Influencing the Run-Time Behaviour of Complex Services Using Contexts

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Zachary Harrington

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Service oriented architecture (SOA) and web services make it possible to construct rich and complex distributed systems which operate at internet scales. However, the underlying design principles of SOA can lead to management problems for processes over web services.

This thesis identifies several potential problems with the management of processes over web services, and proposes the use of explicit context as a possible solution. The available options are explored, and the WS-Context specification is implemented and evaluated.

The SOA design principles of loose coupling, interaction at an interface, autonomy, and composablity can lead to management problems for processes over web services. Processes over web services where one composite service invokes other composite services which in turn invoke other composite services can lead to complex invocation trees. These invocation trees may be different at different times due to the shifting effect of loose coupling, as new services are swapped in to replace those in previous invocations. In such an environment how well can we define the interface of the top level service in a static document such as a WSDL? Because there is a separation between the ultimate service consumer, and the ultimate service provider how can the service consumer correctly assign fault when a service fails? When concurrency is used, and encouraged, how can we deal with the inevitable race conditions and deadlock? In distributed systems where portions of processes execute on systems beyond our organizational control, how can we pause, or kill these processes? Many of these systems model long-running business processes. How do we communicate changes in process requirements?

The use of an explicit context is a potential solution to these types of problems. The abstraction context provides an environment in which the process participants can describe their requirements, query those of other process participants, and react to changes in the environment.

A sample context server, based on the WS-Context specification, was implemented using the Erlang language. The sample context server provides the basic operations required to manage and store contextual information about a process.

The sample context server was evaluated to determine the cost of employing a context as part of a web service based software system. The performance of the sample server was also evaluated.

Test were conducted on the time costs of the basic operations of the context server, and they were found to have a constant time cost. The operations for getting and setting the contents of the context were found to have a time cost dependant on the size of the context. The cost of propagating the context along a chain of service invocations was tested and found to have an overhead which increased linearly with the length of the service invocation chain.

The context server was stress tested using a closed loop test which simulated the interaction of

a number of concurrent clients, and an open loop test which simulated bursts of arriving requests. The open loop testing showed that the context server could handle 75 concurrent clients. Beyond 75 concurrent clients, the response times of the context server began to slowly increase. The closed loop testing showed that the context server had a maximum throughput of 190 requests per second for bursts of 200 requests with an interarrival time of 4 milliseconds.

Acknowledgements

I would like to thank my supervisor Ralph Deters for his patience and his assistance. I would also like to thank the members of my committee: Julita Vassileva, John Cooke, and Seok-Bum Ko for their time spent evaluating my thesis. Finally, I would like to thank my wife for her support and encouragement.

Contents

List of Tables

List of Figures

List of Abbreviations

ACID	Atomicity Consistency Isolation Durability
BCCF	Business Collaboration Context Framework
BPEL	Business Process Execution Language
CATWALK	Context-aware Adaptation through Transformations for Web Applications Leveraging Knowledge
COA	Context Oriented Architecture
CWSDL	Context-Based Web Service Description Language
DMTF	Distributed Management Task Force, Inc.
FTP	File Transfer Protocol
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
HTTPS	HTTP Secure
JMS	Java Message Service
LAN	Local Area Network
LOF	List of Figures
LOT	List of Tables
MQ	Message Queueing
OASIS	Organization for the Advancement of Structured Information Standards
OWL-S	Web Ontology Language for Web Services
OWL	Web Ontology Language
P2P	Peer to Peer
PC	Personal Computer
QoS	Quality of Service
REST	Representational State Transfer
RPC	Remote Procedure Call
scp	secure copy
SOAP	Simple Object Access Protocol
SOA	Service Oriented Architecture
SOUPA	Standard Ontology for Ubiquitous and Pervasive Applications
UML	Unified Modelling Language
WS-ACID	Web Services ACID
WS-BP	Web Services Business Process
WS-CF	Web Services Coordination Framework
WS-Context	Web Services Context
WS-LRA	Web Services Long Running Activity
WSDL	Web Services Description Language
WSDM-MOWS	Management of Web Services
WSDM-MUWS	Management Using Web Services
WSDM	Web Services Distributed Management
WSPeer	Web Service Peer

Chapter 1

Introduction

Service Oriented Architecture (SOA) is a paradigm for the development of internet scale software systems. SOA models the world in terms of services. A service is some well defined capability (data storage, data manipulation, or action with some real-world effect) which is offered by a service provider and may be discovered, and utilized by a service consumer.

The appeal of SOA is that it aligns well with the way businesses tend to organize their work, in terms of delegation. Automating a business process therefore becomes a matter of identifying its natural points of delegation and modelling them as services. Services at a higher level delegate to those at a lower level, this continues down to the most basic services which cannot be decomposed further. The entire business process becomes a well defined set of contractual interactions between service providers and service consumers.

Currently, most SOA systems are implemented using web services. Web services are software services which are accessible over a network. Web services may be implemented in different ways (SOAP, REST). SOAP is a high-level XML (Extensible Markup Language) based messaging protocol modeled as envelopes of data, typically exchanged using the HTTP (Hypertext Transfer Protocol). REST (Representational State Transfer) is a lower level messaging scheme which focusses on the use of HTTP to interact with web addressable objects. The focus of this thesis is SOAP based web services. SOAP based web services are defined in WSDL (Web Service Definition Language) documents which describe the services being offered and the format of the messages through which a service consumer may communicate with them. The messages between the service consumer and service provider are SOAP envelopes wrapping an XML document payload delivered by HTTP.

Web services define the method of communication between services, but the services themselves could be implemented using any programming language which includes a web services capability, such as Java, C#, or BPEL (Business Process Execution Language). Quite often web services wrap legacy business systems, allowing them to interface easily with the business' current systems.

1.1 Problem Description

One of the guiding concepts of SOA is that service providers and service consumers interact at an interface. The participants have no understanding of the process in which they execute beyond the interface. This myopic view of the process' execution context is meant to simplify software design, but can complicate application management, and in some ways the applications themselves.

Several problems can arise from such a narrow view of the execution context:

- SOA is a flexible architecture. The loose coupling between service provider and consumer means that swapping service providers is easy (and encouraged) for maintenance, cost, or strategic reasons. In complex applications which cross several levels of abstraction, these shifting dependencies may result in subtle bugs. If it is possible to observe the process as a whole (which services were consumed), it may be possible to work around these problems.

- The document style, asynchronous, routed messaging patterns which typify web service interaction can lead to complex patterns of communication which make understanding what is happening with an application difficult.

- It is difficult for high level consumers to communicate their intentions to low level providers. A consumer must find a provider which meets their functional and non-functional requirements, or they must create their own composite service to meet their needs. Abstraction suffers in this environment. Either providers expose all the functionality of lower level services, leading to complicated interfaces, or they decide on defaults which simplify the interface, forgoing flexibility.

- Things change, especially in business. In a world of long running business transactions, it is likely that the intentions of the participants may change. Current management technologies leave the consumer little recourse once a service has been invoked.

