Caching of XML Web Services to Support Disconnected Operation

Justin Reabow
jreabow@cs.uct.ac.za

Darryl Pillay
dpillay@cs.uct.ac.za

Technical Report Number: CS04-09-00
Department of Computer Science
University of Cape Town

ABSTRACT

XML Web services can now be accessed in all places and at all times. The problem now facing these XML Web services is the need be to universal availability. Caching can be used by client applications that use XML Web Services on wireless or mobile networks in the face of intermittent connectivity. The idea of interjecting a client side cache proxy may be a step in the direction towards the ultimate goal of a seamless online/offline operating environment of these XML Web Services. But, Web services present new challenges to existing cache managers since they have generally been designed without regard to caching and hence offer little support. The WSDL description of a Web service specifies the message format of a necessary to invoke a service operation but lacks the information needed to indicate whether an operation will modify the server state or produce different results on different invocations. We have suggested several annotations to the WSDL document that will allow custom cache managers to tailor their behavior based on the specific requirements of the Web service. We then built a caching system onto an HTTP proxy and interjected it between a Web service and its application client, to test our assumptions. We demonstrated that a XML Web services could be operated to a limited extent disconnected from the server, without modifying the implementation of the service or their applications.

Categories and Subject Descriptors

C.2.4 [C Computer Systems Organization C.2 COMPUTER-COMMUNICATION NETWORKS Distributed Systems]: J2SE XML

General Terms

Design Experimentation Reliability

Keywords

XML Web Services. Caching, SOAP Message Fabrication, WSDL Annotations, Local Cache Consistency Availability

1. INTRODUCTION

Web Services are emerging as the dominant application on the Internet. The web is no longer just a vast repository for information but is evolving in an active medium for providers and consumers of services. We are now seeing a wide variety of services being offered on the Internet; peer-to-peer services providing access to such services as access to personal contact information, business services, and a whole host of consumer services such as online bidding (ebay) and online shopping (amazon)[9]. Web services are evolving into services designed for programmatic use rather than human access. Thus, we are seeing XML Web services becoming the building blocks for a new generation of Internet applications. The potential for growth of personal and business services are limitless.

One of the key requirements for the success of these Web services is universal availability. These XML Web services tend to be accessed in many different places and at all times. People are now being found to access these services on a wide range of devices, ranging from desktop computers, laptops, PDAs, a whole host of hand held devices, and now with a new generation of smart phones using wireless networking technology. But, with frequent disconnections and unreliable bandwidth characterizing these networks, the availability of the Web services become of significant concern.

A solution to problem of availability would have to be transparently deployable and generally applicable [9]. The meaning of this is that a system that intends improving the availability of Web services must not require changes to the implementation of the client, or the server, or to the communication protocols between them. Due, to the fact that Web services are growing too rapidly, changes to each individual Web service implementation would be infeasible. So a solution to the problem would have to involve interposing storage and logic that acts transparently in the communication path between the client and server, and would require no changes to the implementation of either the client or the server.

So what is the solution? Caching satisfies both the requirements of being transparently deployable and generally applicable. This project puts forward an architecture for a client side request-response cache that mimics the behavior of our Web Service to a limited extent. Our cache system is transparent to both the client and the server, and hence no changes on either the client or the server were needed. Our system conforms to the standards put forward by the W3C, so theoretically can be applied to any Web service that conforms to these standards.

This paper is organized in the following way. Section 2 introduces Web Services to the uninitiated and section 3 discuses

the background to the project. In section 4 an investigation into caching XML Web Services and the various issues exposed are summarized. The prescribed WSDL annotations and architecture of the enhanced Web Service that was developed is covered in section 5. An evaluation of the system follows in Section 6. Finally, section 7 concludes the paper and discusses future work.

2. WEB SERVICES

Web Services generally consists of a service provider and multiple consumers based on a client-server architecture. Each Web service uses a custom communication protocol for the client to access the servers. The most common access pattern for a Web service consists of requests and responses. This pattern involves the client sending a request message that specifies the operation to be performed and all the required information to the server. The server completes the relevant operations and replies with a response message. This series of activities may result in permanent changes to state of the server.

Web services represent black box functionality, without wondering how the actual service is implemented [1]. Web services provide RPC like services to the client. But, due to the vast diversity of the operations that are exported by these web services, it is difficult to distinguish clear categories of operations. For example, an address book web service may support operations such as insert, delete, replace, and/or query operations to perform database functions. But, one web service may support an interface that is vastly different to that of another. For example, a travel reservation web service may support operations to provide services to search airfares, make reservations, book tickets, etc.

