WS-Replication: A Framework for Highly Available Web Services*

Jorge Salas, Francisco Perez-Sorrosal, Marta Patiño-Martínez, and Ricardo Jiménez-Peris
School of Computer Science
Universidad Politecnica de Madrid
Madrid, Spain
{jsalas,fpsorrosal,mpatino,rjimenez}@fi.upm.es

ABSTRACT

Due to the rapid acceptance of web services and its fast spreading, a number of mission-critical systems will be deployed as web services in next years. The availability of those systems must be guaranteed in case of failures and network disconnections. An example of web services for which availability will be a crucial issue are those belonging to coordination web service infrastructure, such as web services for transactional coordination (e.g., WS-CAF and WS-Transaction). These services should remain available despite site and connectivity failures to enable business interactions on a 24x7 basis. Some of the common techniques for attaining availability consist in the use of a clustering approach. However, in an Internet setting a domain can get partitioned from the network due to a link overload or some other connectivity problems. The unavailability of a coordination service impacts the availability of all the partners in the business process. That is, coordination services are an example of critical components that need higher provisions for availability. In this paper, we address this problem by providing an infrastructure, WS-Replication, for WAN replication of web services. The infrastructure is based on a group communication web service, WS-Multicast, that respects the web service autonomy. The transport of WS-Multicast is based on SOAP and relies exclusively on web service technology for interaction across organizations. We have replicated WS-CAF using our WS-Replication framework and evaluated its performance.

Categories and Subject Descriptors

H.3.5 [**Information Systems**]: Online Information Systems—*Online Information Storage Retrieval, Information Services, Web-Based Services*; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems, Client/server*; C.4 [**Computer Systems Organization**]: Performance of Systems —*fault tolerance, reliability, availability, and serviceability*; H.2.4 [**Information Systems**]: Database Management—*Transaction processing*

General Terms

Reliability, Performance

Keywords

Web services, group communication, availability, transactions, WS-CAF

1. INTRODUCTION

Web service technology is witnessing a rapid acceptance in all fields. If this trend continues, an increasing number of mission critical web services will start to be deployed during the next few years. However, current web service technology still has serious lacks for providing the levels of availability required by mission critical systems. Although there are specifications dealing with reliable message exchange (e.g. WS-Reliability [32]) availability has been largely overlooked.

After some level of maturity of basic web service infrastructure specifications such as SOAP [42], WSDL [43] and UDDI [31], a wide number of specifications are being developed to support more sophisticated applications and providing stronger guarantees. Among these new efforts, a number of them addresses the support for transactional business processes across different business domains [25]. Two recent efforts in this direction are OASIS Web Service Composite Application Framework (WS-CAF) [33] and WS-Coordination, WS-Transaction [26] promoted by IBM, Microsoft and BEA. These specifications provide support for interoperable transactional context propagation and transactional coordination. Transactional coordination requires among other things the appointment of a site as transaction coordinator. The designation of a coordination entity presents an inherent weakness; if the coordinator becomes unavailable the coordination protocol blocks, what has an impact in the availability of all partners participating in the transaction that cannot progress anymore. This means that coordination services become a critical infrastructure requiring higher levels of availability.

Traditional clustering solutions are not enough in an Internet setting in which network partitions are more likely than in LANs (e.g. due to link overloads, site and connectivity) and that can separate the domain where the coordinator entity lies from the rest of the network. In this paper, we address precisely the issue of web service availability and present a framework, WS-Replication, for WAN replication of web services. WS-Replication uses multicast (i.e, group communication [9]) to communicate with the replicas of a web service. One of the main challenges of this work is how to replicate web services preserving the underlying web service autonomy principle avoiding the use of ad hoc mechanisms, such

*This work has been partially supported by the Spanish Science Foundation (MEC) under grant TIN2004-07474-C02-01, the Madrid Regional Research Council (CAM) under grant P-TIC-285-0505, and the European EUREKA/ITEA S4ALL project under grant FIT-340000-2005-144 from the Spanish Ministry of Industry (MITyC).

WWW 2006, May 23–26, 2006, Edinburgh, Scotland.
ACM 1-59593-323-9/06/0005.

as the opening of ad hoc ports at partner sites. For this purpose, we have developed a web service for group communication, WS-Multicast, that uses SOAP as transport protocol. The paper also presents an evaluation of the WS-Replication framework. WS-Replication has been used to replicate an implementation of WS-CAF and has been evaluated through an instrumented version of WS-I.

The contributions of the paper are: the design and implementation of an infrastructure for seamlessly providing high availability of web services in a WAN setting based exclusively on web service technology; and a multicast web service component that can be used independently. These contributions have been thoroughly evaluated through micro-benchmarks and a replicated version of WS-CAF.

The rest of the paper is organized as follows. Section 2 introduces replication and group communication. Section 3 describes the components of the replication framework. A case of study (WS-CAF) is presented in Section 4 and evaluated in Section 5. Finally, related work and conclusions are presented in sections 6 and 7.

2. REPLICATION AND GROUP COMMUNICATION

Replication is the main technique to provide high availability. Availability is achieved by deploying the same service in a set of sites (replicas), so if one site fails, the others can continue providing the service. There are three different process replication techniques: active replication, semi-active replication and passive replication. In active replication [39], all requests to a replicated service are processed by all the replicas. Service requests must be processed in the same order in all the replicas to guarantee that all replicas have the same state. Moreover, replicas must be deterministic. That is, with the same sequence of requests, they act deterministically producing the same output. With this approach, if one replica fails, the rest of the replicas can continue providing service in a transparent way for the client, without loosing any state. Depending on the level of reliability, the client can resume its processing after obtaining the first reply from a replica, a majority of replies or all the replies. If the server code is non-deterministic, it is possible to use semi-active replication [41] in which all sites process all requests, but non-deterministic actions are executed only by a site appointed as master. The master sends the resulting state to the other sites, named followers, after executing a non-deterministic action. A complementary approach is passive replication [10]. In this approach, one of the servers is appointed as primary and the other as backups. Clients submit requests to the primary that processes them and sends the resulting server state to the backups. That is, only the primary executes client requests. Upon failure of the primary, one of the backups takes over as new primary. In this case, failover is not totally transparent from the client point of view. That is, the last request issued by a client may need to be resubmitted (the primary failed before replying) and a duplicate removal mechanism is needed in the backups (the new primary may have already received the state produced by that request in the failed primary).

One of the main building blocks for implementing a replicated system is group communication. Group communication systems (GCS) provide *multicast* and the notion of *view* [9]. A set of processes may join a group (*group members*). A view contains currently connected and active group members. Changes in the composition of a view (*member crash* or *new members*) are eventually delivered to the application. Multicast messages are sent to a group. Multicast primitives can be classified attending to the order guarantees and fault-tolerance provided. *FIFO ordering* delivers all messages sent by a group member in FIFO order. *Total order* ensures that messages are delivered in the same order by all group members. With regard to reliability, *reliable multicast* ensures that all available members deliver the same messages. *Uniform reliable multicast* ensures that a message that is delivered by a member (even if it fails), it will be delivered at all available members.

