Portal Replication for Web Application Availability Via SOAP

Simon Woodman, Graham Morgan & Simon Parkin
School of Computing Science, University of Newcastle
Newcastle upon Tyne NE1 7RU, UK
{S.J.Woodman, Graham.Morgan, S.E.Parkin}@ncl.ac.uk

Abstract

The interoperability of SOAP has eased the provision of B2B (business to business) solutions where web applications (web form submission via browser) are only suitable for supporting B2C (business to client). As many services exist that are delivered to clients via web applications, enabling the delivery of such services via SOAP to provide B2B solutions is desirable. One mechanism for achieving this is via the use of portals. A portal may provide clients with a single point of access to geographically distributed services and tailor such services to satisfy client requirements. As a single point of access to many services, a portal may present a single point of failure. The provision of fault-tolerant portals may be viewed as essential by an organisation. This paper describes a portal suitable for exhibiting web applications as SOAP based services and presents an approach for replicating portals to overcome the problem of a portal representing a single point of failure.

Keywords and phrases: SOAP, replication, group communications, Web services, portal, CORBA

1. Introduction

We are concerned with a particular class of Internet-based applications that provide clients with a single point of access to geographically distributed services. This class of applications are commonly described as *portals*. The term portal usually refers to an application that provides access to information sources, integrating heterogeneous data systems and possibly allowing users to tailor information presentation to satisfy their individual requirements [11]. Such portals are termed *Enterprise Information Portals* (EIPs). EIPs are specifically designed to deliver information to end users via web browsers, which makes them appropriate for delivering business-to-client type services. The provision of such services to enable organisations to use each other's services when satisfying end user requirements is considered an integral part of future web application development. Examples of this can be seen in attempts by Amazon.com and Google to provide access to their services in a manner suitable for satisfying business-to-business requirements [12]. It is now possible for an organisation to utilise Google search engine technologies in their own web based applications without actually requiring Google technologies to be installed within their organisational domain. *Web services* are promoted as providing a suitable paradigm for application integration across organisational boundaries. Services may be implemented and deployed using platform specific mechanisms with interoperability achieved via Web service standards and communications over standard protocols. The protocol specified by Web services for ensuring interoperability is the Simple Access Object Protocol (SOAP) [20].

Web services, including SOAP, are specified using the Extensible Markup Language (XML) [6]. XML allows a developer to represent different elements of data in a text file that may be read and processed by applications. SOAP provides extendable XML-based mechanisms allowing data exchange between distributed applications and may be used with a number of network protocols (e.g., Hyper Text Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP)). The use of SOAP over HTTP is viewed as a solution for providing interoperability between web-based applications [13].

We propose the use of a portal that allows clients to access web applications provided by different organisations via SOAP. This has two main advantages:

- Organisations can allow their web forms to be exhibited in a manner more appropriate for satisfying business-to-business solutions without the need to reengineer their own services.
- A portal may tailor services derived from a number of different organisations to satisfy client requirements.

As a single point of access to services, the failure of a portal will inhibit clients from accessing services even when such services may be reachable by clients and functioning correctly at an organisation's site. A mechanism widely used to increase the availability of a service is replication: a service is replicated over a number of nodes in a network and as long as one of the service replicas is correctly functioning and reachable by clients then client requests may be satisfied. Therefore, we propose portal replication to ensure a portal is not a single point of failure.

Efforts to provide fault-tolerant services via group communications [18] using Common Object Request Broker

Architecture (CORBA) [22] technologies are amongst the most successful [14] [15] [16] [3]. These approaches, and in particular [16], have resulted in a fault-tolerant specification for CORBA [21]. CORBA's goals of interoperability through standardisation of services and protocols are similar to that of Web services. The choice, for application developers, between these two technologies is a subject of debate [17]. Rather than develop a new replication service, we propose the refining of an existing CORBA group communication service for use in portal replication.

In the next section we describe the design and implementation of our portal. Section 3 describes the ability to tailor existing services for use by portal clients. Section 4 presents an approach to portal replication. Performance figures of our system are presented in section 5 and concluding remarks are presented in section 6.

2. Design and Implementation

We have implemented a portal that allows client requests, issued using SOAP, to be satisfied by web applications that exhibit their functionality via web forms. Java was chosen as the implementation language as a number of freely available Java based technologies eased the development of our service and the suitability of Java for handling XML, HTTP and SOAP is well known. The diagram in figure 1 presents an overview of our portal.