The World Wide Web Consortium (W3C) has recommended a set of standard for web services based on the Extensible Markup Language (XML) [2]. These XML-based standards provide globally recognizable protocols for discovering, describing and accessing the custom interfaces of Web Services [3]. This set of standards compromise of two important standards: SOAP (Simple Object Access Protocol) and WSDL (Web services Description Language).

Web Service Description Language (WSDL) [4] is an XML based standard which defines the interface to a Web Service. It provides all the information required for an application to access the specified service. A Web Service is defined as a collection of ports, which, in turn, are a collection of abstract operations and SOAP messages. Keeping the operations abstract allow them to be bound to different protocols and data formats such as SOAP, HTTP GET/POST or MIME. Thus a Web Services' WSDL document provides an adequate description of the entire specified Web Service by defining its operations, the operation parameters and the format of SOAP messages to be exchanged.

The Simple Object Access Protocol (SOAP) standard [5][6] is a lightweight communication protocol based on XML for information exchange between various entities of web services. SOAP is typically transmitted over HTTP although other transports are possible. A SOAP message consists of an outermost element called the envelope. In turn the envelope consists of a compulsory body element and an optional header element. The body element carries the content of the message which for example could be name of an operation to be performed and its parameters. The header is responsible for any additional useful information which the Web Service may require, for instance password authentication.

3. BACKGROUND

There has been plenty of research into providing availability of Files [7] [8], Databases [9][10][11], Web Pages [14] [15] [16] [17] and Remote Objects [12] [13] during periods of disconnection on mobile devices via caching. However, to date there has been only one significant exploration and contribution to cache XML Web Services to support disconnected operation on mobile devices [18]. The ultimate goal of all these systems is similar if not the same and as such, they share a common architecture - that being the cache. How these systems differ is dependant on what is being cached and how, as well as the degree of co-operation between clients and servers. Due to the diverse nature of web services the matured strategies employed above to provide availability of file, databases, objects and web pages is not applicable to XML Web Services.

Techniques and strategies used to support availability of File Systems and Databases are well researched and implemented. These traditional systems export relatively straight forward operations such as Read, Write, Open and Close. Therefore, the cache manager implements a rather simple interface. In the case of Web Services, each service exposes its' own unique interface. A Web Service Cache must somehow be able to categorize each operation the service exports, in order for the cache to know how it should be handled. Thus the semantics of a Web Service need to be known to the cache manager. This hints at the possibility of any web service requiring their own specialized cache manger.

Web Services encapsulate their data in critical business logic which, in fact this is one of the selling points of Web Services. Having clients that are able to utilize this high level logic [18] means an attempt to replicate data of Web Services to support disconnected operation of the service is not a sufficient technique [7] [19] [8] [32].

Distributed object systems also employ caching to support availability during periods of disconnection as well as increased performance [21] [12] [22] [13] [23]. The common ground between objects and web services is that they both expose operations through published interfaces. The techniques utilized to support availability of distributed objects relies on code and data (the objects) being cached in their entirety. This technique is not applicable to Web Services as resources are most probably limited on mobile devices and the Business Logic of Web Services is often proprietary.

Web Browsers are well established in supporting a cache for web pages [24]. However the current status and research with respect this realm of caching to support usage during disconnection, focuses on maximizing the cache hit rates while only preserving a mediocre degree of cache consistency. Caching web pages is dependant on directives provided by Web Servers to indicate what is cacheable and for how long what is being cached is fresh for. Web services are active entities and so the passive nature of caching web pages and the mediocre consistency it provides is not adequate to apply to web services.

Consequently no current strategies for providing availability of files, databases, web pages and distributed objects are applicable to XML web services. Therefore a new approach is required to

provide access to Web Services for intermittently connected devices.

4. RELATED WORK

4.1 An experiment in caching XML Web Services

XML Web Services present new challenges, due to the diverse set of operations exported and supported by such services as well as the lack of involvement in the caching process [18].

The first and only significant exploration of caching of XML Web Service was conducted by Douglas B. Terry and Venugopolan Ramasubramanian at Microsoft Research Silicon Valley. The study was conducted to research the suitability of caching to support disconnected operations on Web Services. An experiment was conducted in which a caching proxy was placed between Microsoft .NET My Services[1] and the sample client that was shipped with these services. This suite of services represented a non-trivial XML Web Service that supports both query and update operations [18].

The Microsoft .NET My Services contained many different services, including: MyContacts, which allowed users to store and retrieve address and phone information; amongst others [18]. This service was used in the running of the experiment. This particular service was chosen because it exported four significant operations on shared databases:

Query lets users select portions of the database. This operation takes in a String as the query argument.