- The very properties of SOA which make it easy to develop business systems, make it difficult to manage them. The management of SOA based web services would benefit from the notion of a process, with an explicit acknowledgement that all the participating services were working towards the same goal, and process level information available to all participants and management applications. This can be accomplished by providing a *context* for the process.

The focus of this research is the use of explicit context with SOAP based web services.

1.2 Motivation

This research is motivated by the possibility that large internet scale systems may be constructed using web services using the SOA approach. It is inevitable, given the tools available that these systems will consist of complex composite services which cross many levels of abstraction and organizational control. While web services and the concepts of SOA dramatically simplify the process of constructing these internet scale systems, the complexity of their message-based runtime behaviour, and the lack of overall process visibility or control, can make attempting to manage these systems difficult or impossible.

1.3 Objectives

The objectives of this research are to explore the use of context in a SOAP based web services environment. An argument will be made for the use of context. An infrastructure for the use of contextual information based on the WS-Context specification will be constructed. The use of context, focusing on the overhead of employing a context will be evaluated. Finally, some potential management uses of the context will be discussed.

Chapter 2

Service Oriented Architecture

In their reference model [1] the OASIS group describe Service Oriented Architecture (SOA) as "a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains."

SOA employs the concept of a service to satisfy the goals/needs of interacting entities. The participants in SOA are the service provider and service consumer. The Service provider makes some capability (data storage, data manipulation, real-world effect) available to be consumed. A service consumer whose needs may be satisfied by that service may then consume it.

The concept of a service aligns well with the way humans tend to organize work; through delegation. As a result, automating an existing workflow involves decomposing it at natural points of delegation which may be modelled by services.

Service providers must make their capabilities visible to potential service consumers. These capabilities may be described in terms of their effects, technical requirements, policies, and the method for accessing the service. The descriptions may be circulated to potential service consumers, or published in centralized queriable registries.

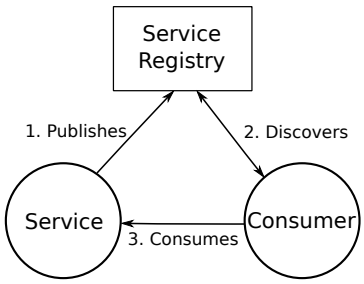


Figure 2.1: 1. A service publishes its information in a service registry. 2. A service consumer queries the registry and discovers the service. 3. The service consumer consumes the service.

The classic illustration of the service lifecycle depicts the publication of a service's information to a queriable third party service registry. A service consumer queries the service registry, and discovers the service, then consumes it. Figure 2.1 illustrates this process.

SOA promotes reuse. Constructing applications by composing well defined capabilities exposed

as services, leaves those capabilities exposed, and available to other, future applications.

SOA promotes flexibility through loose coupling between services. By hiding application logic behind a service interface with well defined capabilities, it becomes easy to substitute an alternative service providing an equivalent capability. In aid of loose coupling, SOA encourages the use of stateless services. If a service consumer relies on a service which maintains some state information, then the consumer looses the option of choosing another equivalent service. Ideally any state information relative to the consumption should be passed as part of the messages between the service consumer and service provider.

The following is a collection of potential problems with the solely consumer-provider view of SOA.

2.1 Brokers

Broker services, or virtual web services are a popular example of a composite service.

A common example of a broker service is one which manages airline ticket reservations[2][3]. If you want to construct airline ticket reservation application using web services, you would design some user interface, most likely a website. On the server side, you would have to consume all the reservation services of the various airlines to present the available trip plans to your customers. But this is something which would be common to many airline ticket reservation systems, so someone may create a service which aggregates all the individual airline services behind a single uniform web service, vastly simplifying your ticket reservation application.

Brokers aggregate the functionality of several similar services, and may or may not add some extra value. At their simplest, they could act as middle men, collecting a commission and passing the request on to another service. But, more typically, they would add value by selecting the service with the fastest, most reliable, cheapest, highest quality, or other desirable attributes, from a set of similar available services.

The Broker Problem

A potential problem arises from this separation of the service consumer and ultimate service provider. It may be the case that some peculiarity of the consumer's request (message, timing) causes a failure in the ultimate service provider. If the service consumer is separated from the ultimate service provider by a broker, the service consumer sees only the failure of the broker. In subsequent transactions, the service consumer may select different brokers, but the newly selected broker may also select the same ultimate provider, resulting in the same failure. This would be experienced as an intermittent failure by the consumer.

It may be argued that some quality of service scheme could be used to register the failures.

But if the failing service performs acceptably for the majority of its other transactions, it may still enjoy a good reputation which results in its continued selection.

This brokering example is just a special case of the problem of separation between the service consumer, and the ultimate service provider. In the case of a process which descends through a tree of composite services, each composite service acts, in effect, as a broker for the requesting service; decomposing its request and consuming other services to satisfy it. Each composite service being another point of choice of which services to consume.

It may be argued that the problem lies with the faulty service provider, and should be addressed there, but from a pragmatic point of view, the interest in securing a successful consumption lies mostly with the consumer. A possible solution would be for the ultimate consuming service (or an agent acting on its behalf) to be aware of potential points of failure, and communicate that to the other participating services in the process.

A Pricing Scenario

A common example for broker services is the case where a broker attempts to get the best price on some service from a set of subservices. If a consumer contacts several services in order to discover the best price for a certain service, some of which are brokers, redundant pricing queries may arrive at same service. This may be interpreted by the service provider as an increased demand for the service which it provides, and it may increase the price it quotes accordingly. Unfortunately, there is ultimately only one interested consumer, and the misquoted price may not be competitive. Figure 2.2 illustrates this scenario with a simple example involving a single broker service.

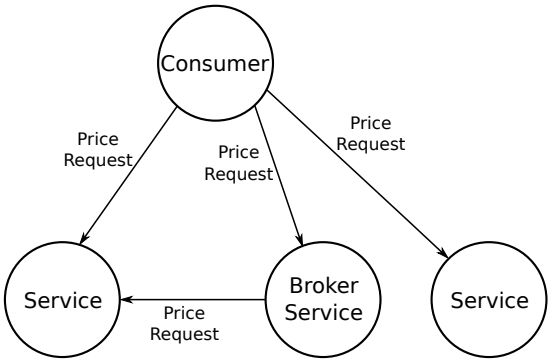


Figure 2.2: A service consumer queries the price of several services. One is a broker service, which results in a redundant price request at the first service.

By making the context of the call explicit, the redundant pricing requests could be observed, and would not be considered when assessing demand.

This pricing scenario may possibly be generalizable to performance tuning scenarios, where system resources are allocated based on some expected demand for a particular service.

Redundant Broker Chaining

Another interesting potential problem with brokers is the case where a broker selects another broker to ultimately perform the service. That a service is a broker may not be immediately evident (possibly advantageously so). The selected broker may also select a broker to provide the service, and so on, unnecessarily slowing the eventual response, or resulting in a potentially infinite chain of unfulfilled requests. Figure 2.3 illustrates this scenario.