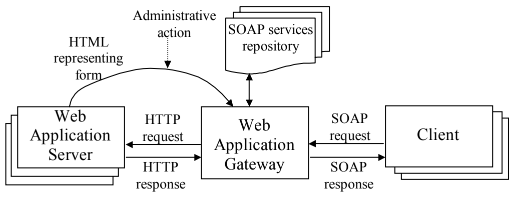

Figure 1 – Overview of system.

The *Web Application Gateway* (WAG) together with the *SOAP Services Repository* (SSR) are the main components of our portal implementation. The WAG may handle simultaneous client requests and may satisfy such requests via an arbitrary number of web application servers. Before enabling client requests, a service must be registered in the SSR by an administrator. This action is trivial, requiring an administrator to submit a configuration file that includes a URL that identifies the web form that will satisfy a proposed service. After registration of a service, the WAG automates the provision of exhibiting a web form as a service and the handling of client requests. We now continue with detailed descriptions of service creation and service invocation.

2.1 Service creation

Passing configuration parameters to the WAG instigates the process of creating and making available to clients a service. Configuration parameters are contained within a file (a configuration file is provided on a per-web form basis), which is supplied by an administrator and may be submitted via an administrative interface (web form). A configuration file contains parameters related to the following:

- **Location of form** – URL of a web page containing the form that is to provide the basis for creating the new service. When required, proxy server information is supplied.
- **Poorly formed data** – Indicate appropriate action to take if errors occur in the processing of data (possibly due to poorly formed HTML).
- **Working offline** – Indicate if the process of generating a service is to be achieved with access to remote services or rely only on services found within the domain of the local machine.

A *RPC file* provides the information to allow the mapping of a client request to a web application request (form submission) and is provided on a per-form basis. We now describe the processes involved when deriving a RPC file with the aid of the diagram in figure 2.

On the submission of a configuration file (step 1), an attempt to retrieve the web form indicated by the URL is made. As the web form may have to be retrieved via a proxy server, details relating to the proxy server are also included in the configuration file. On successful retrieval of the web page (step 2) any unnecessary HTML is removed (e.g., formatting, references to images, descriptive text) (step 3). To ensure the success of later stages of the creation process, it is important that the resulting HTML document is still valid (readable by a browser). Therefore, required tags are unaffected (e.g., <HTML>, <BODY>).

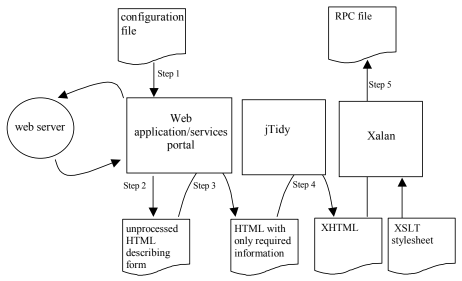
configuration file
RPC file
Step 1
Step 5
web server
Web application/services portal
jTidy
Xalan
Step 2
Step 3
Step 4
unprocessed HTML describing form
HTML with only required information
XHTML
XSLT stylesheet

Figure 2 – Deriving RPC files.

Once the retrieval and initial formatting of the HTML file has been achieved, a RPC file may be derived. RPC files are automatically generated from the retrieved HTML description of a web form. However, the generation of the RPC file is not possible if the HTML describing the form is not well formed. Therefore, jTidy [8] is used (step 4) to convert the HTML to an *Extensible Hypertext Markup Language* (XHTML) document that is well formed and so readable by an XML parser. It is worth noting that a configuration file may indicate an appropriate action to take if there is a failure of any of the parsing stages when deriving a RPC file. For example, failure of jTidy to produce XHTML may result in the "best effort" output of jTidy to be placed in a temporary file, allowing an administrator opportunity to fix jTidy output manually before continuing with the RPC file creation process.

After successful generation of valid XHTML describing the web form, an *Extensible Stylesheet Language Transformation* (XSLT) [10] *stylesheet* is applied, creating a RPC file (step 5). A stylesheet indicates to an XSLT transformer how to achieve the required transformations. Apache Xalan [9] was chosen as the XSLT transformer tool because *Transformation API for XML* (TrAX) is supported (a common API that allows developers to write to consistent interfaces and apply transformations in a polymorphic manner). TrAX allows the passing of parameters by an application to the stylesheet, which override default values in the stylesheet. This is necessary to allow a RPC file to contain the URL of its associated web page. The target URL is not included in the original HTML of the web form and is derived from the configuration file. We make a decision to include the URL of the web form in the RPC file so a client may realise the organisation from which a service is derived. It is not the purpose of our system to hide from clients the identity of service providers. The processing of client requests associated to a RPC file may occur once the RPC file is registered in the SSR.