Insert takes data to be inserted into the database, and uses a String argument specifying the location for the insertion.

Delete takes a query argument and deletes the queried entries from the database

Replace performs an atomic **delete** and **insert** in the database.

These operations were used to test sample applications having the network connected, as well as deliberately disconnecting the network [18].

They team of Terry and Ramasubramanian built a HTTP proxy as defined in the HTTP protocol standard and deployed it on a client device. In this case, the device was a laptop running the Windows XP operating system. All HTTP messages originating at the client were made to pass through the proxy server. The proxy server simply acted as a tunnel for all HTTP packets that are not SOAP messages [18].

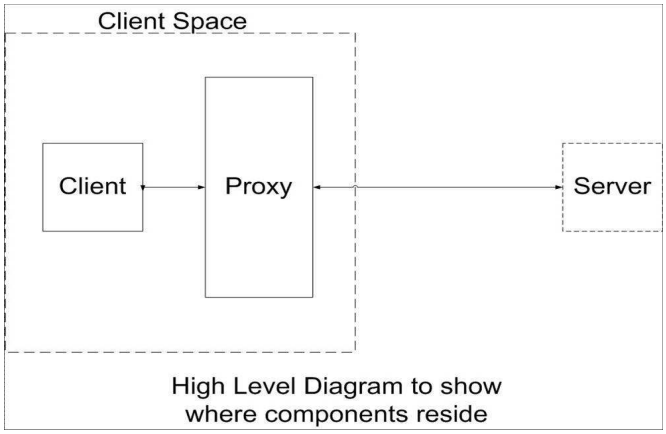

Figure 1

They proceeded to add a cache for storing SOAP messages to the proxy server. Integrating the Web service cache into the proxy served to meet the requirement of transparency. This cache stored the SOAP request received as well as the complimentary response provided by the web service server. All cache policies for expiration and replacement were implemented according to the HTTP standard. The cache served to store only HTTP messages that had SOAP messages as its entities. Whenever the system was acting in connected mode[2], the SOAP request message was sent to the server, and the received response was stored in the cache along with the associated request, replacing the old response if necessary. When the cache contained a previous response for the same request, the new response was compared with the old response, and the result from the comparison was recorded in a log file [18].

If the network was disconnected[3], the SOAP response message stored in the cache (if present) was returned to the client and the SOAP request was stored in the write-back queue. If the cache has no stored response, the client times out waiting for the response, as per normal when the server is unreachable. All requests are stored in the write-back queue for later replay, because the cache manager cannot determine which request modifies the service state and which are simple queries. The write-back queue is responsible for periodically checking the network for connectivity. Whenever the connection to server is restored, the request messages stored in the write-back queue are played back to server and the resulting SOAP response messages are stored, along with their complimentary SOAP request messages, are stored in the cache [18].

[1] Microsoft .NET My Services (Hailstorm) has since been discontinued, after AOL threatened to sue Microsoft on the ground that the suite was encroaching on the market created by their product AIM (AOL instant messenger)

[2] The server was online and the system acts normally

[3] The server is disconnected from the system; therefore the system is relying on the cache for the responses to the requests received.

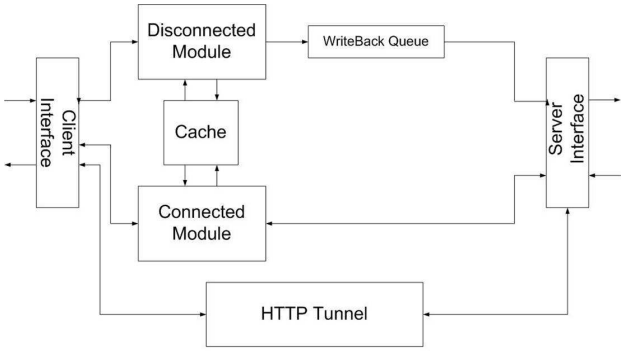

Figure 2

The above experiment highlights the benefits of employing a Web service cache to support disconnected operations [9]. It showed that Web services can be operated offline to some degree of success. Specifically it showed that the test users of Microsoft's .NET MyContacts while using the above system, were little aware of interruptions to the disconnectivity to the network service. It also showed that Web services can be operated, although to a somewhat limited degree, in a disconnected mode without alterations being made to the server or the client implementations [9].

4.2 Issues in Caching XML Web Services

The experimentation done by the Microsoft team of Terry and Ramasubramanian raised several issues that we intend to tackle to improve the consistency and availability of offline access to Web Services.