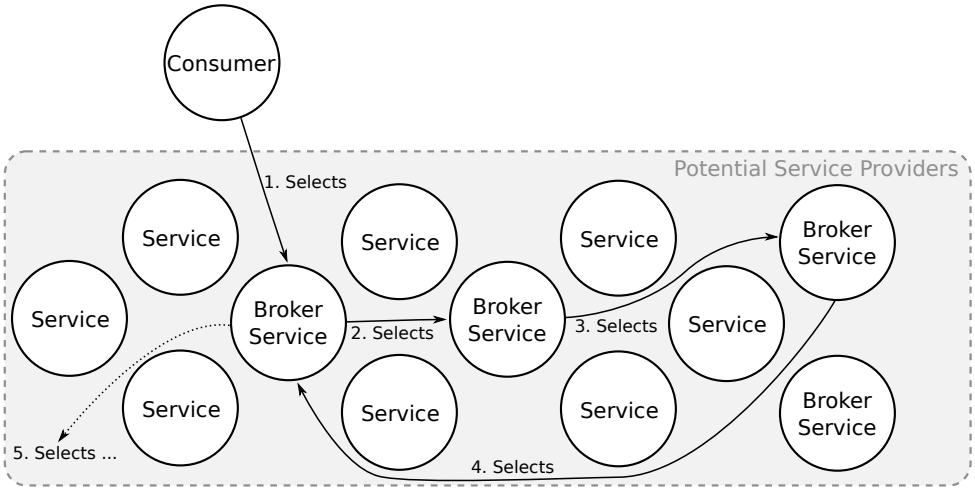


Figure 2.3: A consumer selects a broker which selects a broker, which selects a broker...

Worse than simply slowing the response to this single request, the extra requests which are generated consume network bandwidth, and the waiting service consumers occupy computational resources.

In a service domain saturated with brokers, the probability of this occurrence may be quite high, presuming that the value added by a broker makes it a more desirable selection than a non-broker service.

At some point, hopefully, the consumer will assume that the request has failed and will give up waiting for a response, but that does nothing to recall the in flight request. Perhaps a consumer would notify the provider that it will no longer wait, but this would, at best, chain-on after the initial request, and may or may not catch it. The existence of a context which could be queried to determine the status of the overall process could help in this situation.

2.2 Nonfunctional Requirements

When a service executes, it changes the state of the system/world. Some of the changes are explicitly in the service interface, others are not. If, for example, a service exists which manufactures

a shirt, the resulting shirt is an explicit result of the service consumption. However, where it is manufactured may not be explicitly stated. The materials out of which it is constructed may not be explicitly stated. The carbon foot print may not be explicitly stated. These possible non-functional requirements, and others, may not be taken into account by the service provider.

In an ideal world, a service provider would explicitly state all the effects of the service invocation, but practically, this is not possible. The service provider cannot know what is important to the service consumer, other than in the broadest sense. It could list some obvious effects, in terms of their existence, or nonexistence, but the service consumer would be left guessing about the rest. In the case of composite services, the provider can only pass on as much information as it itself is provided. In the case where the composite service employs some form of dynamic selection, the changing nature of underlying services, can make it impossible to list any but the most obvious, and universal effects with certainty. The more complex and dynamic the service, the less definite it may be in terms of its effects.

Yet, it may be important to a service consumer that their clothing is not manufactured by a particular company (for ethical reasons), or in a particular country (for legal reasons), or from particular materials (for health reasons), or with a limit on its carbon foot print (for environmental reasons). The service provider itself may also be bound by some legal, ethical, or strategic constraints in terms of whether to allow a particular service consumer to consume its service.

It would be nice to have a way to reconcile the nonfunctional requirements of the service consumer/provider with the effects of every service in the process. It seems that separating the negotiation of this functionality from that of the service's core functionality would be best.

2.3 Concurrency

There are many Web service and SOA examples which promote the idea of concurrency. This opens up a host of potential opportunities, as well as problems. There are the classical problems of race conditions and deadlock between competing threads of execution in the same process. More subtly, different threads of execution, may compete for system resources, such as network bandwidth, storage, or access to particular to a particular set of underlying service providers, which may result in intermittent slowdowns or timeouts.

Ideally, a process' threads of execution could be in some way coordinated, so as not to compete with each other.

2.4 Process Control

For management purposes, it may be desirable to pause, resume, and end processes. It may be desirable to know how complete a process is, or get an estimated time of completion.

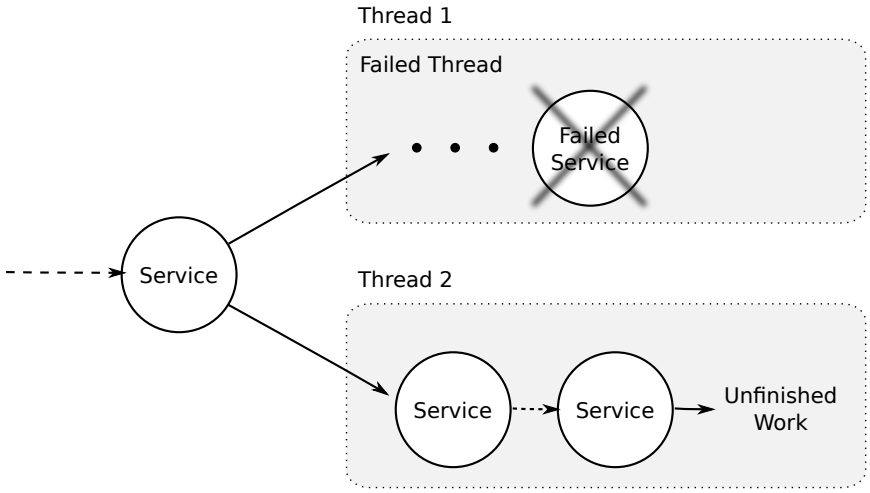


Figure 2.4: If one thread of a concurrent web service process fails, the other threads, unaware, may continue to wasteful conclusions.

In a process over some composite services, which employs multiple threads of execution, a failure may occur on a particular thread which may make the completion of the other threads unnecessary. In such a scenario, it would be advantageous to halt all threads of execution, rather than letting them run to a possibly wasteful conclusion. This scenario is illustrated in figure 2.4.

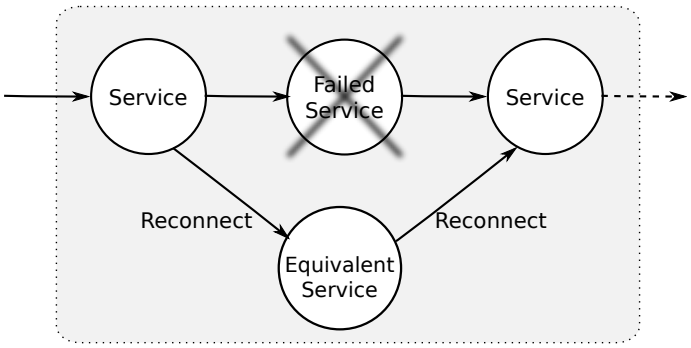


Figure 2.5: If an intermediate service fails, the portion of the process beyond the failure is lost. Would it be possible to reconnect it through a new equivalent service?