When deriving and manipulating XHTML documents, access to the XHTML *Document Type Definition* (DTD) is required. A DTD may be used to define structure and legal elements associated with XML documents. A copy of the XHTML DTD would, by default, be retrieved from a remote service (allowing multiple instances of the WAG to use the same, up to date, standard). However, it is possible to use a local copy and provide offline working. The location of a local copy of the XHTML DTD may be identified in the configuration file.

2.2 Service invocation

A client must retrieve a copy of a RPC file to realise the appropriate contents of a SOAP RPC request. RPC files may be downloaded from the WAG via HTTP requests from a browser or application. A single service capable of satisfying all SOAP RPCs located in the SSR is provided by the WAG. This is achieved by presenting a single RPC that takes two parameters:

- **Identifier of service** – The name of a RPC file that represents the service a client wishes to invoke.
- **Parameter list** – A hash table that contains the parameters expected by a service. The key to an element in the hash table is the name of the parameter and the value in the hash table is the value of the parameter.

The hash table format is suitable for parameter types where the mapping from name to value is one-to-one. However, parameter types that may assume multiple values (e.g., an element representing a group of check boxes) must be treated as a special case. Such parameters are still represented by a single element with multiple values separated by tokens. The ampersand is recognised as such a token, and identifies text separated by an ampersand as separate values. An ampersand may still be passed as a value with the use of the percent character followed by the ASCII code for ampersand. We now describe the processes involved in satisfying a client request with the aid of figure 3.

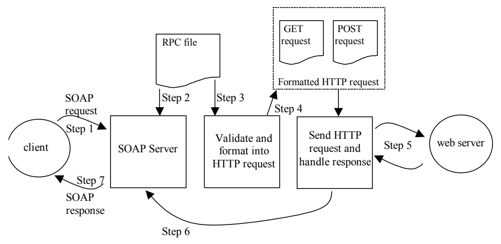

Figure 3 – Invoking services.

On receiving a request from a client (step 1) a check is made to identify if the request relates to an existing service (step 2). This is achieved by ensuring the identifier of service parameter in the client request relates to a RPC file in the SSR. If an appropriate RPC file exists then the hash table of parameters contained in the client request is validated against those represented in the RPC file (step 3) and an appropriate HTTP request is derived (step 4). The type of HTTP request (GET or POST) is described in the RPC file. HTML allows a number of *content types* to be used to encode the form data set for submission to a server. Content types identify how the form submission is to be interpreted by the server. The default content type *application/x-www-form-urlencoded* is supported (i.e., name is separated from value by '=' and name/value pairs are separated from each other by '&'). Other content types may be supported when required. The HTTP request is issued (step 5) with replies returned to the SOAP server (step 6). Replies are formatted to the appropriate return types as identified in the RPC file and returned to the client (step 7).

3. Tailoring Services

Satisfying a client request may require functionality derived from more than one service and may involve multiple organisations. For example, the WAG may present clients with interfaces relating to vacation purchases (e.g., airplane reservations, hotel reservations, car hire). A portal may allow a client to issue a single request that reserves all the elements required for a vacation booking with the functionality associated to each element satisfied by a different organisation. Furthermore, a number of organisations may be able to satisfy a single element of a vacation booking (e.g., multiple organisations offering airplane reservations). It is worth noting that client requests may be related and grouped into sessions. There is an expectation that a client may issue a number of requests in the booking of a vacation as the availability and cost of one service may dictate a client's choice of other services (e.g., choosing resort, hotel, then airline). Component architectures exist that successfully allow developers to incorporate sessions into their applications [7]. However, such architectural support in Web services is not yet suitable for the modelling of sessions across organisational boundaries. Considering the level of abstraction and scope of this paper we do not include the notion of Web service based sessions in our work.

We conclude that the following types of services may be exploited by the WAG:

- **Equivalent** – Services, possibly derived from different organisations, providing a choice to the WAG when satisfying a client request.
- **Composite** – More than one service, possibly derived from different organisations, is required to satisfy a client request.

We now describe how equivalent and composite services are managed by the WAG.