4.2.1 Playback and Cacheablity

Due to the diverse nature of Web services a major problem is posed as to the identification of the semantics of the operation exposed by the Web service. Unlike traditional systems, such as file systems, where the interfaces exported to the client are all pretty much standardized, Web Services almost always export unique interfaces. So the question that arises is whether the operations exposed by these interfaces have to be played back to the server or whether an old cached response can be acceptably returned to the client [9].

A successful Web service cache needs to understand the properties of an operation to effectively function; that is to access the cacheablity and/or the playbackability of an operation. An operation is said to playbackable if, the execution of such an operation leads to a permanent change in the state of the server. An operation is said to be cacheable, if the subsequent execution of such an operation with the same parameters will return the same response. An operation may fulfill either, both or none of these properties [9]. It is up to the developer of the system to deduce this.

4.2.2 Consistency

While operating in disconnected mode the cache cannot provide strong consistency, simply due to the fact that there is no access to updates performed by other users using and making changes to the same Web service. However, another solution that can be adopted is to provide weak consistency[4]. In particular the consistency requirement of each operation needs to be well understood, i.e. which operation would invalidate another operation if executed. For example, an insert operation or query operation on a database system would be invalidated by a delete operation. The goal being to develop a Web service cache which correctly understands the semantics of Web service operations that would enable invalidation or perform transformations of prior operations thereby providing local consistency and increasing the overall effectiveness of the cache proxy system. The ultimate effectiveness of the system would depend on how well it understands the consistency requirements of diverse Web Services [9].

4.2.3 Request and Response Messages

The understanding of the format of messages exchanged between the Web service client and server can pose yet another problem to a caching proxy. Despite using standardized protocols such as SOAP, web services deviate considerably in the structure and format of the request and response messages. Even operations such as identifying the name of the operation being performed vary from service to service. Complicating things further, we may find that operation names may have to be identified in a completely different way for different Web services. This may have drastic consequences on the transparency of the overall system [9].

The correct comprehension of the message structure is also required to accomplish many other fundamental tasks such as comparison of request, and the fabrication and generation of responses. A cache manager must also be aware and be able to understand which elements of a request message should be used as keys for cache lookups. The cache manager must be able to accurately distinguish and differentiate between similar requests, or every request would be cached and the cache would be rendered useless.

The message structure of the request and response must be clearly defined and documented to be effectively categorized the request/response to determine the correct course of action. This is needed since when a cache receives a request for a service operation during disconnected activity, it is expected to return a meaningful response to the client as to deceive the client that it is still operating under normal connected mode of operation i.e. the service is still available. The cache can either send a previously fabricated message or, generate a response that conforms to the message format of the service. Currently WSDL specifications contain enough information to permit fabrication, but lacks information about reasonable default information for each element of the fabricated response. An effective algorithm can only be devised through thorough examination of the SOAP messages of the Web service in question.

5. SOLUTIONS TO THE ISSUES

This sections details the specific solutions to the various issues in caching XML Web Services which this project tackled. A prescribed set of WSDL annotations to facilitate caching are

[4] Weak consistency strives to keep the cache consistent with the user's own actions

discussed. Following this an algorithm to keep the cache locally consistent is explained. Next a discussion follows regarding the generation of responses follows.

5.1 WSDL Annotations

The majority of the issues in caching XML web services are due to diverse nature of the services. In order facilitate the effective caching more information is required regarding the properties of an operation. Is the operation cacheable? Does it cause permanent changes to the server state? A generic technique for providing and describing this information is required. A web services WSDL document provides information as to what operations are exported by the web service and the operations involved but lacks information to assist caching. Thus, as suggested by the experiment [18] the WSDL should be extended to provide such information. Doing so would not affect tools which generate web clients based on WSDL documents as the annotations would be optional. Also this solution satisfies the two governing principles of the system namely, transparency and general applicability.

5.2 Properties of Operations

To facilitate effective and efficient caching of Xml Web services the following attributes can be added to a Web Services WSDL document, in particular to the operation element. The operation element specifies the messages involved in each of the operations provided by the Web service. The attributes elaborate upon the semantics and properties of each operation exported by the web service and so it was natural to include them in the operation element.

One complication is the difficulty in determining the properties of an operation. The *cacheable* and *Playback* attributes defined below serve to resolve the issue described in section 3.3. These two properties of an operation need to be recognized in order for the web service to function properly.