In a process over some chain of services, some intermediate service may fail. The work being completed on its behalf may still be relevant to the successful completion of the overall process, but because of the intermediate service failure, this work will be abandoned. It would be nice if it were possible to replace the failed service with an equivalent service, and reconnect it to the services which the failed service had consumed. This scenario is illustrated in figure 2.5.

2.5 Changing Requirements

In a long running process, for example a process which modelled the building of a house, the requirements of participating entities may change. The service consumer may need to change the number of windows, the door manufacturer may need more time to complete the doors.

2.6 Summary

The previous observations illustrate that in some cases, the SOA ideal of systems constructed of stateless, loosely coupled services, can lead to problems.

The problems outlined all deal with the problem of the services, or agents acting on their behalf, being aware of the other services in the process, and being able to communicate with them. Providing a context for the process, to share information related to the successful completion of the process could be a possible solution.

Chapter 3

Background

3.1 Current Technology

3.1.1 Web Services

Web services are currently the most popular technology for realizing SOA (Service Oriented Architecture). A service may be defined as some capability made available for use by another. Web services are software services which are network accessible. The capabilities of a web service are described in a WSDL (Web Services Description Language) [4] document.

The web service consumer and the web service provider typically communicate by exchanging XML messages wrapped in the SOAP messaging protocol over an HTTP (Hypertext Transfer Protocol) connection.

A UDDI (Universal Description Discovery and Integration) [5] server may be used to store the WSDL documents of deployed web services. The UDDI server provides a method of locating available services.

3.1.2 Communication Styles

Call Style

Web services can communicate synchronously or asynchronously. In a synchronous interaction, the service consumer blocks waiting for the service provider to return a result. This is typical of traditional programming style. In an asynchronous interaction, the service consumer does not block waiting for service provider's response, but may register a callback function to handle the response, or provide a callback operation. This is typical of event driven programming, and implies some level of concurrency in the service consumer.

Encoding Style

In SOAP, the messages can be either Remote Procedure Call (RPC) or document styles. In RPC style the consumed service is thought of as a function, which consists of a list of ordered parameters,

and returns a return value. In document style encoding, the consumed service can be thought to be digesting a document, and producing a document in response.

Transports

The web service messages may be transferred using various transport mechanisms such as: HTTP, SMTP, JMS, MQ, or FTP. While there are many transport options, HTTP and HTTPS, are the de facto standards for SOAP web services.

Message Exchange Patterns

"A message exchange pattern is a template that establishes a pattern of messages between communicating parties." [6] The WSDL 2.0 proposal defines the following MEPs (Message Exchange Patterns): In-Only, Robust In-Only, and In-Out which can be used to describe the way in which a service exchanges messages. An In-Only MEP indicates that the operation accepts a single message and does not return a reply. The robust In-Only MEP is the same as In-Only with the addition that a fault message may be sent instead. The In-Out MEP is the standard request response stye of messaging.

Philosophy

Some see web services as distributed objects, some as RPCs, some as agents, and others as something completely different. The type of communication experienced between services will be different based upon which idea is subscribed to, or what type of underlying legacy system is being exposed.

The various call styles, encoding styles, transports, MEPs, and underlying philosophies mean a potentially chaotic system which is difficult to monitor or understand. Even the emerging standard use of document style encoding and asynchronous communication opens the door to complex patterns of interaction beyond the simple provider-consumer model. Documents may be altered, routed, or stored along the way, before they are eventually processed. The consumer may receive a document in response from an entirely different service than the one it consumed.

3.1.3 Composite Services

Composite services are services which aggregate the functionality of several other services and present it as a new service. Composite services may be as simple as offering a single interface to several equivalent services, or combining several different services with additional business logic to create a value added service.

There are many ways to construct composite web services. It is possible to create a composite web service using traditional languages such as java, C++, or C#, but there is a recognition in the community that the stringing together of the coarse grained functionality which services provide

requires a more appropriate language which simplifies issues such as parallelism and asynchronous service consumption.

Currently, BPEL (Business Process Execution Language) [7] is the most popular standard for defining composite services. BPEL is a workflow language, which is popular for modelling business workflows as service.

Composite services provide the layers of abstraction which make services so easy to use. Once a company has defined a workflow, that business value can be encapsulated as a composite service and offered internally and/or externally at a higher level of abstraction.

3.1.4 Enterprise Service Bus

An Enterprise Service Bus (ESB) is an abstract messaging backbone for connecting web services. The main benefit of an ESB is to further decouple the system, by acting as a broker and common point of communication. As a common point of communication, the problem of changing which service to consume moves from being a coding task in the program consuming the service, to being a configuration task in the ESB. As a broker, the ESB can route messages to the most appropriate service, based on rules, or the content of the message, or QoS data.

The ESB also acts as a translator between services implementing different messaging protocols. It provides standard wrappers for various communication/middleware protocols (for example: SOAP, REST, JMS, MQ, and many others). This saves service providers from having to write web service interfaces to standard legacy systems. With some configuration, they can be offered directly by the ESB.

ESBs can provide management capabilities such as: monitoring (system resources, services), auditing the flow of messages in the system, diagnosing performance problem, and start and stop services[8].

In a way, the ESB can be used to establish a context for a process. If the entire process uses the ESB, it can inspect the messages between services to infer the structure of the process. It can then alter the routing of messages, or their content manage the process. There is however no explicit idea of a context, and all messages must use the ESB to benefit.

3.2 Contexts

A *context* may be defined as "a set of constraints that influence the behavior of a system (a user of computer) embeded in a given task" [9].

There has been some work in the use of contexts with web services. This work covers several areas:

3.2.1 Standards

WS-Context

The Organization for the Advancement of Structured Information Standards (OASIS) (`http://www.oasis-open.org`) released the WS-Context specification [10] as part of the WS-CAF (Web Services Composite Application Framework)[11] family of specifications [12] whose purpose is to provide a framework to support multiple services used in combination. WS-Context forms the general foundation for the other WS-CAF specifications: WS-BP, WS-CF, WS-LRA, and WS-ACID which are seen as specific contexts of an activity's operation.

In the WS-Context specification, a context is a container for storing information about the execution environment of one particular activity, where an activity is "a series of related interactions with a set of web services" [10]. The context's information is encoded as an xml document. The context can be embedded within SOAP messages exchanged between services, or stored in an external context service.

The context service provides only the *getContents* and *setContexts* operations which get or set the entire context. It is mentioned that the context service may return a subset of context where security warranted. Also, it is mentioned that should the activity require co-ordination between web services operating on the context, that should be externally implemented.

WS-Context defines two participants in a contextual interaction: the *ContextService* and the *UserContextService*, and outlines interfaces which each must implement. The interface covers, creating a context/beginning an activity (the *begin* operation), destroying a context/forcing completion of an activity (the *complete* operation), polling the status of the activity, and retrieving and setting timeouts, and retrieving and setting the content of a context.