Replicas of a server form a group in order to implement replication[1]. Failures of replicas are detected when a new view is delivered excluding members of the previous view. Total order multicast is used to propagate requests in the active replication model to guarantee that all replicas deliver requests in the same order and therefore, they will reach the same state. FIFO order is used in passive and semi-active replication to send the state or the result of the non-deterministic actions. Since this information is only sent by one replica, the application of those messages in FIFO order will guarantee that all the replicas reach the same state. Depending on the consistency level needed by the application, multicast messages are either reliable or uniform.

Two important properties of GCSs are *primary component membership* and *strong virtual synchrony* [11]. In a primary component membership, views installed by all members are totally ordered (there are no concurrent views), and for every pair of consecutive views there is at least one member that survives from one view to the next one. Strong virtual synchrony ensures that messages are delivered in the same view they were sent (also called *sending view delivery*) and that two members transiting to a new view have delivered the same set of messages in the previous view (*virtual synchrony*) [11].

Several group communication toolkits have been developed in the last two decades. Isis was the first one and since then, others have been developed i.e., Horus [38], Ensemble [17], Transis [14], Totem [29], NewTop [16], JGroups [20] and Spread [3]. Some of these toolkits are implemented as a stack of micro-protocols. Each micro-protocol is in charge of implementing a reliability, ordering or view property. Horus was the first to adopt this architecture and others followed, such as Ensemble and JGroups.

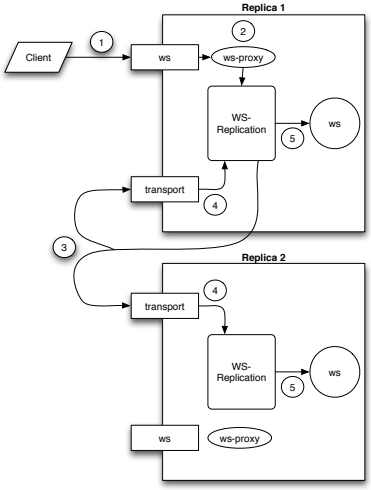


Figure 1: High Level View of WS-Replication

[1]Note that group communication is not required by any replication technique, but it largely simplifies the implementation.

3. WS-REPLICATION: A REPLICATION FRAMEWORK FOR WEB SERVICES

WS-Replication is a framework for seamless active replication of web services, that is, it respects web service autonomy and provides transparent replication and failover. The communication within the framework is exclusively based on SOAP without forcing the use of an alternative communication means. WS-Replication allows the deployment of a replicated web service using web service technology and SOAP for transporting information across sites. What is more, clients invoke a replicated web service in the same way they invoke a non-replicated one. Internally, WS-Replication takes care of deploying the web service in a set of sites, transparently transforming a web service invocation into a multicast message to replicate the invocation at all sites, awaiting for one, a majority or all replies, and delivering a single reply to the client.

Figure 1 depicts the high level vision of WS-Replication. Let us assume that the web service we want to replicate is named *ws*. In the figure, we can see a replicated web service, *ws*, deployed at two replicas represented by large rectangles, using active replication. We distinguish among its WSDL interface (shown as a rectangle with the name of the web service), the proxy (shown as an ellipse) and the *ws* implementation (shown as a circle). The client is local to the replica on the top of the figure[2].

The client invokes a web service *ws* with exactly the interface of the target web service (step 1 in the figure). Internally, the WS-Replication framework intercepts the request through a proxy (step 2) and takes care of reliably multicasting in total order this invocation to all the replicas using a private transport web service (step 3). Upon reception of the multicast message (step 4), the message is processed by the WS-Replication framework to provide the required delivery guarantees (e.g. total order). Finally, the web service is locally executed at each replica (step 5).

WS-Replication consists of two major components: a *web service replication component* and a *reliable multicast component* (WS-Multicast).

3.1 Replication Component

The replication component enables the active replication of a web service. The service to be replicated has to behave as a state machine [39], that is, the service should produce the same output if the same input sequence is provided. The replication component enables the replication of stateful services even with persistent state.

The web service replication component consists of a *web service deployer*, a *proxy generator* for seamless web service replication, and a *web service dispatcher* (WS-Dispatcher).

The deployer provides a distributed deploying facility in the form of a web service that enables the deployment of a service to be replicated at different sites from a central location. It takes a web service bundle as argument that contains all what is necessary to deploy the web service in a single file (like ear or jar bundles for Java applications) and deploys it at all replicas.

The proxy generator generates a proxy for each operation of the web service that intercepts the invocation and interacts with the dispatcher to replicate the invocation. The proxy code carries out two tasks at the site where the invocation is issued: 1) it intercepts invocations to the replicated web service and forwards them to the dispatcher; 2) upon reception of the reply from the dispatcher, it returns it to the client.

[2]If needed, WS-Replication also supports to deploy proxies at sites that do not hold replicas of the web service. This is useful when clients are remote to all replicas.

The web service dispatcher is a component that interfaces with the group communication and takes care of the matching between invocations and messages. More concretely, the dispatcher implements two functionalities. As a sender (the instance collocated with the client), it transforms web service invocations forwarded by the proxy into multicast messages using the underlying multicast infrastructure. As a receiver (collocated with a replica), it reconstructs the web service invocation from the multicast message and invokes the target web service using a particular deployment style. Depending on the required degree of dependability the dispatcher can return control to the client (via the proxy) after receiving the first reply, a majority of replies or all replies. The treatment of replies is indicated in the deployment descriptor of the replicated web service.

There are two types of deployment for a replicated web service: as a private web service or as a Java implementation. The former enables to deploy any web service as a replicated web service, although the price of this flexibility is an extra SOAP message involved in each invocation (the one from the dispatcher to the web service). The latter only works for Java implementations of the web service, but it benefits from saving that SOAP message which is replaced by a Java method invocation, which is significantly cheaper. This (SOAP) Java invocation corresponds to step 5 in Figure 1.