Attribute	Description
Cacheable	This Boolean attribute specifies whether the operation is cacheable or not. An operation is cacheable when subsequent executions of the operation using a particular parameter results in an identical response. For example, typical get and query operations such as `getName(), getAddress()`.
Playback:	A Boolean attribute which the cache manger uses to determine whether the request currently being processed should be replayed to the server once the connection is restored. An operation which results in a permanent change to some state on the server side is an update operation, and therefore needs to be played back to the server upon reconnection. For example, whenever `setAddress(Bob, 21 NewAddress Road)` is called while the web service is being used during disconnection, it must be sent to the server
Generate Response	Another Boolean attribute which specifies whether the cache manager can generate a default response for the web service if one is not cached.
Type	The final attribute that can be added to the operation element is a type attribute. That is used to keep the cache store consistent. A attribute which categorize the operation. It can be one of 5 values, “delete”, “set”, “replace”, “query” and “undefined”.

5.3 Identifying SOAP messages

In order to process SOAP requests and responses properly, the Web service cache needs a method to distinguish between SOAP messages via the operation being performed (A.K.A operation name) and its parameters. Each operation name together with one or more of its parameters can be used to uniquely identify any request response pair. This combination of operation name and parameter(s) can also be used to calculate a cache key for the request response pair. Determining the operation name is an easy affair as the first child element of the SOAP body (section 4.1.2) is always the name of the operation being performed. However there is no way to distinguish which of a message’s parameters uniquely identify a request response pair. Thus an attribute has been defined which is added to the Binding Element is the Web Services WSDL document. The binding element describes the components of the SOAP messages that are exchanged.

Identifier: The attribute is an XPATH string which represents the path to a parameter which when used in conjunction with the operation name can uniquely distinguish a request. XPATH is an XML Path Language is a W3C [1] (see section 2) standard used to query, navigate and identify specific portions of XML documents.

5.4 Cache Consistency

As with any caching solution the cache store needs to be consistent with, that is be as up to date as possible with the server-side to operate effectively. Once again the diverse nature of Web services is the source of complications. All web services have custom interfaces and so identifying and controlling read/write and write/write conflicts is an issue. Updating the cache store with the server is simply not possible during periods of disconnection and the cache would be inconsistent (out of date) at such times. As the experiment into caching web services [18] outlines it is possible to provide local consistency that is a cache which is consistent with the users own actions. This section outlines the design to implement local cache consistency.

In order to keep the cache consistent with the users own actions the cache manager needs to determine whether the current request being processed can invalidate any of the request response pairs stored in the cache. Invalidation can be detected by determining whether the operations intersect each other and whether the parameters of the two operations match. For example, `request1` which is a `getCellNumber(Bob)` operation, that returns the Bob’s cell phone number, is stored in cache along with it’s corresponding `response1`. The next operation, `request2 a deleteCellNumber(Bob)` is performed. Request 1 and 2 intersect and the parameters match, therefore `request2` should invalidate `request1`. Thus determining whether a request invalidates another is a function of the operations names and parameters:

```
Operation1(param x,…,param x-1)
```

```
Operation2(param y,…,param y-1)
Invalidation(operation1, operation2, param x, … , param y-1)
```

Determining the operation names and parameters of the requests and responses utilizes the prescribed WSDL annotations. The invalidation algorithm also relies on the WSDL document labelling `getCellNumber` as a query operation and `deleteCellNumber` as a delete operation. It also knows from WSDL document that the parameter of `getCellNumber` is a cell phone number that is the same cell phone number as the parameter of `deleteCellNumber`. Finally it needs to determine the *common denominator* `CellNumber` in `getCellNumber` and `deleteCellNumber` which can be easily achieved through simple string manipulation. If an annotated WSDL document does not exists for operation types then the invalidation algorithm resorts to matching keywords such as "del", "get" to operations to determine an intersection.

Whenever a new request is being processed the cache manger applies the invalidation function or cache update function to each of the entries stored in cache along with the current request. This is an expensive process but it is the only way to maintain a consistent cache during disconnections. Below is a very basic pseudo code version of the algorithm.

```
For Each Cached Entry
If (Current Request is a DEL|REPLACE|SET) & (Cached Request is a QUERY)
        If(Parameters match)
                Make appropriate changes
```

5.5 SOAP RESPONSE FABRICATION

This module is at the core of the system that we have designed. It is responsible for creating properly formatted SOAP responses in response to SOAP request that are received from the client. The understanding of the structure of the SOAP request/response messages exchanged between client and server is key to the success of the overall system. Despite the fact that Web Services are now mostly using the de facto standard of SOAP - Web Services deviate drastically with the respect to the structure of their SOAP messages. Even mechanisms for the identification of operation name vary from service to service.

When a client issues a SOAP request during disconnect operations, our system is expected to return a properly formatted, meaningful response, so that the client can pretend to be still functioning under normal circumstances. If this operation is deemed cacheable, and the specific response has not been previously cached, then this fabricated response has to cached so that it can returned to the client the next time the operation is processed.