3.2.2 Standards Built on WS-Context

WS-CF Web Services Coordination Framework

The WS-CF standard [13] is part of the WS-CAF set of standards, and is build on top of WS-Context. WS-CF is used to register a service "as a participant in some domain specific function"[13]. The standard gives examples such as registering to a publish-subscribe topic to receive asynchronous messages. WS-Context's context type is extended to create a registration context type by adding a web service reference. Web services participating in an activity can be registered in the context creating an activity group. The registration can be the basis for some form of communication between participants. The standard mentions "work flow, atomic transactions, caching, and replication security, auctioning"[13] as possible applications. Again, WS-CF is meant to be a building block for higher level standards such as WS-ACID, WS-LRA, and WS-BP.

WS-ACID Web Services ACID

WS-ACID[14] is a standard for managing atomic transactions. "WS-ACID defines a pluggable transaction protocol that can be used with the coordinator to negotiate a set of actions for all participants to execute based on the outcome of a series of related Web services executions"[14]. This can be used to accomplish various types of transaction protocols, such as two-phase commits. WS-ACID assumes that all services in the activity have native ACID semantics, and that activities complete quickly. The transaction participants must be part of a WS-CF activity group. The WS-Context begin operation triggers the creation of a transaction coordinator, and the complete operations triggers the commit or rollback of the activity.

WS-LRA Web Services Long Running Activity

WS-LRA[15] is a transaction scheme for long running business processes, which would strain traditional transaction models. Unsuccessful activities employ a compensator to undo the work they have done. Not all services must be compensatable. It is a deployment decision. It may be the case that compensation must be undertaken outside the application, such as an alert to a system administrator.

WS-BP Web Services Business Process

WS-BP[16] is a transaction model in which business processes operate over business domains. "Business process transactions are responsible for managing interactions between these domains". A business process has a manager which is informed of its success or failure, or may query the task to determine its status. The business process will either succeed or fail in which case its work will be undone. Any work which cannot be undone will be logged to be handled offline. The business process model is optimistic, assuming that only a small number of failures will occur, which can be handled offline.

3.2.3 Criticism of WS-Context

There are three basic options for creating a context service: Create your own, use a common context server, or extend and implement some basic server standard which has a common subset of the required functionality. WS-Context has chosen the third option. Any attempt to prescribe the functionality of a context service may limit its usefulness, but if everyone implements their own, there will be much duplication of work, and inconsistencies between approaches to the problem. A middle ground seems like a good solution, but it may be the worst of both worlds.

There may be an independent need for a context for different reasons, for example to communicate nonfunctional requirements, but also to record monitoring information. In an everyone

implements their own context service world, these would be independent context services. In a standard server example, these could be separate pieces of data within a single context.

The Context Manager

Getting and setting the whole context at a time seems too blunt an approach. This method of access seems to have been added as an afterthought. The standard seems to favour simply passing the context along, inside the SOAP header, of messages between participants. The *getContext* and *setContext* operations rely on the participants which have retrieved the content of the context to put back the existing content, and only alter the portion of the context which is of interest to them. It also seems that searching for and obtaining only the elements of interest within the context would be more appropriate. In this respect, WS-RF seems to have a much richer set of operations.

Since the context manager can be separate from the context service, the context manager must check with the context service to ensure that the context being accessed is valid. Since there is the possibility of of a timeout for the context, a separate context manager may process a request after the context has timed out, almost a race condition.

If different parts of a process interact with the context manager concurrently, there is the possibility of lost edits as participants' updates clobber one another. Some sort of locking mechanism seems to be appropriate.

As it stands, with the getContents operation, participants must poll the context manager to discover changes in a context of interest. It seems as though this would be such an obvious use case, that some sort of notification system would be of great value.

Overall, the WS-Context specification seems to have been abstracted out of other work in the wider WS-CAF set of specifications. It is a good start at a notion of a process context, but it could use some extension.

The Context Service

The structure of the context is maintained in the context service, but there is no way to access it by the participants. It can be communicated by storing it separately as information in the contents of the context, but this relies on the participants to propagate this information correctly. It seems that this would be better handled by allowing the participants to query the structure of the context directly from the context service.

3.2.4 WS-RF

WS-RF (Web Services Resource Framework `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`) is a standard for dealing with stateful services. While SOA encourages stateless services, there are many real world instances where state is unavoidable. An example would be a

service which exposes a physical resource, which is unique, and cannot be accessed alternatively through another service such as a physical printer, or an order with a particular retailer. In these cases it is necessary to be able to identify the unique resource in the messages to the service. Since resources have common usage patterns such as getting and setting properties of the resource, WS-RF, provides a standard way to do this.

While WS-RF does not define the concept of a context explicitly, it is mainly used to maintain sessional data. Contextual data could be attached using the ReferenceProperty element. [17]

3.2.5 Service Selection

Mostéfaoui et al. [18] demonstrate an extension to WSDL which marks certain operations as providing contextual information. Services providing these functions are polled and the responses are used to select the service to invoke.

Chen et al. [19] use a semantic matchmaker which attempts to discover a suitable service based on the requested functionality, and contextual data supplied by the requester.

3.2.6 Frameworks and Architectures

Keidl and Kemper [20] developed a framework to support contextual information about the web service client which would provide personalized web services.

Elsafty et al. [21] introduce an architecture for contextual semantic web services. The architecture focusses on agreeing on a set of sensors which will develop the context by sensing the client and server environments. A monitor observes the sensors, and triggers a response to a defined change in the context.

Chen et al. [19] introduce CA-SOA (Context Aware Service Oriented Architecture) which utilizes an agent based broker system for service selection based on contextual data.

3.2.7 Customization

Keidl and Kemper [20] developed a framework to support contextual information about the web service client which would provide personalized web services. They focus on the client device characteristics and user data such as preferences and current location. The framework pre- and post-processes web service messages based on contextual information, so individual web services need not be context aware, but may access the context blocks directly through an API. The contextual information is transmitted from the client to the web service in the soap header block. Each context may contain multiple context blocks each of a particular type, for example *location* or *client device*. Context information is utilized either explicitly by the web service, or by automatically through pre- and post-processing of the web service messages by plugin modules. The context may be

altered by the invoked web service, and is propagated to any web services which it invokes. The returned context may be integrated into the existing context which is returned to the client. The framework includes the facility to direct the processing of contexts. Which context plugin/service is used to process a context block may be specified in context block in the SOAP header, advertised on a UDDI server, or selected dynamically (context service). Which host processes the service may also be specified either *next* (the next web service which no longer propagates the context block) or *all* (every web service to which this context is propagated).

Matsummura et al. [22] transmit data about available communication protocols as contextual data which the service can use to select a higher performance protocol for future communications.

3.2.8 Session Data

In any system where an activity requires multiple interactions between distinct participants, it may be necessary to remember what occurred during previous interactions. An example might be using a web browser for online shopping. Each page requested from the web server is a separate interaction. It is necessary for the web server to remember what you have already added to your shopping cart. This can be accomplished by maintaining some small amount of data on the server, usually called a session, and providing the user with a some unique identifier. The identifier, encoded in the resulting web page is returned to the server as part of the next web page request.