3.1 Equivalent Services

As each service is derived from a web form and is represented by a RPC file, there is a need to identify different RPC files as equivalent and provide an interface that allows clients to make use of equivalent services. As the variable identifiers used in a RPC file are taken from web form labels, the likelihood of these identifiers being the same across equivalent services is unlikely (e.g., in the case of airline seat reservations, the variable identifying the cost of a flight may be represented by the label "price" in one RPC file and the label "charge" in an equivalent RPC file). Furthermore, RPC files may be identified as being equivalent when they provide differing functionality. For example, there may exist a number of RPC files representing flight bookings that are each derived from different organisations and each present varying functionality (e.g., some airlines may allow clients to choose vegetarian meals whereas others may allow the identification of limited changes en route) yet still present the same subset of functionality (core functionality) that allows the WAG to consider them equivalent. Considering our observations, RPC files may be considered equivalent if differing functionality is derived from parameters that are optional. Considering our flight bookings example, if vegetarian meals must be specified in RPC file X, then a RPC file, say Y, may only be considered equivalent to X if Y also has a vegetarian meal option. This problem of differing functionality may be tackled one of two ways:

- **Transparent** – Clients are unaware of equivalent services and rely on the WAG to determine the choice of service provider. Any differing functionality is ignored and only functionality provided by all providers is exhibited.
- **Client driven** – The WAG provides a client with an informed choice of equivalent services, allowing a client to choose (based on variations in service) the appropriate service provider.

The client driven approach provides obvious advantages to the client over the transparent approach (by allowing the full functionality provided by a service to be utilised). However, additional processing is required at the client side to indicate to the WAG which service provider to choose. Therefore, a trade off between the transparent approach and the client driven approach is proposed: the client may specify provider, but this is not necessary when dealing with core functionality only. We now describe how equivalent services are represented and used within the WAG.

A RPC file (root RPC file) that represents the core functionality provided by equivalent services is created (manually by an administrator). This is used to remove the problem of differing variable identifiers across equivalent RPC files. A stylesheet per RPC file is produced to map the variable identifiers in the root RPC file to their alternate representation in an equivalent RPC file. The root RPC file presents clients with an interface to equivalent services. The root RPC file contains a list of all the information from each equivalent RPC file that may implement the core functionality. Optional functionality, and the equivalent service capable of providing such extensions over the core functionality, is described in the root RPC file. A client that uses the additional functionality is required to request the appropriate RPC file and use it directly (rather than use the root RPC file). This represents the client driven approach. If a client uses a root RPC file directly (only using core functionality), then the WAG will determine the service provider.

3.2 Composite Services

A composite service is created by combining the functionality described by more than one RPC file contained in the SSR into a single RPC file. Therefore, the functionality described by a composite service is also described elsewhere in our system in RPC files that are derived from web forms. Presenting a composite service via a RPC file does not make a distinction between the way a client accesses a composite service compared to a non-composite service. However, due to the nature of composite services (derived from RPC files and not directly from a web form), the URL that identifies the web form from which a RPC file is derived is missing. As no URL is present, there is a possibility that a client is unaware of the organisation(s) that implement a composite service. This approach is undesirable, as it is not the purpose of our system to hide from clients the identity of the organisations that may be satisfying their requirements. Therefore, we include identifiers that may be used by a client to discover the organisations that contribute to the implementation of a composite service. This is achieved by listing the identifiers of the RPC files that have been combined to create the composite service in the RPC file that represents a composite service. A client may discover the organisation(s) that implement a composite service by retrieving the appropriate RPC files as described in a composite service's RPC file.

A composite service's RPC file may be generated automatically by our system after receiving configuration details from an administrator. This process is trivial and simply requires a direct copying of the functionality described in contributing RPC files to the new composite service's RPC file. An administrator, via a web interface, lists the RPC file identifiers that are required to create the composite service's RPC file and provides a name for identifying the new RPC file.

When satisfying client requests via a composite service there may be a need for the WAG to submit web form data to multiple organisations. There is an assumption made by the WAG that the order in which data is submitted to different web forms is irrelevant when satisfying composite service requests. When dependencies exist between different requests (the choice of one request determines the input to another request), we suggest that a client manage this process by accessing the required RPC files separately.

A RPC file used to contribute to a composite service may be a composite service itself. Furthermore, composite services may be considered equivalent and may be constructed from services that may be considered equivalent.

4. Replicated Services

Equivalent services provide a redundancy that may be exploited by the WAG to provide continued service provision in the presence of limited service failure. If an organisation's service fails, then continued service may be provided to clients as long as a correctly functioning and reachable equivalent service exists. To make use of such a scenario it is important that the WAG itself is not a single point of failure. Therefore, we propose a system whereby WAG replicas are distributed over a number of nodes in a network.

Replicated processing is typically done in two different ways: active and passive. In active replication client requests are directed at each replica. Each replica then attempts to process the request and may reply to client requests. Even when equivalent services exist, a decision must be made to invoke only one of the equivalent services to avoid duplicated request processing. Therefore, we propose a passive replication scheme. Passive replication requires only one member of the replica group, the primary (sometimes referred to as the coordinator), to receive, process, and reply to client requests.