To properly design an effective algorithm for the fabrication of such responses, needs a deeper understanding of the operations exported by the Web service, and the correlations and dependencies between the SOAP service request and the resulting SOAP responses.

By the examination of the structure and correlations between SOAP request and their resultant response, we can classify almost all of these operations into one of at least four categories.

1. Those responses that are returned as default response
2. Those responses that require values extracted from the request to fabricate the response
3. Those responses that require values to be extracted from other request
4. Those responses that are special cases

By annotating the specific portType, of the Web services's WSDL document, that references that specific operation, so that it corresponds to one of the above four categories. We can now implement the algorithm which understands the semantics of the SOAP requests by referencing the annotated WSDL to accurately ascertain the proper course of action when fabricating a properly formatted, meaningful response. This response can be returned to the client, so as to deceive the client into believing that it is still operating under normal connected circumstances. The general algorithm is described in the diagram below.

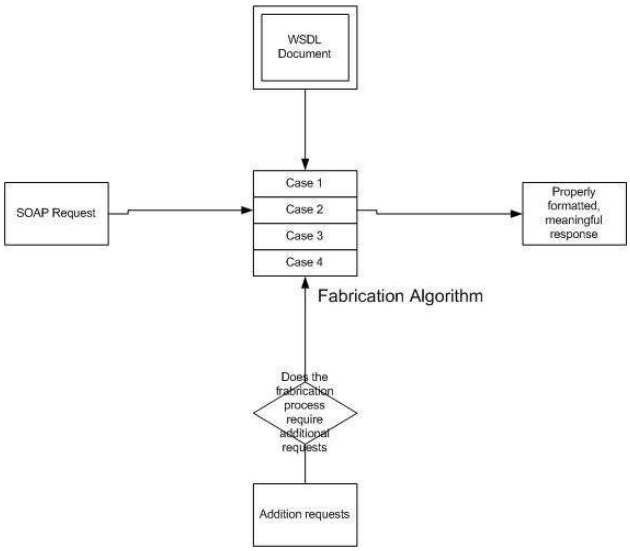

Figure 3: The general algorithm for the fabrication of SOAP responses

6. SYSTEM ARCHITECTURE

An HTTP proxy server intercepts messages passed between the client and server. A message containing a SOAP request or response is determined via the presence of an SOAPACTION header within the HTTP header. HTTP messages are simply tunneled to the respective server. SOAP messages are passed to the inherent Web Service Cache. A Specialized Cache Manger is instantiated for every Web Service with an annotated WSDL file. Web Services without an annotated WSDL are handled by the Default Cache Manger.

The cache store is responsible for storing and retrieving the requests and responses together with the corresponding unique numeric. The cache key is calculated by hashing the operation name and parameters of the request. A pre-determined number of request response pairs can be stored. Cache elements are stored on disk and exist for a lifetime specified at runtime; upon expiry the elements are deleted. Finally the cache store implements a Least Recently Used (LRU) policy to cache a request response element when the cache is full.

Messages that are deemed playbackable[5] are placed on the Write-Back queue during the first stage of the disconnected module, due to network outages at the time of reception. The Write-Back Queue module is responsible for polling the server periodically to check for connectivity and playing back queued up request to the server. Queued messages are stored as files on the local host. Periodically, the write back queue module checks for network connectivity by issuing the first request in the queue to the server. If the network is still disconnected, the write back queue waits a pre-determined duration and repeats the same issuing process again to check for network connectivity.

If connectivity between the server and the cache-proxy system is restored, then the Write-Back queue module purges its internal queue of the stored messages and plays this back to the server. These service requests are played back to the server in a FIFO (First in First out) order. The resulting responses returned from the server are given to the cache manager to be inserted into the cache and the necessary transformations (e.g. replacing outdated cache entries) done on the contents of the cache to maintain an acceptable level of consistency.

A Specialized Manager is automatically created for each Web service that has an annotated WSDL file. The Specialized Manager then behaves according to the WSDL annotations.

If the network is connected the request is forwarded to the server and the response waited upon. If the response is a cacheable operation it is cached. Before the response is forwarded to the client the consistency algorithm is applied to the cache.

If a disconnection is being experienced the Specialized Cache Manager attempts to either locate the appropriate cached response or generate a suitable response. If the request is not cacheable a suitable response is generated and returned to the client. If the request is cacheable and a response is cached then it returns that response to the client. However, if no response is found in the cache the request is checked to see whether it is of type "query". If it is then a response can be generated provided the corresponding "insert" request is cached by using the contents of the insert message. In the case where no corresponding "insert" is found the operation is unavailable in disconnected mode and the client will time out.