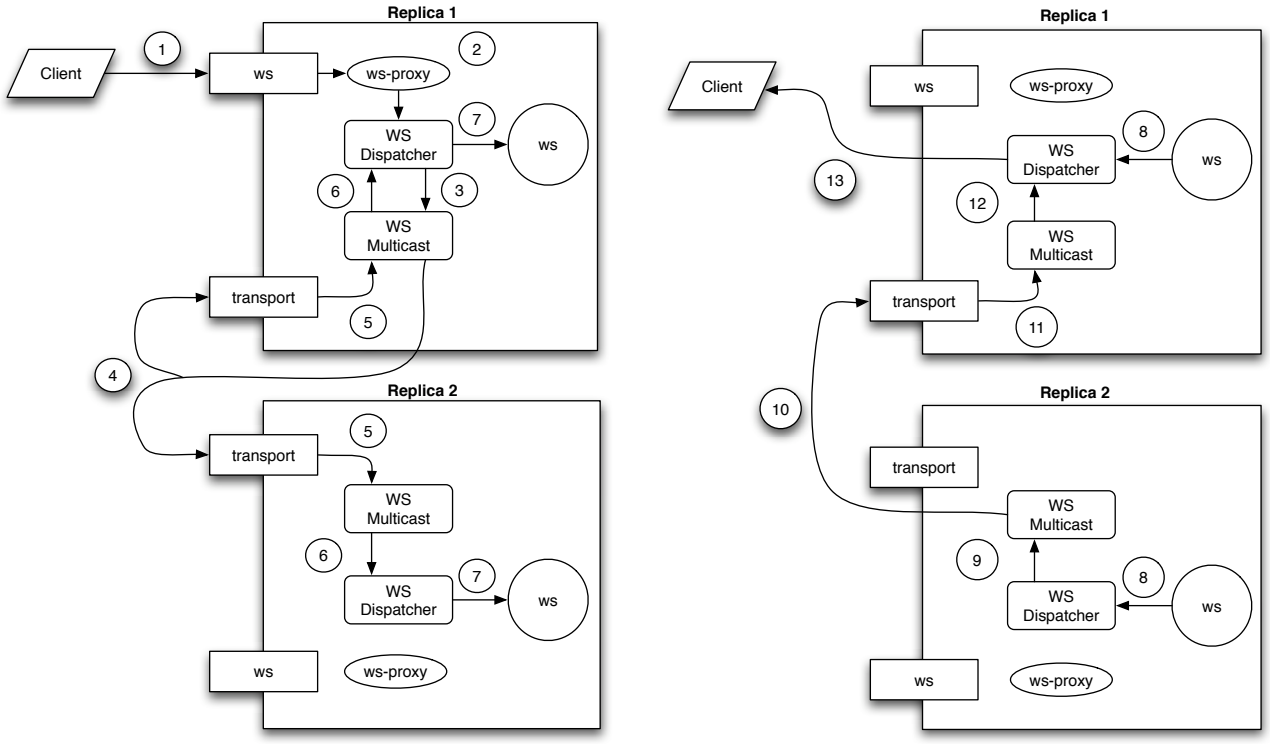
A more detailed view of the path of a replicated web service invocation is shown in Figure 2. The client invokes the replicated web service as a regular web service (step 1 in the figure). The invocation is intercepted by the proxy that delegates it to the dispatcher (step 2). The dispatcher multicasts the invocation through WS-Multicast (step 3). WS-Multicast uses the transport web service to multicast the message to all replicas (step 4). Upon reception of this message (step 5), WS-Multicast enforces the delivery properties of the message (ordering and reliability properties) possibly exchanging other messages between micro-protocols. When the message fulfils its delivery properties, WS-Multicast delivers it to the dispatcher (step 6). The dispatcher executes locally the target web service (step 7). The reply of the local web service invocation is returned to the dispatcher (step 8). The dispatcher where the request was originated will just wait for the specified number of replies (first, majority, all). The dispatchers at other replicas use WS-Multicast (step 9) to send the reply back via unicast to the dispatcher where the request was originated (step 10). When each reply is received (step 11), the local WS-Multicast forwards the reply to the local dispatcher (step 12). The dispatcher will deliver the reply to the client via the proxy once it has compiled the specified number of replies (step 13).

Since all replicas have received the invocation and all replicas are returning the resulting reply, replica failures are masked as far as there is an available replica. That is the replicated service will be available and consistent. The proxy will return to the client as soon as it compiles the required number of replies despite failures. Note that by collocating a proxy with the client the client can access the replicated web service as far as there are enough replicas available.

3.2 Multicast Component

The WS-Multicast component provides group communication based on SOAP. More concretely, it consists of a web service interface for multicasting messages, an equivalent Java interface for multicasting and receiving messages, a reliable multicast stack for group communication, and a SOAP-based transport.

WS-Multicast as a web service exhibits a WSDL interface that is used to attain the reliable multicast functionality using a SOAP transport. It consists of two parts: a user interface and an internal interface. The user interface enables the creation and destruction



(a) Request Path

(b) Reply Path

Figure 2: Low Level View of WS-Replication and WS-Multicast

of groups (via channels that act as group handles) and their configuration. It also allows discovering the members of a group as well as unicasting and multicasting messages. The second interface is used internally by the GCS to disseminate multicast messages, perform failure detection, and other tasks to achieve group communication properties. Failure detection is achieved through a specific SOAP based ping micro-protocol that monitors the group members. A private operation of the WS is a transport operation (shown as `transport` in Figure 2) that takes an array of bytes as parameter, and is used to transfer messages with a SOAP transport.

The group communication stack implements the multicast functionality. In order to reuse some well-known micro-protocols such as those for total order and reliability, our SOAP group communication support has been integrated into an existing group communication stack, JGroups [20]. In this way, WS-Multicast can use micro-protocols of JGroups, and JGroups can be used over SOAP. A similar approach can be used with other stack-based GCSs. WS-Multicast also relies on a SOAP engine to enable web service interaction for transporting messages over SOAP. Our implementation uses Apache Axis SOAP engine [4].

Both the public web service interface and the Java one of WS-Multicast provide the same operations. The Java interface is exactly the same as the one provided in JGroups. In order to send (receive) multicast messages to (from) a group, the sender (receiver) has to create and connect to a channel (the channel is like a socket or file handle). The parameter of the *createChannel* operation is the list of properties to configure the micro-protocols in the group communication stack. To join a group, the group member must connect to a channel providing the name of the group, after connecting to a channel. The *send* operation is used both for multicast and unicast messages. The send operation has three arguments: the message destinations, the source address and the message to be sent. The message destination may be null, then the message is multicast to all members of the group. If it is not null, the message is unicast to that address. The source address indicates the recipients of the response to that message (null indicates the channel address).

The *receive* operation is used to receive messages, views, suspicions and blocks. Internally, messages are stored in a queue. When the receive operation is invoked, the message in the head of the queue is removed and returned. This operation blocks, if there are no messages. The receive operation can return an application message, a view change, a SuspectEvent (indicating that a group member is suspected to be failed), or a BlockEvent (indicates the application to stop sending messages in order to implement strong virtual synchrony).

Finally, the *disconnect* operation removes the invoker from the group membership. The *close* operation destroys a channel instance and the associated protocol stack.

4. A CASE STUDY: REPLICATION OF WS-CAF

Web Services Composite Application Framework (WS-CAF) is a framework for WS coordination at different levels of sophistication, from context sharing to advanced transactional models, such as long running activities (LRAs) [33]. WS-CAF consists of a set of three related specifications: Web Services Context (WS-CTX), Web Services Coordination Framework (WS-CF) and Web Services Transactions (WS-TXM).

WS-CTX defines a standard extensible context structure that enables different business partners to propagate, interpret and extend

this context. The basic operations are creation, update and termination of a context. The basic context stores a context identifier and might be extended to store additional information.