Group communication services have been shown to be useful in the development of applications that use replication schemes to achieve fault-tolerance [14] [15] [16]. We assume the availability of a group communication sub-system (NewTop [3]) that provides reliable, total ordered, delivery of multicast messages and ensures that members have a mutually consistent view of the order in which events (such as membership changes, invocations) have taken place. By reliable multicast we mean that either all or none of the functioning members (replicas in our case) are delivered a given multicast. Total order results in all functioning members delivering a set of multicasts in the same order that preserves causal precedence. The qualities exhibited by NewTop ensure replica states remain mutually consistent and that state changes are consistent with causal precedence. As NewTop is designed to support CORBA applications, tailoring of NewTop to work with our system is required.

The failure assumptions made by the NewTop service are the same as made in other group services referred to in this paper. It is assumed that processes/objects (in our case WAGs) fail only by crashing, i.e., by stopping to function. The communication environment is modelled as asynchronous, where message transmission times cannot be accurately estimated, and the underlying network may well get partitioned, preventing functioning members from communicating with each other. The protocols and the implementation details of the NewTop service will not be described here, as these details are not directly relevant to this paper; the interested reader is referred to [1] [2].

4.1 Client Invocations

A client may connect to any WAG replica to retrieve a RPC file. Once a RPC file is retrieved by a client, a client may issue requests. The WAG replica responsible for receiving and replying to client requests is known as the *request manager*. An assumption is made that a suitable mechanism exists that will enable a client to locate WAG replicas in the first instance (possibly using some location and discovery service similar to Universal Description, Discovery and Integration [23]). However, we consider the provision of a fault-tolerant location and discovery mechanism for Web based services as a further topic of research and is therefore beyond the scope of this paper.

Figure 4 identifies the overall architecture of WAG replication. The replica service satisfies the replication requirements of our system.

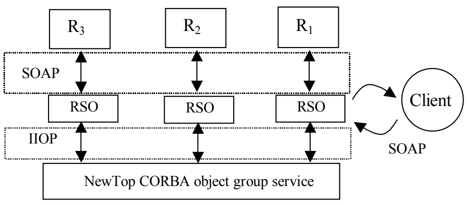

Figure 4 – Replicated Services.

The replication service is a distributed service and achieves distribution with the aid of the *Replica Service Object* (RSO). Each replica is allocated an RSO. Replication related service requirements of a replica are satisfied by its RSO. We now describe the process of handling a client request with the aid of figure 5.

A WAG does not deal directly with client invocations. Instead, an RSO intercepts an incoming client request and marshals the request into a form suitable for transmission over the Internet Inter-ORB Protocol (IIOP - protocol used for CORBA communications) (figure 5.i). The RSO acts as a proxy for a WAG and uses the same mechanism as the WAG for receiving client requests. As all client requests adhere to the same format (hash table), and all data is text based, the marshalling of a request is easy to achieve via the CORBA data types of *sequence* and *string*. The request manager multicasts (via NewTop) the marshalled request to all replicas (figure 5.i). NewTop ensures that client requests are delivered to all RSOs in the same order. A member of the replica group is identified as the primary. When required to do so (i.e., after group membership changes), the RSO of each member run an agreement protocol to elect a primary. NewTop supports this functionality and uses the Interoperable Object Reference (IOR – object reference scheme used by CORBA) of each RSO in the group to determine the primary and indicates the designated primary to member RSOs. The RSO of the primary unmarshals the client request back into its SOAP form and passes it to the primary replica. The primary then issues requests to appropriate service(s) via HTTP (figure 5.ii). The RSO associated to the primary assumes the role of client, and waits for a reply from the primary. The reply returned to the RSO is multicast to all replicas (figure 5.iii). After NewTop delivers a reply to the RSO layer, the request manager unmarshals the reply and returns the reply to the client (figure 5.iv). The request manager does not necessarily have to be the same as the primary (clients may connect to any WAG replica).

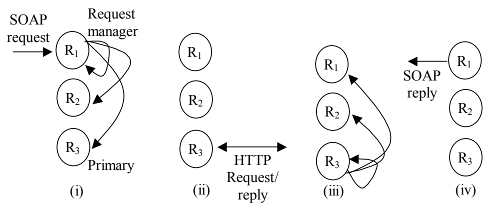

Figure 5 – Handling client requests.