The interactions between participants in a process over web services is no different. Some interactions take on a conversational style with multiple invocations of the operations of a distinct service. The WS-RF[23] standard seeks to support and standardize these types of interactions.

In other cases, a service consumer might favour a particular service provider, and return to consume it again and again. In such cases remembering something about the previous interactions may be of benefit. Information may be stored in order to improve performance such as Matsummura et al.'s [22] investigation of the storage of static (unchanged between invocations) operation parameters as contextual data.

It is possible to avoid storing the session data directly by the participants, by encoding it in the messages which they exchange. This avoids the problem of tight coupling between the participants, as the session could be continued between any other functionally equivalent pair. There are some potential problems with this approach. The message may be lost or corrupted in transit. One, or both, participants may alter the sessional data in the message to their advantage.

Other research in this area includes Elsafty et al. [21], who look at maintaining a context repository on both the client and server side to cache the context over multiple client server interactions.

3.2.9 Acquiring Contextual Information

There is an area of context research which focuses on the environment of the end user. In this case, the properties of the end user and their environment must populate the context which can then be used in dynamic service selection, or passed along during any service consumption to further customize the service.

Acquiring the context information can be done through the use of some form of sensor scheme[21]. The properties of the user's environment could be their location (acquired by GPS), the properties of the device they are using to access the service, the contents of their electronic calendar, or the responses to a survey [19].

3.2.10 Encoding Context Data

The information stored in a context must be in some way interpretable to the participants utilizing that context. The WS-Context specification uses XML for the encoding the context content. The Chen et al. [19] developed requester and service context ontologies to encode the contextual information. Unfortunately, the ontologies are limited to location, social calendaring, and device network settings.

3.2.11 Co-ordination

The WS-CAF specification is intimately related to coordination. The WS-CF standard introduces the use of registration to group participants in an activity. This grouping of related participants forms the basis for coordination between participants. The WS-ACID, WS-LRS, and WS-BP specifications provide different levels of transaction management to ensure some level of process consistency.

3.3 Management of Web Services

The major focus of management activities in the web service world has been providing standardized web service interfaces to manageable resources, and the management of web services as resource.

The OASIS Group has published the WSDM[24] (Web Services Distributed Management) standards, which include the WSDM-MUWS (Management Using Web Services) [25] [26] and WSDM-MOWS (Management of Web Services) [27] standards.

DMTF (Distributed Management Task Force, Inc.) has published the WS-Management specification [28].

At the workflow level, many BPEL (Business Process Execution Language) engines (Active-BPEL, Apache Ode, Oracle) provide limited management capabilities.

Apache Ode provides a Java management API which is exposed as web services. The management API consists of two interfaces *ProcessManagement* and *InstanceManagement*.

There seems to be no concept of management at an entire process level. This is not surprising, as it may be difficult to decide where a process begins or ends, and who has the responsibility for managing it; especially since services often cross organizational and intra-organizational boundaries. Never the less, situations may arise because of the natures of SOA and web services in which it would be desirable to have some level of process control.

3.4 Co-ordination of Web Services

Coordination of web services deals with managing the interactions of participants in a process to achieve some goal. This area covers ideas such as transaction management, service choreography, and service orchestration.

Transaction Management

There has been much work on transaction management of processes over web services. WS-CAF (section 3.2.11) and WS-TX [29] are two such specification groups.

WS-CAF (WS-ACID and WS-LRA, and WS-BP were discussed previously in section 3.2.11).

WS-TX [29] (WS-Coordination [30] and WS-AtomicTransaction [31] and WS-BusinessActivity [32]) provide a basis for coordinating participants in an activity. It is used as the basis for Transaction management.

Choreography

Choreography is a method of coordinating the communication between the web services in some set. The choreography defines the protocol of message exchanges between the interacting web services. The WS Choreography Model [33] describes this model of communication. WS-CDL (Web Services Choreography Description Language) [34] is a specification for describing choreographed communications.

Orchestration

Orchestration is another, more familiar, method of coordinating web services. "Orchestration refers to an executable business process that can interact with both internal and external Web services." [35]. BPEL (Business Process Execution Language) [36] uses orchestration to coordinate the execution of a process over a set of services.

WSCI Web Service Choreography, WSFT Web services transaction framework, Web services transaction WS-TX, BPEL.

3.4.1 Table of Papers

The following table summarizes the work on the use of context with web services. The summary is presented by area of interest with references to the papers and notes on their contributions.

Area	Papers	Notes
Acquiring Context	Pokraev et al. [37]	Context acquired dynamically from context services.
	Sashima et al. [38]	Context from context services which sense environment at runtime.
	Mostefaoui et al. [18]	Discovers context by running client side utilities described in CWSDL.
	Elsafty et al. [21]	Context is acquired using sensors (vague).
	Blake et al. [39]	Context inferred by agents, which observe user.
Mobile User Context	Pokraev et al. [37]	Personalized voice and data, point of interest selection.
	Sheng and Bentallah [40]	Users expect awareness of personal environment.
Context Based Service Selection	Pokraev et al. [37]	Location based service selection.
	Riaz et al. [41]	Uses narrowing approach.
	Sashima et al. [38]	Location/Capability/Requirements based service selection.
	Mostefaoui et al. [18]	Elects service based on client computed context.
	Elsafty et al. [21]	Service selected negotiator based on sensor data (context).
	Blake et al. [39]	Agents proactive select and consume services based on context.
	Chen et al. [19]	Agents assist the requester and provider with semantic matching.
Context Encoding	Keidl and Kemper [20]	Context Ids, context types/blocks, SOAP Header.
	Kaltz and Zeigler [42][43]	Context represented in ontologies (OWL[44]).
	Elsafty et al. [21]	Uses OWL-S[45] ontology with extensions and SOUPA[46] ontology.

Area	Papers	Notes
Changing Context	Pokraev et al. [37]	Notification of context changes.
	Elsafty et al. [21]	Context changes monitored by sensors.
Formalizations and Standards	Sheng and Bentallah [40]	ContextUML: describe context, generate software.
	Mostefaoui et al. [18]	CWSDL Language for describing context in terms of client computable functions.
	Little et al. [10]	WS-Context standard.
Constraints	Sheng and Bentallah [40]	Use context constraints to filter service replies.
Session Data	Tatsubori and Takahashi [47]	Create service compositions from existing web apps, encode session information in context.
	Matsumura et al. [22]	Save static portions of client server messages, and protocol choice to improve performance.
	Harrison and Taylor [48]	Stateful loosely coupled web services.
Frameworks and Architectures	Orriens and Yang [49]	BCCF
	Kaltz and Zeigler [42][43]	CATWALK framework, alter web pages with XSLT based on Context.
	Elsafty et al. [21]	Context Oriented Architecture (COA). All service interactions have context.
	Harrison and Taylor [48]	WSPeer P2P context service.
	Chen et al. [19]	Context Aware Service Oriented Architecture (CA-SOA). Agent based service selection.
	[23]	Web Service Resource Framework (WSRF).
	Little et al. [10]	WS-Context. Context Service and Context Manager services.
Context Propagation	Keidl and Kemper [20]	Context is propagated in SOAP Header to further services.
Nonfunctional Requirements	Elsafty et al. [21]	Context expresses nonfunctional requirements.