WS-CF extends WS-CTX by defining the coordinator role that takes charge of submitting notification messages to WSs in a particular context. Finally, WS-TXM defines different transaction coordination protocols: two phase commit (2PC) for traditional ACID transactions, long running activities (LRAs) and business process transactions. These transaction coordination protocols aim to agree on a common transactional outcome among the transaction participants. In this paper we will replicate (active replication) and evaluate the performance of LRAs.

The LRA model is designed for business interactions that can expand for long periods of time. An LRA is structured as a set of basic ACID activities. When a basic ACID activity completes its effects are visible. LRAs provide atomicity despite relaxing isolation using compensation. A compensating action logically reverses the effects of an LRA (e.g. a cancellation reverses a booking). LRA activities can be nested. The work performed by LRAs is required to remain compensatable till the enclosing LRA informs that compensation is no longer needed. The coordinator registers the compensator through the LRA protocol before the LRA activity terminates.

The LRA protocol consists of three messages: Complete, Forget and Compensate. Complete is sent as a notification of successful completion. Compensate compensates work that has already been completed by an LRA. The information needed to compensate is kept till a Forget message is received at that time it can be garbage collected. We have implemented WS-CAF (version 0.1). The implementation is available as an open source project at ObjectWeb[3].

In order to evaluate the performance of WS-Replication with a real application we have replicated WS-CAF using active replication. The replicated services are the operations of WS-CTX (the context management services, activity services, and ALS services), WS-CF (coordinator services) and WS-CAF (LRA services).

5. EVALUATION

The evaluation has been performed in two steps. First of all, a micro-benchmark was conducted to understand the overheads of the proposed framework. In a second stage, the replication framework was used to build a replicated WS-CAF that was benchmarked using an implementation of WS-I application enriched with LRAs [45].

The micro-benchmark is based on a simple web service: a web service that implements a counter. It just takes an integer and adds it to a local counter. This simple web service will let us show the overhead of the GCS. We conducted an evaluation that led to a series of improvements in the WS-Replication framework (Section 5.1). Then, we evaluated and compared WS-Replication with two baselines: (1) the non-replicated web service with SOAP transport; (2) the replicated service with group communication based on TCP transport. Baseline (1) measures the cost of a SOAP invocation. It enables to measure the cost of replication with respect a non-replicated web service. Baseline (2) measures the cost of replicating an invocation using group communication based on TCP transport. Therefore, Baseline (2) enables to quantify the cost of introducing a SOAP transport in group communication compared to TCP-based one. This evaluation was run both in a LAN and a WAN (Section 5.2).

In order to show the performance in a realistic application, the aforementioned replicated version of WS-CAF was built based on the WS-Replication framework. This replicated version was used by an implementation of WS-I enriched with the LRA advanced transactional model supported by WS-CAF. The benchmark was run in a WAN (Section 5.3).

Four locations were used for the WAN evaluations: Madrid (Spain), Montreal (Canada), Bologna (Italy), and Zurich (Switzerland). The distances among locations in terms of message latency (in ms) are summarized in Table 2. They have been obtained running 10 times the ping command between each location pair and taking the average among them. The hardware configuration at each site is shown in Table 1. All sites run linux 9.0, Axis 1.1 with JBoss 3.2.3 and PostgreSQL 7.3.2.

Site	# Processors	CPU	Memory
Madrid	Bi-processor	AMD Athlon MP 2 GHz	0.5 GB RAM
Bologna	Quad-processor	Intel Xeon 1.8 GHz	2 GB RAM
Zurich	Bi-processor	Intel Pentium IV 3 GHz	1 GB RAM
Montreal	Bi-processor	Intel Pentium IV 3 GHz	1 GB RAM

Table 1: Hardware configuration at each site

	Madrid	Bologna	Zurich	Montreal
Madrid				
Bologna	30			
Zurich	41	23		
Montreal	130	140	123	

Table 2: Average time (ms) returned by ping between pairs of endpoint locations

5.1 Evolution of WS-Replication

This experiment uses the counter web service to show the overhead of the framework. The experiment was run with a range of 1-3 replicas and a single client in a LAN. Each experiment consists of an overall number of 10,000 requests. There is a single client that submits a request, waits for the reply, and immediately after submits a new request. The results are shown in Figure 3.

When comparing WS-Replication (WS-Replication (v1) in the figure) with the performance of a replicated web service using TCP-based group communication (GC-TCP), it turned out that there was a huge performance degradation even for a single replica. In terms of throughput WS-Replication (v1) behaved 3 to 4 times worse than using TCP transport (GC-TCP), which it is an upper bound on the performance achievable by WS-Replication. In terms of response time, it did relatively worse, from 3 to 5 times fold increase in response time with respect the TCP based group communication. The reason for the overhead was that the serialization performed at the web service level (the SOAP message) for a multicast message (send operation) was generating a 1KB message for an invocation with a single integer parameter, whist a regular invocation with SOAP was generating a 40 bytes message. The send operation of WS-Multicast has the destination of the message, the source and the message itself as parameters. The message contains the name of the invoked operation, the parameters and the corresponding stub class. Even Java serialization was still very inefficient although, more efficient than the one of web services. A non-negligible number of bytes was being generated for a class without any attribute. The solution was to define a streamable interface that enabled the serialization into basic types and a multicast transport web service with a single generic parameter, an array of bytes. The alternative of rewriting the serialization methods was not enough since it did not provide enough control over the generated stream. It serialized

[3]ObjectWeb JASS project. http://forge.objectweb.org/projects/jass/

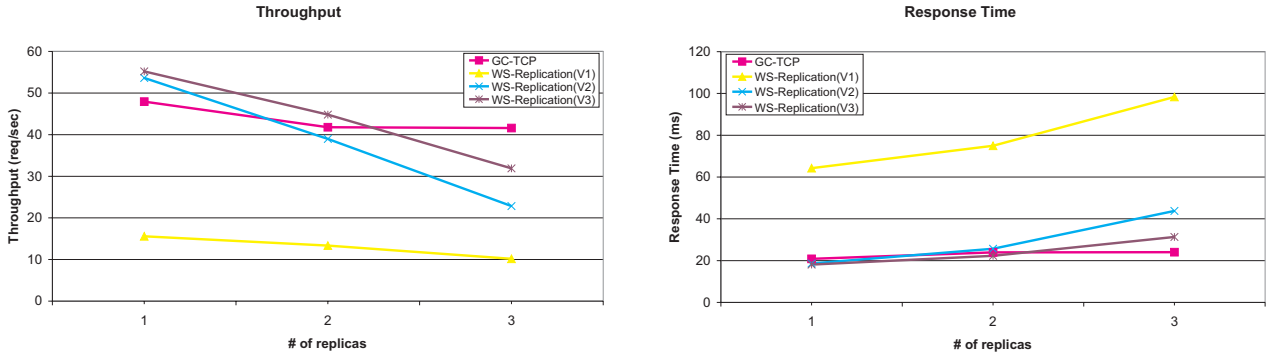


Figure 3: Different implementations of WS-Replication and WS-Multicast in a LAN

class information that was not needed. Note that the serialization improvements are only performed on the private transport web service so they do not affect to WS interoperability. This is just an internal communication means based on SOAP transport for the replicas to communicate among them.