4.2 Handling Failure

For ease of exposition, we shall assume in the rest of the paper that a client request received by a server is always a valid one that needs to be processed. This enables us to concentrate on a server's core task of processing the requests. Failure may occur at any one of the replicas or a remote service provided by an organisation. We assume the underlying communication medium (HTTP over TCP/IP) raises an exception when an attempt is made to submit a request to a failed service. If an exception is raised by the communication medium, indicating that the request was not sent, then an equivalent service may be tried by the primary. If no equivalent service exists (or all equivalent services are unavailable) then the primary informs all replicas of the exception. Such an exception is described in the reply that is sent from the primary to the other replicas (figure 5.iii). The request manager then returns this exception to the client (figure 5.iv).

We assume an organisation's services are accessed via the Internet and that the Quality Of Service (QOS) provided by the Internet is that of a best effort asynchronous network. Given this QOS, there is an inability to distinguish a slow service from a failed service. Consider the scenario where an exception is raised (due to some timeout related to the lack of a reply to a request) and it is not clear if a request has been processed by an organisation's service (i.e., after a request has been sent by the WAG but before a reply has been received). A decision must be made to determine if the use of equivalent services is appropriate. Unfortunately, exploiting equivalent services for a request that may have been processed already may result in the multiple processing the same request. To partially solve this problem, agreement may be reached between all organisations that provide an equivalent service to determine the organisation that will satisfy a client request and exclude those organisations that are considered failed. However, given the domain differences (organisations may be competing against each other to satisfy client requirements) the implementation of such a scheme is unlikely. Another solution would be to use a transaction for the request. If a request fails, the transaction will ensure all operations and procedures are undone, and all data rolled back to its previous state (as if the request was not processed by an organisation's service). The qualities associated with transactions would remove the problem of the same request being processed more than once. Providing end-to-end transactions across organisational boundaries is an ongoing research activity [4] [5]. Therefore, we treat this scenario as if an organisation's service did not receive the request, resulting in an exception eventually passed back to a client (as described previously). This exception indicates to a client that their request may or may not have been processed, leaving the decision of how to handle such an exception to a client.

A composite service may require a primary to issue a number of requests to more than one organisation. The presence of a transaction service would allow related requests to be contained within a single transaction, with the inability to satisfy any one request resulting in the ability to abort related requests without any adverse consequences (failed requests do not cause state changes at an organisation's site). However, as previously mentioned, providing transactions across organisational boundaries is an active area of research and appropriate enabling technologies are unavailable at present. Therefore, the default behaviour of a primary is as follows: if there is an inability to satisfy a single part of a composite request then an exception is raised and returned to the client (as described in figure 5). It is worth noting that the default behaviour of our system may result in exceptions raised even though requests may have been (partially) processed. The exception returned to a client describes any part processing of the request that may have been carried out. We now consider failures of the WAG replicas themselves.

The failure of the request manager will cause the surviving replicas to deliver a view-change message indicating a change in the membership of the replica group, and the binding between the client and the request manager to be broken. The client has to bind with another replica and reissue its request. Consider this scenario further. Assume that the request manager fails as the replicas are multicasting their replies (during the stage depicted in figure 5.iii). The replica group will be reformed with the request manager removed, and no reply will be sent to the client. Client retries can be handled by the new request manager without causing re-execution, provided retries contain the same request number as the original request and servers retain the data of the last reply message (enabling the new request manager to resend the reply). These are 'standard' techniques used in many RPC implementations.

The failure of the primary will cause a group membership change that will result in the remaining, correctly functioning replicas, electing a new primary. Any request delivered to the primary that lacks a reply prior to the installation of the delivery of a view change message (due to primary failure) is considered outstanding. It is not possible to determine if outstanding requests have been processed or not. Therefore, request managers must reply to all clients of outstanding requests indicating that their requests may or may not have been processed due to primary failure. The new primary assumes responsibility for satisfying client requests that were not consumed by the old primary.

5. Performance

Experiments were carried out to determine the performance of our system over a single LAN in failure free, non-replicated scenarios. We consider the figures presented here as initial investigations into the performance of our system, further work is required to identify the performance of replicated services (as described in section 4) over the Internet. We have published comprehensive figures relating to NewTop in both failure free and failure prone environments over LAN and Internet [2] [3] [24].

The system used in our experiments consisted of 10 Pentium III PCs running Windows 2000, each with 128 megabytes of RAM, connected together using 100 Mbit fast Ethernet. Jakarta Tomcat 4.1.12 [25] was used as the application server and Apache SOAP 2.3.1 [26] as the SOAP server.

