson, J., Joyce, S., Kaler, C., Klein, J., Langworthy, D., Little, M., Nadalin, A., Newcomer, E., Orchard, D., Robinson, I., Shewchuk, J., and Storey, T. (2005). Web Services Coordination (WS-Coordination). Technical report, Arjuna Technologies Ltd., BEA Systems Inc, Hitachi Ltd., IBM Corporation, IONA Technologies, Microsoft Corporation.

Cox, W., Cabrera, L. F., Copeland, G., Freund, T., Klein, J., Storey, T., and Thatte, S. (2004). Web Services Transaction (WS-Transaction). Technical report, BEA Systems Inc, International Business Machines Corporation, Microsoft Corporation.

Flanagan, D. (2002). *Javascript: the definitive guide*. O'Reilly.

Garrett, J. J. (2005). Ajax: A new approach to web applications. *Adaptive Path 2005*.

Gosling, J., Joy, B., Steele, G., and Bracha, G. (2005). *Java (TM) Language Specification*. Addison Wesley, 3rd edition.

Gray, J. and Reuter, A. (1993). *Transaction processing : concepts and techniques*. Morgan Kaufmann Publishers, San Mateo, Calif.

Grinder Open Source Project, The grinder. http://grinder.sourceforge.net/.

Henskens, F. (2007). Web service transaction management. *International Conference on Software and Data Technologies (ICSOFT)*.

Kristol, D. (2001). HTTP Cookies: Standards, Privacy, and Politics. *ACM Transactions on Internet Technology*, 1(2):151–198.

McCormack, D. (2002). *Web 2.0: The Resurgence of the Internet and E-Commerce*. Aspatore Books.

Mozilla, Mozilla firefox. http://www.mozilla.com/en-US/firefox/.

Object Management Group (2004). *Common Object Request Broker Architecture: Core Specification*.

O'Reilly, T. (2005). What is web 2.0. *O'Reilly Net*.

Paul, D., Wallis, M., Henskens, F., and Hannaford, M. (2008). Transaction support for interactive web applications. In *4th International Conference on Web Information Systems and Technologies (WEBIST-2008)*, volume 4th of *WEBIST*. INSTICC.

Subramanian, R. and Goodman, B. D. (2005). *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*. IGI Publishing.

Wallis, M., Henskens, F., and Hannaford, M. (2007). A system for robust peer-to-peer communication with dynamic protocol selection. *The 8th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*.