These improvements where evaluated in WS-Replication (v2) in Figure 3. We can observe a substantial improvement, especially for one and two replicas. The throughput for two replicas was very close to the one of TCP-based group communication and increased almost four times compared to WS-Replication (v1). For three replicas, the improvement in throughput although it was significant, it showed a high degradation compared to the one showed by two replicas. The response time of WS-Replication (v2) was significantly reduced to a third of the one of WS-Replication (v1) and almost matched the one of GC-TCP, except for three replicas.

Another source of the overheads was related to the fact that each web service invocation produced three web service invocations: the one made by the client and intercepted by the proxy, one to the transport operation and finally, another one to invoke the web service replica. Since the implementation of the web service is in Java, we used the Java deployment style of WS-Replication. This deployment style enables to forward the invocation to the web service through a plain Java invocation. Of course this performance saving only works for WS Java implementations. WS-Replication (v3) shows the results of this deployment style. It can be observed that the throughput increases for two and three replicas. The response time for three replicas now increases smoothly compared to WS-Replication (v2).

All these improvements (WS-Replication (v3)) prevented the bottlenecks created with 3 replicas in WS-Replication (v2) resulting in a reduction of the response time and an increased throughput.

It should be noted that with one replica WS-Replication (v2) and WS-Replication (v3) behaved better than the TCP-based group communication. The reason is that the TCP implementation creates dynamically the invocations from TCP messages, whilst WS-Replication generates specific stubs upon deployment. This resulted in better performance when there was no replication (1 replica).

It can be concluded that the throughput attained by the final version of WS-Replication was very close to its ceiling, the TCP-based group communication (GC-TCP). In terms of response time, it was almost the same as the one shown using TCP-based group communication. This means that the engineering performed in WS-Replication has been very effective in minimizing the overheads induced by group communication and SOAP.

5.2 Performance of WS-Replication

The experiment in this section compares the performance of WS-Replication with the two baselines previously discussed, a non-replicated web service (SOAP in the graphs), and a replicated web service using group communication based on TCP transport (GC-TCP in the graphs). We show the performance from 1 to 3 replicas for an increasing load for both the TCP-based group communication and WS-Replication. The load is increased by augmenting the number of clients submitting requests. Each client submits a request and as soon as it gets the reply, it submits a new request. The total number of requests sent by each client is $1/n$ of the overall number of requests (10,000), where n is the number of clients. The experiments have been conducted both in a LAN and a WAN setting. The client and the web service replicas were located at different sites. In the WAN setting, the client was in Madrid and the replicas in Bologna, Zurich and Montreal. The replicated web service was configured to wait for the first response.

In a LAN we can see that for one replica WS-Replication performs better than GC-TCP (Figure 4). As explained in the previous section, this is an artifact of the experiment. The GC-TCP implementation has a generic code to manipulate and build web service invocations. This generality results in a lower performance. In contrast, WS-Replication generates specific handling code for each replicated web service that results in a lower serialization and invocation construction overheads that compensate the use of SOAP instead of TCP for one replica. That is, the overhead due to manipulation of serialization and invocations has a higher impact than the overhead introduced by the less efficient SOAP transport. The cost of using group communication and replication is shown comparing the SOAP curve with the 1-replica curves of GC-TCP and WS-Replication.

When looking at the curves for more than one replica in a LAN setting, one interesting observation is that for two replicas WS-Replication is still slightly better than GC-TCP both in terms of response time and throughput, and only for three replicas the higher overhead in the transport layer becomes the dominant factor over the serialization and invocation handling cost.

The results for the WAN are illustrated in Figure 5. In order to quantify the impact of the distance in the throughput and response time we show the performance of a single replica at the three different sites (BOLogna, ZURich and MONtreal). As one would expect, the farthest site from the client (located in Madrid), Montreal, yields the worst throughput and response time. The comparison of the SOAP and 1-Rep-BOL curves shows that the overhead of the

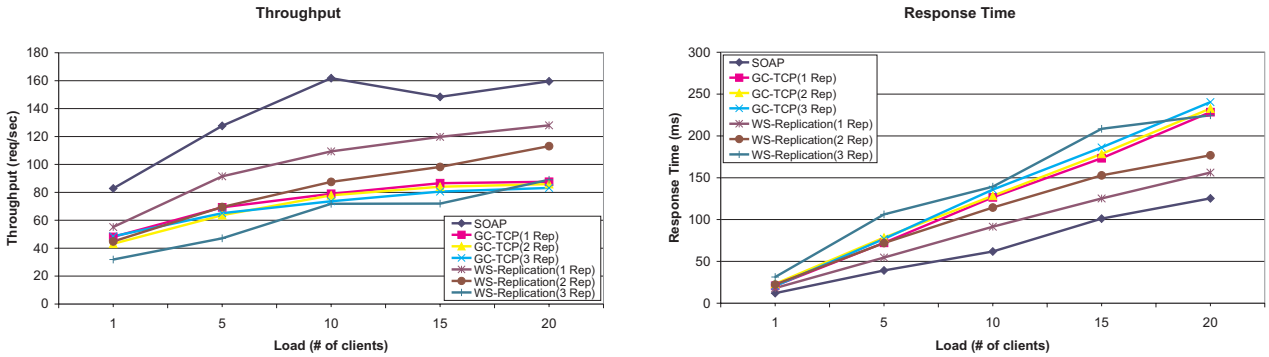


Figure 4: Evaluation of WS-Replication and WS-Multicast in a LAN

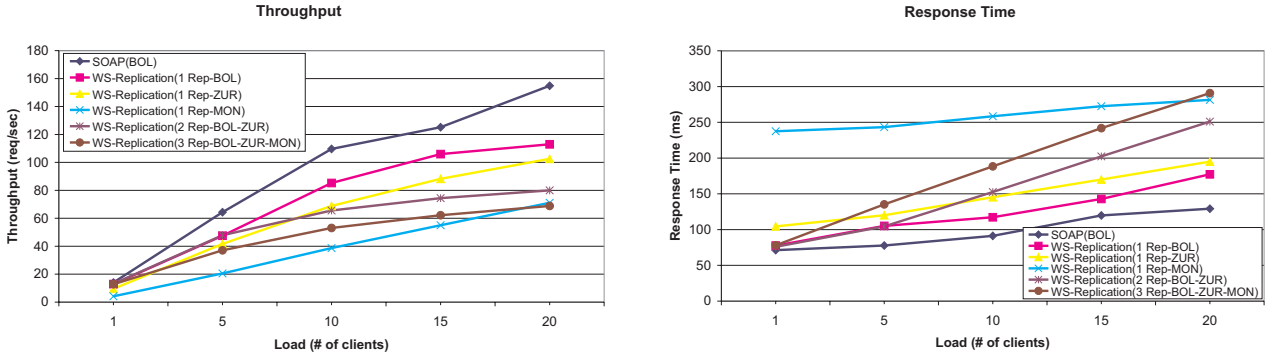


Figure 5: Evaluation of WS-Replication and WS-Multicast in a WAN

replication framework with respect to SOAP is smaller than in a LAN in relative terms (from about a 50% to a 25%) for the closest replica.