To enable comparative analysis of the performance figures, web form requests without the use of a WAG were obtained (table 1 column 2). Two types of system configurations were used in the experiments: (1) WAG residing on same machine as web application (table 1 column 3), (2) WAG residing on different machine to web application (table 1 column 4). Each client is hosted on a different node in the network. A single web server satisfied all user requests and was deployed on a different node to clients. Configuration 1 represents the type of environment an organisation might present when enabling its own web form based services for use by clients via SOAP. Configuration 2 represents a more portal type approach, where the WAG is geographically distant from an organisation that is providing services.

Each experiment required a client to issue 10 requests. The time taken to satisfy all 10 requests was measured (from issuing request to receiving reply) with the mean used to identify the timing of a single request. Each request was synchronous in nature and issued in sequence by a client (next request was issued only after a reply was received for prior request). All timings are described in milliseconds and were derived by repeating the experiment throughout the course of a day (to negate variable network traffic associated to our LAN). Table 1 identifies the time taken for a single request to be satisfied given client numbers 1 through 10. The experiments relate to client invocations only, not the creation of services. The web application was a web form that accepts two numbers (supplied in the client request), adds these two numbers, and returns the result as a reply.

Clients	No WAG	local WAG	remote WAG
1	29.7	339.5	441.7
2	35.9	382.5	510.8
3	45.3	457.7	537.8
4	48.3	474.6	596.2
5	59.3	521.0	657.9
6	64.0	642.3	703.1
7	76.6	752.8	785.2
8	86.7	812.1	932.7
9	104.6	862.2	988.4
10	114.1	931.3	1052.5

Table 1 – Satisfying client requests.

A noticeable performance overhead can be seen when the WAG is introduced (local or remote). With only 1 client present, this overhead is approximately 11 times greater for configuration 1 and 15 times greater for configuration 2. However, with an increase in client numbers to 10 these overheads drop to approximately 8 times and 9 times respectively. We can deduce that the processing required by a WAG to validate SOAP requests (including a read from persistent storage when accessing the SSR) and translate such requests to HTTP and replies back to SOAP is substantially greater than handling web form requests. However, in an Internet environment, where message latency may be more appropriately measured in 10s or 100s of milliseconds, the overhead introduced by our system is more acceptable. The smaller percentage increase in overhead when client numbers are increased to 10 can be associated to the ability of a WAG to handle simultaneous client requests in a manner that utilises processing resources more efficiently. Client requests are not queued and handled in sequence, allowing some client requests to be processed and issued while other client requests block until replies are received. An increase in client numbers reduces the percentage overhead of configuration 2 more than configuration1. The only difference between the two configurations is the placement of the WAG (remote in configuration 2 as apposed to local in configuration 1). Configuration 2 should provide the Web application and WAG with greater processing resources over configuration 1. This greater availability of resources may be attributed to the greater drop in percentage overhead of configuration 2 over configuration 1 when client numbers are increased.

6. Concluding Remarks

We have described a portal that allows web forms (typically a business-to-client solution) to be delivered to clients via SOAP (providing the possibility of business-to-business solutions). From a developer's point of view, this process is almost totally automated. Furthermore, tailoring of services to satisfy client requests may be achieved by our portal via the use of equivalent and composite services. To ensure our portal does not present a single point of failure, we propose the use of a passive replication scheme. Tailoring existing CORBA fault-tolerant technologies satisfy our system's replication requirements.

Inadequacies with existing SOAP/HTTP based technologies limit the effectiveness of our system when attempting to provide end-to-end reliability across organisational boundaries. For example, a lack of standard transactional and/or agreement services for Web services suitable for inter-organisational communications makes it unfeasible to ensure client requests could be satisfied once and only once. However, the use of equivalent services may overcome this problem in certain circumstances.

Future work is directed at providing a system that may be more integrated with Web service based technologies. For example, the location and discovery mechanism is not adequate for real world scenarios and the Web Services Definition Language (WSDL) [19] could address the presentation of service interfaces in a more appropriate manner. Additional work is required to fully develop our replication scheme and gain comprehensive performance figures.

7. References

[1] P. Ezhilchelvan, R. Macedo and S. K. Shrivastava, “NewTop: a fault-tolerant group communication protocol”, 15th IEEE Intl. Conf. on Distributed Computing Systems, Vancouver, May 1995, pp. 296-306.

[2] G. Morgan and S. K. Shrivastava, "Implementing Flexible Object Group Invocations in Networked Systems", International Conference on Dependable Systems and Networks, FTCS-30 and DCCA-8, New York, NY, USA, June 25-28, 2000.