Area	Papers	Notes
Management	Maamar et al. [50]	Manage web service compositions, using context and policies.
Security	Riaz et al. [41]	Implicit identification from client context.
	Keidl and Kemper [20]	Context should be selectively propagated.

Table 3.1: Reviewed context papers, by area.

3.5 Summary

Web services are a technology for constructing distributed systems according to the principles of service orientation. Systems constructed using web services can have complex and varied communication patterns.

Composite web services allow for the construction of complex hierarchies of service consumptions.

The available tools for managing web service based systems tend to focus on managing the services themselves (in the case of WSDM, ESBs), or managing the flow of messages to and from services (in the case of ESBs).

The idea of a supporting web services with contextual information is relatively new. The contextual information usually provides some real world context for the web service consumption, such as the consumer's current location, or it stores sessional data relating to past consumptions of a service, such as a purchase history.

The main available standard in the area of context and web services is WS-Context. The WS-RF standard provides many desirable properties for constructing a context, but its focus is on maintaining state information associated with a particular service.

The current research in the area of contexts focuses mainly on acquiring contextual information, encoding contextual information, and storing sessional data. The applications of context seem focussed on location based service personalization, or context based service selection.

There does not seem to be research on the use of context for process level communication and management. There does not seem to be any implementation or analysis of the costs of employing a context service such as WS-Context.

Chapter 4

Operating Within a Shared Context

4.1 Idea

A context model can be constructed to represent the context within which a whole process executes. Each service, or an agent acting on its behalf, operating within the context has an opportunity to react to contextual information provided by other participants and to contribute contextual information itself.

Contextual information could be anything the participants feel is relevant to the successful completion of the process. Generally it would consist of non-functional requirements which focus the intent of the service consumers beyond the functional parameters required by an immediate service provider. Participants may also provide information about their current execution status in the context.

For example a participant may be prohibited by law from conducting business with a particular state. By encoding this in a context, it is possible to ensure future selected services conform to these requirements.

If a context becomes too constrained, it is possible to request that the constraint be relaxed to facilitate the successful completion of the execution.

In a situation of parallel execution, if a critical failure occurs in one thread of execution it is possible to alert the remaining threads that their completion is redundant.

An Example

Figure 4.1 illustrates an example of a process over web services, which uses uses contextual information to communicate nonfunctional requirements and record logging information for a portion of the process.

Suppose a retailer uses a web service to make an order of desks from a furniture company. The furniture company then uses a broker service to select a manufacturer for the desks. The selected manufacturer receives the plans for the desk, and uses services to order select suppliers and order the materials for the desk. The materials suppliers (lumber, paint, bolts, etc) each select a manufacturer as well.

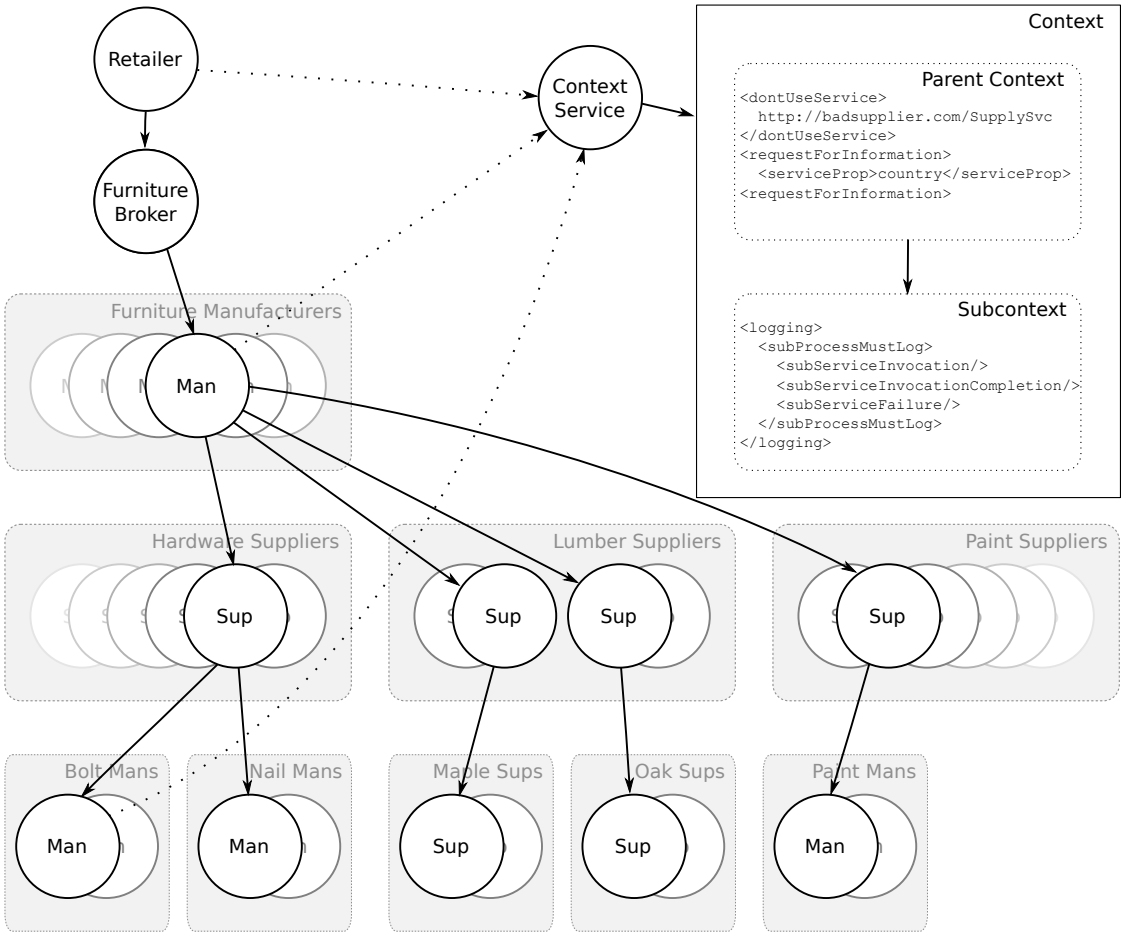


Figure 4.1: A manufacturing process is modelled using web services, using a context service.

In this simple example, there may be the need for separate contexts at the retail and manufacturing levels of decomposition. It may be the case that the retailer knows, from experience, that a bolt sold by a particular supplier, which may sometimes be used in the construction of the desk is prone to failure. It adds an assertion that this supplier's service should not be selected in this process, to avoid receiving desks with poor bolts.

The manufacturer creates a subcontext and adds an assertion that call logging must be done on the sub process. The manufacturer adds an assertion that the lumber supplier must not be a Brazilian company, as it has ethical concerns about deforestation in the amazon.