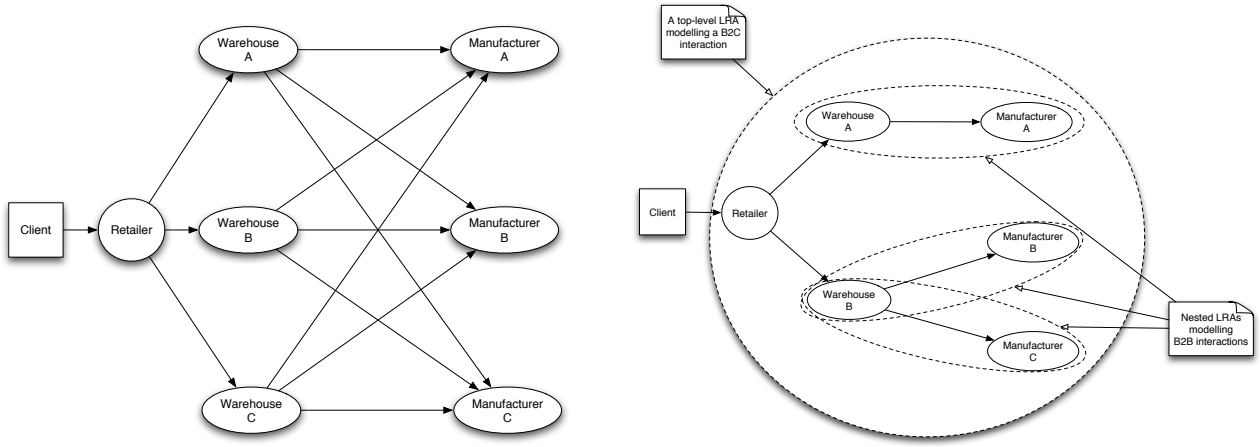
The overhead introduced by the coordination among replicas is shown comparing the 2-replica curve with the 1-replica ones at Bologna and Zurich. The overhead of two replicas in terms of throughput is about a 20%. If we look at the overhead of a 3-replica setting, it can be seen that it has a slightly higher overhead than two replicas, but it behaves better than a single replica at Montreal. The reason is that the experiment was configured to wait only for the first reply. With three replicas, Bologna or Zurich always replied before Montreal. In terms of response time, the triplicated version is significantly better than the 1-replica located at Montreal. This means that clients by using replication will get an average performance respect to the closest replicas, despite the existence of some distant replica.

As a summary we can conclude from the LAN experiment that the engineering performed in WS-Replication has resulted very effective to attain a performance competitive with a TCP-based group communication. The overhead with respect to a non-replicated web service is higher in terms of throughput, due to the extra CPU consumption spent in manipulating extra messages. However, in terms of response time is less significant. The WAN experiment shows that the relative overheads are smaller than in a LAN. What is more, it shows that a triplicated web service will behave better than a single distant replica. An important observation is that this experiment is using an almost null web service, so it measures the pure overhead introduced by extra communication. This means that using a realistic web service with some relevant processing associated to each request (i.e. performing some relevant CPU processing and possibly some IO), this performance loss will be much smaller in relative terms, as we will show in the next experiment.

5.3 Replication of WS-CAF

In this experiment we use the WS- I application to evaluate the performance of a replicated WS-CAF. WS-I is a supply chain management application [45]. It consists of four roles: consumers (clients), retailer, warehouses, and manufacturers. The latter three exhibit a web service interface. A retailer offers goods to consumers. The retailer places orders to warehouses to fulfil consumer requests. Orders should be served completely by a retailer. The warehouses have to keep stock levels for the items they offer. If the stock of an item falls below certain threshold, the warehouse must refill the stock of the item from a manufacturer. Figure 6(a) shows the interactions among the different actors in WS-I.

We have enriched the WS-I application with advanced transactions, more concretely with LRAs, in order to evaluate our WS replication framework. The interaction between the client and the retailer is modeled as a top level LRA that may include several nested activities. Nested LRAs model the interaction between the retailer and a warehouse, or between a warehouse and a manufacturer. Warehouses and manufacturers are registered as participants



(a) Participants and Interactions **(b) Transaction Nesting**

Figure 6: WS-I application

in the active LRA when they include items in an order or when they have ordered or manufactured, respectively. These interactions can be compensated if the client interaction (top-level LRA) is not fully accomplished (i.e. if the warehouses cannot deliver the whole order to the client).

Figure 6(b) shows a potential scenario of LRA nesting in which the retailer contacts warehouse A and warehouse B. Warehouse A needs to contact manufacturer A, whilst warehouse B contacts both manufacturers B and C. The whole client request is a top-level LRA that encompasses three nested LRAs, corresponding to each interaction between warehouses and manufacturers to refill the stock of a particular item. When a nested LRA completes, it registers the corresponding compensators with its parent. This enables compensation in case the top-level LRA does not succeed.

5.3.1 WS-CAF Evaluation

Figure 7 shows the results of the WS-I application evaluating the replicated WS-CAF implementation in a WAN setting. The client and the WS-I application were run in Madrid at two different sites in all the experiments. For comparison purposes we run the same experiment without replicating WS-CAF. In this case WS-CAF was located in Zurich (No Rep curve in Fig.7). Then, we run the experiments with one replica in Bologna (Rep-1), two replicas adding Zurich (Rep-2) and three replicas, adding Montreal (Rep-3). Each client submits 100 requests during which the measurements are done plus 25 warm-up and 25 cold down requests.

It can be noted that the difference between 1-3 replicas is very small in terms of response time, around a 15%. This is mainly due to the stub configuration that is set up to return to the client after receiving the first reply from a replica. Awaiting the first reply is sufficient for many applications and can be considered the default case. In a later experiment we compare the performance of getting the first reply with the one of getting a majority of replies. Comparing the throughput curves for 2-3 replicas with the non-replicated one we can observe that the throughput degradation is smaller for higher loads than for lower loads. The reason is that the non-replicated setting achieves its maximum throughput with 5 clients. After that, the throughput does not increase anymore. The replicated case achieves the maximum throughput with a highest load. Interestingly, the overhead of replication in terms of response time is much smaller in relative terms. The increase in response time between one and three replicas is smaller than a 10% for the higher load. This is quite beneficial because the main concern in WAN replication is response time and the obtained overheads are very affordable.