[3] G. Morgan, S.K. Shrivastava, P.D. Ezhilchelvan and M.C. Little, “Design and Implementation of a CORBA Fault-tolerant Object Group Service”, Distributed Applications and Interoperable Systems, Ed. Lea Kutvonen, Hartmut Konig, Martti Tienari, Kluwer Academic Publishers, 1999, ISBN 0-7923-8527-6, pp. 361-374.

[4] S. Frølund, R. Guerraoui, “Transactional Exactly-Once”, International Conference on Dependable Systems and Networks, FTCS-30 and DCCA-8, New York, NY, USA, June 25-28, 2000.

[5] S. Frølund, R. Guerraoui, “Implementing e-Transactions with Asynchronous Replication”, IEEE Transactions on Parallel and Distributed Systems, Vol. 12, No. 2, February 2001.

[6] W3C Recommendation, “Extensible Markup Language (XML) 1.0 (Seconf Edition), http://www.w3.org/TR/2000/REC-xml-20001006.html, as viewed October 2002.

[7] Sun Microsystems, Bill Shannon, “Java™ 2 Platform Enterprise Edition Specification, v1.4”, http://java.sun.com/j2ee/j2ee-1_4-pfd-spec.pdf, Proposed Final Draft, August 2002.

[8] S. Lempinen, “jTidy – HTML parser and pretty-printer in Java”, http://lempinen.net/sami/jtidy, as viewed May 2002.

[9] The Apache XML Project, “Xalan - XSLT processor for transforming XML documents into HTML, text, or other XML document types”, http://xml.apache.org/xalan-j, as viewed May 2002.

[10] The World Wide Web Consortium (W3C), “XSL Transformations (XSLT) Version 1.0”, W3C Recommendation 16 November 1999

[11] M. Vering, et al, “The E-Business Workplace: Discovering the Power of Enterprise Portals”, John Wiley & Sons, February 2001.

[12] “Google Web APIs”, http://www.google.com/apis/, as viewed October 2002.

[13] K. Gottschalk etal, “Introduction to Web services architecture”, IBM Systems Journal, Vol 42, No 2, 2002.

[14] M. Cukier et al., “AQuA: an adaptive architecture that provides dependable distributed objects”, Proc. of 17th IEEE Symp. on Reliable Distributred Computing (SRDS'98), West Lafayette, October 1998, pp. 245-253.

[15] Felber, “The CORBA Object Group Service: a Service Approach to Object Groups in CORBA”, PhD thesis, Ecole Polytechnique Federale de Lausanne, 1998.

[16] P. Narasimhan, L, E. Moser and P. M. Melliar-Smith, “Replica consistency of CORBA objects in partitionable distributed systems”, Distributed Systems Eng., 4, 1997, pp. 139-150.

[17] A. Gokhale et al., “Reinventing the Wheel? CORBA vs. Web Services”, WWW2002, The Eleventh International World Wide Web Conference, Honolulu, Hawaii, USA, 7-11 May 2002.

[18] K. Birman, “The process group approach to reliable computing”, CACM , 36, 12, pp. 37-53, December 1993.

[19] The World Wide Web Consortium (W3C), “Web Services Description Language (WSDL) (version 1.1)”, W3C Note 15 March 2001.

[20] The World Wide Web Consortium (W3C), “Simple Object Access Protocol (SOAP) (version 1.1)”, W3C Note 08 May 2000.

[21] Object Management Group, “Fault tolerant CORBA (final adopted specification)”, OMG Technical Committee Document, ptc/2000-04-04, March 2000.

[22] Object Management Group, “The Common Object Request Broker: Architecture and Specification, 2.3 edition”, OMG Technical Committee Document formal/98-12-01, June 1999.

[23] Organization for the Advancement of Structured Information Standards (OASIS), “UDDI Version 3.0 Published Specification”, http://www.uddi.org/specification.html, as viewed October 2002, 19 July 2002.

[24] G. Morgan and P.D. Ezhilchelvan, "Policies for using Replica Groups and their effectiveness over the Internet", Proceedings of the 2nd International COST264 on Networked Group Communication (NGC 2000), Stanford University, Palo Alto, California, USA, 2000.

[25] The Jacarta Project, “Tomcat Document Index”, http://jakarta.apache.org/tomcat/tomcat-4.1-doc/index.html, as viewed October 2002.

[26] The Apache XML Project, “Apache SOAP 2.3.1 Documentation”, http://xml.apache.org/soap/docs/index.html, as viewed October 2002