The default cache manager processes all requests and responses for Web Services' that do not have an annotated WSDL document. It can operate in two modes each assumes a specific configuration of default values for the missing annotations. The "mode" is set a runtime. "Mode 1" assumes that all requests are cacheable and replays all requests to the server. "Mode 2" conservatively treats all requests and responses as non-cacheable. "Mode 1" serves to maximize availability of a server during periods of disconnection as is how the cache manager in [18] operated an mode 2 simple tunnels messages.

7. Evaluation

The Web Service Caches' ability to provide seamless online offline transitions with losing as little operation of the service as possible was tested by using the cache with a Web Service that models a car rental service. Exercising the service with the proxy deployed and simulating disconnections; it was demonstrated that the system was successful in providing disconnected operation for the Car Rental Web. All aspects of the system were successful in design and execution. The prescribed set of WSDL annotations proved worthy in dictating the behavior for an effective and efficient cache. Specifically, these annotations demonstrated that they provide sufficient semantic information of the web service to assist generation of default responses, maintain a consistent cache and generally enhance the effectiveness of the cache.

With respect to cache consistency the Web service used to test the real world functionality of the system was a little lacking in terms of operations. There was only one combination of operations which truly tested the cache consistency algorithm. If there was a replace operation, one which modified a record, the consistency algorithm could be improved to not only invalidate responses but also modify them. This smarter consistency algorithm could locate a path, specified by the "replace" request, in the query response stored in the cache, perform the respective operation and modify the query response. Nevertheless, the current consistency principle is sufficient and sound.

For web services lacking WSDL annotations the default Cache manager provided a configurable level of availability to best suit a Web Service. Mode 1 which caches all requests and responses would be useful to provide a high cache-hit rate for classes of web services which predominantly consist of query operations on static data. Such services like currency conversion services, map services and directory listings [18]. In this mode playback is set to true for all operations so that if any update operations are executed the server is notified upon reconnection. This in turn requires that a default response be generated for requests which are cache-misses. Playing all requests back to the server and generating some responses is a costly affair however, it is a necessary. Conversely, modes two treats all requests as non-cacheable and are not placed in the write back queue. It simply relays messages between client and server. Thus service is unavailable and would best suit services which involves plenty of insert, deletes and replaces.

Finally, the Web Service, Car Rental, which was used to evaluate the system, was not as complicated as planned. It only exported three of the required four operations. This does not negate the success of the project but rather suggests that to completely qualify the system it needs to be further tested using multiple non-trivial XML Web Services.

8. CONCLUSION AND FUTURE WORK

In this paper, a XML Web Service Cache solution has been described. The goal was to design and implement a solution to provide access to a web service despite frequent connections and limited bandwidth. The solution had to be transparent and generally applicable to all Web services. Building upon the design of a previous experiment [18] into caching XML web services we built a HTTP cache proxy that cached SOAP requests and responses. The experiments' [18] design was extended upon by prescribing WSDL annotations which enhance the effectiveness of the cache managers. Modules to keep the cache locally consistent and generate default responses were incorporated into

[5] A message is deemed playbackable are requests that make permanent changes to the state of the server. These are usually "update" requests, i.e. insert, delete or modify requests.

the system. A prototype was built and thoroughly tested using a web service which models a car rental service. The system satisfactorily provided a relatively high degree of availability during intermittent disconnection demonstrating that caching web services is viable and useful. However, the system needs to be used with a number and variety of web services to completely explore the utility of such a system. Areas for future improvements include a smarter cache consistency algorithm – one which not only invalidates responses but can perform modifications to cached response. There are still other issues remaining with respect to caching web services which need to be addressed such security and user experience [18]. Ultimately, what would be ideal are web service tools which can automatically deduce properties of a web services' operations and provided the necessary annotations to its WSDL document.

9. REFERENCES

[1] Daniel Barbara and Tomaz Imielinski. Sleepers and Workaholics: Caching Strategies in mobile environments. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, 1994 pages 1-12.

[2] Bharat Chandra, Mike Dahlin, Lei Gao, Amjad-Ali Khoja, Amol Nayate, Asim Razzaq and, Anil Sewani. Resource management for scalable disconnected access to web services. Proceedings of the tenth International Conference on World Wide Web, 2001, Hong Kong, Pages: 245-256.

[3] Boris Y. L. Chan, Antonio Si, Hong Va Leong: Cache Management for Mobile Devices: Design and Evaluation. Proceedings of the Fourteenth Internation Conference on Data Engineering, February 23-27, 1998, Orlando, Florida: pages 54-63.