In Figure 8 we show the performance difference when the replicated web service is configured to wait for a majority of the responses before returning to the client. For two replicas waiting for a majority of replies means to wait for all the replies. As one might expect, the throughput decreases. However, the overhead is not very high especially in the case of three replicas, that have a 13% lower throughput. Regarding response time the relative overhead is slightly higher. This is unavoidable, since one has to wait for the second slowest reply, when waiting for a majority of responses.

6. RELATED WORK

Several specifications have been developed that tackle different aspects of web service dependability. For instance, WS-Reliability [32] deals with quality of service in the delivery SOAP messages. For example, a message is sent at most once, exactly once or at least once. Transactional semantics has been addressed by OASIS WS-CAF [33] and WS-Transaction and WS-Coordination [26]. Basically, they provide a means to describe a transactional context, appoint a coordinator entity for transactions and distributed coordination protocols (either two-phase commit for ACID transactions or more sophisticated coordination for advanced transactions). To the best of our knowledge none of the WS specifications have addressed the availability issue yet. WS-Replication addresses this issue and provides a framework for replication of web services exclusively based on web service technology.

An architecture for web services availability has been presented in [8]. The proposed system does not rely exclusively on web service technology. Instead, it relies on TCP to provide group communication across sites. They provide a (LAN) clustered intermediary message queue for reliable message exchange that is resilient to site failures but not to WAN connectivity failures. WS-Replication provides a more complete infrastructure that deals with highly available web services and deals with both site and connectivity failures.

Replication of a particular web service, UDDI, has been researched in [40]. The paper presents an implementation of a replicated UDDI

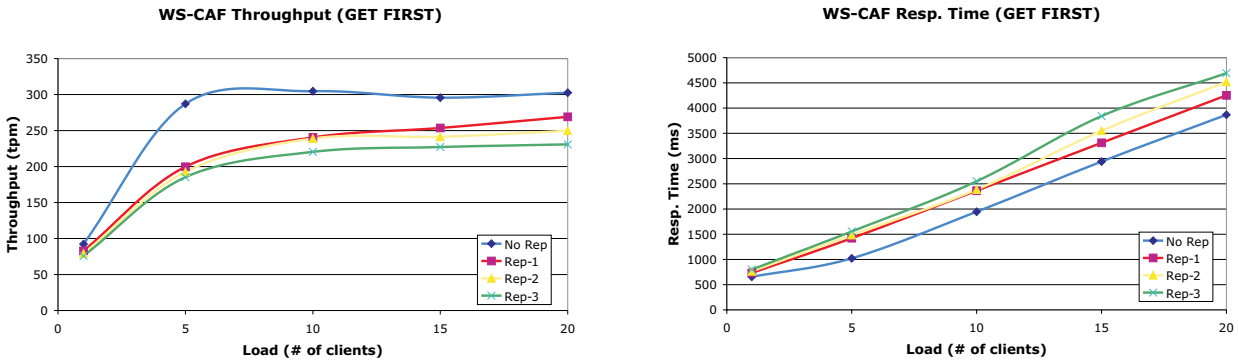


Figure 7: WS-I performance in a WAN waiting for the first response

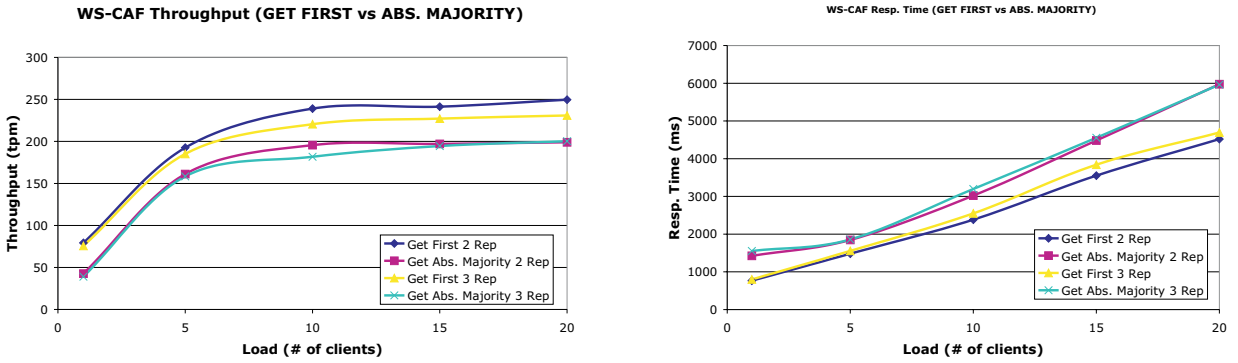


Figure 8: WS-I performance in a WAN comparison of majority and first

registry using TCP-based group communication and compares it against the replicated UDDI specification [30]. There are two differences between this work and ours. The first difference is that it is a TCP-based solution. Thus, they do not rely exclusively on web service technology. However, unlike the previous approach, they deal with connectivity failures since they perform WAN replication as we do. An additional difference is that the reported approach is particular for UDDI and not a generic framework for WAN replication of web services like the one provided by WS-Replication.

The techniques used for replicating web services exhibit some similarities with the replication of other middleware platforms. The first breed of replicated middleware was possibly fault-tolerant CORBA that finally led to the FT-CORBA specification [34]. Different approaches were proposed for replicating CORBA either transparently, as we proposed for web services, or explicitly by exporting some services to build fault-tolerant applications [28, 27, 13, 22, 6]. Some recent work has studied the clustering of application servers [46, 5, 36] in the context of the Adapt project [1, 7].

The replication of transactional applications has also been studied in other contexts such as object oriented systems (i.e., Arjuna [24]) and databases (Postgres-R [21], Middle-R [35, 23], Ganymed [37]). Regarding advanced transactions there was a lot of theoretical work on proposing advanced transaction models [19, 15], how to model their atomicity and isolation properties [12], and their applications, e.g. to workflows [2, 44]. This research has resulted in the current specifications for implementing advanced transactions such as CORBA Activity Service [18] or WS-CAF that we have used as case study for benchmarking WS-Replication.

7. CONCLUSIONS

In this paper, we have presented WS-Replication a replication framework for seamless replication of web services. The framework allows the deployment of a web service in a set of sites to increase its availability. One of the distinguishing features of WS-Replication is that replication is done respecting web service autonomy and exclusively using SOAP to interact across sites. One of the major components of WS-Replication is WS-Multicast that can also be used as a standalone component for reliable multicast in a web service setting. The evaluation shows that with an adequate engineering, the overhead of replication on top of SOAP can become acceptable. We have tested WS-Replication with a complex service, WS-CAF, using a realistic application such as WS-I. We believe that the kind of support provided by WS-Replication will be essential for the upcoming breed of critical web services with more exigent availability requirements.

8. REFERENCES

[1] *Adapt: Middleware Technologies for Adaptive and Composable Distributed Components. IST-37126.* http://adapt.ls.fi.upm.es/adapt.htm.

[2] G. Alonso, D. Agrawal, A. Abbadi, M. Kamath, R. Günthör, and C. Mohan. Advanced transaction models in workflow

contexts. In *Proc. of the IEEE Int. Conf. on Data Engineering (ICDE)*, pages 574–581, 1996.

[3] Y. Amir, C. Danilov, and J. R. Stanton. A low latency, loss tolerant architecture and protocol for wide area group communication. In *DSN*, 2000.

[4] Apache. *Axis SOAP Engine*. http://ws.apache.org/axis/.

[5] O. Babaoglu, A. Bartoli, V. Maverick, S. Patarin, J. Vuckovic, and H. Wu. A Framework for Prototyping J2EE Replication Algorithms. In *Proc. of Int. Symp. on Distributed Objects and Applications (DOA)*, pages 1413–1426, 2004.

[6] R. Baldoni and C. Marchetti. Three-tier replication for ft-corba infrastructures. *SPE*, 33(8):767–797, 2003.

[7] A. Bartoli, R. Jimenez-Peris, B. Kemme, C. Pautasso, S. Patarin, S. Wheater, and S. Woodman. The adapt framework for adaptable and composable web services. *IEEE Distributed Systems On Line*, September 2005.

[8] K. Birman, R. van Renesse, and W. Vogels. Adding High Availability and Autonomic Behavior to Web Services. In *Proc. of Int. Conf. on Software Engineering (ICSE)*, 2004.

[9] K.P. Birman. *Building Secure and Reliable Network Applications*. Prentice Hall, NJ, 1996.

[10] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. Primary–backup protocols: Lower bounds and optimal implementations. In *Proc. of DCCA*, September 1992.

[11] G. V. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: A comprehensive study. *ACM Computer Surveys*, 33(4), 2001.

[12] P. K. Chrysanthis and K. Ramamritham. ACTA: A Framework for Specifying and Reasoning about Transaction Structure and Behavior. In *Proc. of ACM SIGMOD Int. Conf. on Management of Data*, pages 194–203, 1990.

[13] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz. AQuA: An Adaptive Architecture that Provides Dependable Distributed Objects. In *IEEE SRDS'98*.

[14] A.Y. Dolev, D. Krameer, and S. Malki. Transis: A Communication Sub-system for High Availability . In *FTCS-22*, 1992.

[15] A. K. Elmagarmid, Y. Leu, J. G. Mullen, and O. Bukhres. Introduction to Advanced Transaction Models. In *Database Transaction Models*. 1992.

[16] P. D. Ezhilchelvan, R. A. Macêdo, and S. K. Shrivastava. Newtop: A fault-tolerant group communication protocol. In *ICDCS*, pages 296–306, 1995.

[17] M. Hayden. The Ensemble System. Technical Report TR-98-1662, Department of Computer Science. Cornell University, January 1998.

[18] I. Houston, M. C. Little, I. Robinson, S. K. Shrivastava, and S. M. Wheater. The CORBA Activity Service Framework for Supporting Extended Transactions. *Software Practice and Experience*, 33(4):351–373, 2003.

[19] S. Jajodia and L. Kerschberg, editors. *Advanced Transaction Models and Architectures*. Kluwer, 1997.

[20] *JGroups: A Toolkit for Reliable Multicast Communication*. http://www.jgroups.org.

[21] B. Kemme and G. Alonso. Postgres-R, a new way to implement database replication. In *VLDB*, 2000.

[22] M.O. Killijian, J.C. Fabre, J.C. Ruiz-Garcia, and S. Chiba. A Metaobject Protocol for Fault-Tolerant CORBA Applications. In *Proc. of IEEE Symp. On Reliable and Distributed Systems (SRDS)*, 1998.

[23] Y. Lin, B. Kemme, M. Patiño-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Proc. of the ACM Int. Conf. on Management of Data (SIGMOD)*. ACM Press, 2005.

[24] M. C. Little and S. K. Shrivastava. Object Replication in Arjuna. Technical Report 50, BROADCAST Project, 1994.

[25] Mark Little. Models for web services transactions. In *SIGMOD Conf.*, page 872, 2004.

[26] Microsoft, IBM, and BEA. *WS-Coordination/WS-Transaction Specification*, 2005.

[27] G. Morgan, S.K. Shrivastava, P.D. Ezhilchelvan, and M.C. Little. Design and Implementation of a CORBA Fault-tolerant Object Group Service. In *Proc. of DAIS*, 1999.

[28] L. E. Moser, P. M. Melliar-Smith, P. Narasimhan, L. Tewksbury, and V. Kalogeraki. The Eternal System: An Architecture for Enterprise Applications. In *EDOC*, 1999.

[29] L.E. Moser, P.M. Melliar-Smith, D.A. Agarwal, R.K. Budhia, and C.A. Lingley-Papadopoulos. Totem: A Fault-Tolerant Multicast Group Communication System. *Communications of the ACM*, 39(4):54–63, April 1996.

[30] OASIS. *UDDI Replication Specification*.

[31] OASIS. *Universal Description, Discovery and Integration(UDDI)*. http://uddi.org/.

[32] OASIS. *Web Service Reliable Messaging*, 2004.

[33] OASIS. *Web Services Composite Application Framework (WS-CAF)*, 2005.

[34] OMG. *Fault Tolerant CORBA*. OMG, 2000.

[35] M. Patiño-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Middle-r: Consistent database replication at the middleware level. *ACM TOCS*, 2005.

[36] F. Perez, J. Vuckovic, M. Patiño-Martínez, and R. Jiménez-Peris. Highly available long running transactions and activities for j2ee applications. In *Proc. of the IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 2006.

[37] C. Plattner and G. Alonso. Ganymed: Scalable replication for transactional web applications. In *Proc. of the ACM/IFIP/USENIX Int. Middleware Conf.*, 2004.

[38] R. Van Renesse, K.P. Birman, and S. Maffeis. Horus: A Flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, April 1996.

[39] F. B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[40] C. Sun, Y. Lin, and B. Kemme. Comparison of UDDI Registry Replication Strategies. In *ICWS*, 2004.

[41] P. Verissimo, P. Barret, A. Hilborne, L. Rodrigues, and D. Seaton. The Extra Performance Architecture (XPA). In D. Powell, editor, *Delta-4: A Generic Architecture for Dependable Distributed Computing*, pages 211–266. 1991.

[42] W3C. *Simple Object Access Protocol (SOAP) 1.1*. http://www.w3.org/TR/soap/.

[43] W3C. *Web Services Description Language (WSDL)*. http://www.w3.org/TR/wsdl.

[44] D. Worah and A. P. Sheth. Transactions in transactional workflows. In *Advanced Transaction Models and Architectures*, pages 3–34. Kluwer Academic Press, 1997.

[45] WS Interoperability Organization. *Web Service Interoperability (WS-I)*, 2005.

[46] H. Wu and B. Kemme. Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In *SRDS*, 2005.