[4] Bray T, Paoli J, C. M. Sperberg-McQueen, Eve Maler. 2000. W3C Recommendation "Extensible Markup Language (XML) 1.0 (Second Edition)". (See HTTP://www.w3c.org/TR/2000/REC-xml-20001006.)

[5] P. Cauldwell, et. al. Professional XML Web Services. Wrox Press Ltd. Birmingham, U. K. 2001.

[6] Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, Sanjiva Weerawarana. W3C Working Draft "Web Services Description Language (WSDL) Version 1.2", 9 July 2002 (See HTTP://www.w3c.org/TR/wsdl12/.)

[7] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau Henrik Frystysk Nielsen. W3C Working Draft "SOAP 1.2 Part 1 Messaging Framework", 2 October 2001. (See HTTP://www.w3c.org/TR/soap12-part1.)

[8] Martin Gudgin, Marc Hadley, Jean-Jacques Moreau Henrik Frystysk Nielsen. W3C Working Draft "SOAP 1.2 Part 2 Adjuncts", 2 October 2001. (See HTTP://www.w3c.org/TR/soap12-part12.)

[9] R.G. Guy, J.S. Heideman, W. Mak, T. W. Page, Jr., G.J. Popek, and D. Rothmeier. Implementation of the Fiscus replicated file system. Proceedings Summer USENIX Conference, June 1990, pages 63-71.

[10] Kahol, S. Khurana, S.K.S. Gupta and P.K. Srimani. A strategy to mange cache consistency in a disconnected distributed environment. IEEE Trans. On parallel and Distributed Systems, Vol. 12. No. 7, July 2001 pp. 686-700.

[11] James J. Kistler, M. Satyanarayanan, Disconnected operation in the Coda File System, ACM Transaction on Computer Systems (TOCS), v10 n.1, p.3-25, Feb 1992.

[12] Antony D. Joseph, M. Frans Kasshoek, Building reliable mobile aware applications using rover toolkit. Wireless Networks, v.3 n.5, p.405-419, Oct 1997.

[13] Barbara Liskov, A. Adya, M. Castro, S. Ghemawat, R. Gruber, U. Maheshwari, A. C Meyers, M. Day, and L. Shira. Safe and Efficient Sharing of Persistant Objects in Thor. Proceedings International Conference on Managament of Data (SIGMOD), 1996, Montreal, Quebec, Canada, pages 318-329.

[14] Ing-Ray Chen, Ngoc Anh Phan, and I-Ling Yen. Alogorithms for supporting Disconnected Write Operations for Wireless Web Access in Mobile Client-Server Enviroments. IEEE Transactions on Mobile Computing, Vol. 1, No 1, January-March 2002, pages 56-58.

[15] Rick Floyd, Barron Housel and, Carl Tait. Mobile Web Access using eNetworks Web Express. IEEE Personal Communications, Vol. 5, No 5, October 1998, pages 47-52.

[16] Douglas B Terry, Venugopalan Ramasubramanian. For Mobility. QUEUE. May 2003. Page 71 -78.

[17] Joanne Holliday, Divyakant Agrwal, and Amr El Abbadi. Disconecction Modes for mobile databases. Wireless Networks, Issue 8, 2002, pages 117-157.

[18] D. B. Terry. Ramasubramanain, V. Caching of XML Web Services for Disconnected Operation. Microsoft Research.

[19] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, C. H. Hauser, Managing update conflicts in Bayou, a weakly connected replicated storage system, Proceedings of the fifteenth ACM Symposuim on Operating System Principles, Copper Mountain, Colorado, December, 1995, pages 172-182.

[20] Gregory V. Chockler, Danny Dolev, Roy Freidman, and Roman Vitenberg. Implementing a Caching Service for Distributed CORBA Objects. Proceedings International Conference on Distributed System Platforms (Middleware), New York, 2000, pages 1-23.

[21] R. Kordale, M. Ahamad, and M. DevaraKonda. Object Caching in a CORBA Compliant System. Proceedings USENIX Conference on Object Orientated Technologies. (COOTS), Toronto, Canada, June, 1996

[22] Micheal N. Nelson and Yousef A. Khalidi. Generic Support for caching and Disconnected Operation. Proceedings Fourth Workshop on Workstation Operating Systems, Napa, CA, October, 1993, pages 61-65.

[23] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk Nielson, T. Berners-Lee. IEFT "RFC 2616: Hypertext Transfer Protocol- HTTP/1.1" January 1997. HTTP://www.IEFT.org/rfc/rfc2616.txt.)

[24] World Wide Web Consortium. HTTP://www.w3c.org