The selected bolt supplier adds a request that participants disclose their nationality, as it is illegal for it to supply its product to Cubans. This request is placed in the top level context so that it is visible to all participants.

4.2 Considerations

4.2.1 Location

The context could be passed as part of the messages between services, but this would be limiting as communication during the completion of a process tends to travel in one direction (until the operation has completed). Also, it may be the case that there is contextual information a process participant would wish to disseminate after it had already invoked a service.

A scheme of context messages could be passed along the call structure of the process, but this would greatly increase the number of messages exchanged by the process. Also, in the case of a failure of a portion of the process, the messages would not reach the disconnected portions of the process.

A separate service for storing the contextual information has the advantage that it can be reached by any process participant at any time during the completion of the process. This allows for the flow of contextual information in any direction at any point in the lifetime of the process. There is, however, a danger in using a separate context service that you introduce a central point of failure to the process.

4.2.2 Operations

Creation

There must be some way to create a context model. At very least, the creation of a context should create some identifier which represents the shared context, and can be used by process participants to access it.

Population

There must be a way to place contextual information inside the context model. Given the location of the context service, and an identifier which identifies the particular context of interest, the consumer can address the context. The write scheme could be to write the context as a whole, or add to the context, or update some information already in the context or remove some information from the context.

Observation

There must be a way for interested process participants to inspect the context model. Again, given the location of the context service, and an identifier which identifies the particular context of interest, the consumer can address the context. The observation could be active or passive.

In an active observation, the participant requests some portion of the context (the whole, or some part).

In a passive observation, the participant registers an interest in some portion of the context (the whole, or some part), under some condition of interest (addition, deletion, alteration, some particular value). In this case, the participant is notified of the change when it occurs.

4.2.3 Structure

There could be a single context for all participants, and/or some structure of related contexts specific to some sub-groupings of participants.

A natural structure for a context is a hierarchy. This maps well to the functional decomposition style of service consumption. Functionally decomposed activities naturally form contexts over the activities they attempt to complete.

4.2.4 Ownership and Visibility

It may be beneficial for there to be a concept of ownership associated with the information stored in the context. It may be the case that only a subset of participants should be allowed to update a particular piece of information, and only a subset of participants should be able to observe it.

In this case a scheme involving the assigning of privileges starting with the creator of the context could be employed. The creator could assign privileges such as the ability to add pieces of information, update, or delete them. These participants could then assign privileges to the other participants with regards to the information that they add. The context creator could also assign the privilege of creating a sub context as well.

The visibility of sub-contexts, and parent contexts is another issue to consider. Again, the creator of the context could control the visibility of the context by the parent or the children.

Maintaining a system of privileges would require that the participants themselves be identifiable, and verifiably so.

4.2.5 Semantics

The participants in a process must have some shared understanding of the meaning of the entries in the context. Process participants could be selected based on their understanding of the elements which make up the context, as defined in their WSDL entries. Alternatively, services could use an ontology based scheme for writing to the context.

4.2.6 Encoding

The information stored in the context could be encoded in any way that makes sense to the process participants which will make use of it. However, it is standard practice that web services communicate using XML documents, and it seems appropriate to use XML to encode the information within the context too.

4.2.7 Requests for information

The dynamic nature of SOA means that it is difficult to know what information is important to other services in system, especially if it is a service provider attempting to learn about a service consumer. It is important that participants have a mechanism for soliciting information from other participants within a context.

4.2.8 Technical Considerations

In the situation where a process has separate threads of execution, the context must have some mechanism for ensuring that writes to the context do not clobber one another. This could be accomplished by a locking mechanism, either at a context wide level, or per entry level. An alternative might be to require that no other writes have been made since the writing service's last read. Again, the granularity of the reads and writes (how much of the context they affect) will have a bearing on the performance of these operations.

4.2.9 Potential Problems

The addition of a context to processes over web services has some potentially beneficial properties, but it also introduces its own problems: The context service introduces tight coupling, and a single point of failure. The process participants must use the context. The process participants must understand the information stored in the context. The process participants must provide truthful information to the context. The portion of the process employing the context may only be a small part of the overall process, and/or part of some larger non-web service process for which it can not obtain the information it desires. The participants in the process may make unreasonable demands on the other participants, over constraining the process, and causing its overall failure.

Chapter 5

An Erlang Implementation of WS-Context

This chapter introduces a sample implementation of a context server. The context server implements the basic functionality described in the WS-Context specification. The server is implemented using the Erlang programming language (`http://www.erlang.org`). The Erlang programming language was chosen because of its ability to handle high concurrent processes, and its growing popularity as platform for implementing network services.

5.1 Architecture

The architecture of the basic context service follows the WS-Context specification. The WS-Context specification is meant to be extended to implement specific context types, as such it tends to avoid all but the most basic issues associated with implementing a context. Even so, some of its decisions seem too general.

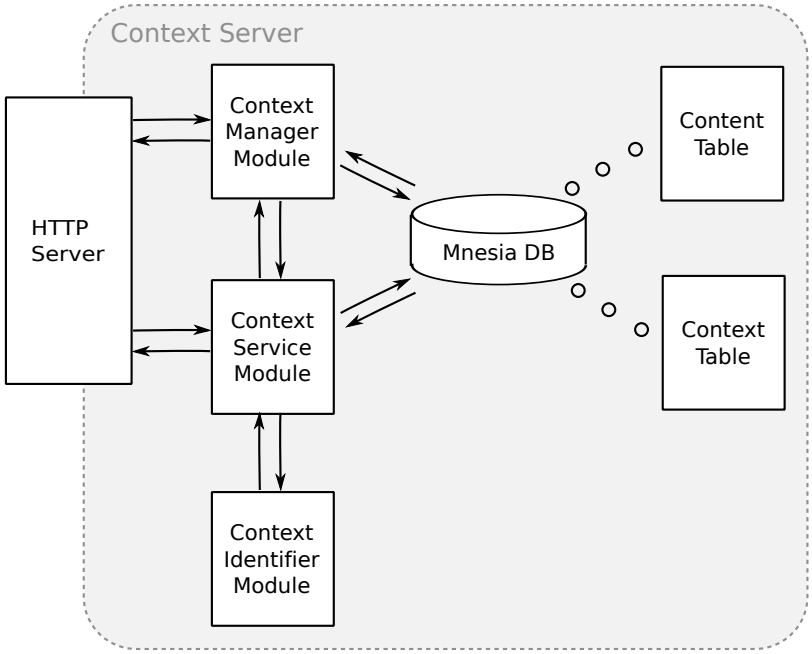


Figure 5.1: The architecture of the basic context server.

Figure 5.1 illustrates the architecture of the basic context server. The context server is composed of the Context Service, and the Context Manager. The Context Service manages the creation, structure, and completion of the contexts. The Context Manager stores the information placed in the context. The HTTP server handles the transfer of messages to and from the server. The Mnesia database (`http://www.erlang.org/doc/apps/mnesia/`) stores the internal data for the context service, and context manager. Mnesia is a small efficient table-based based database which is part of the Erlang platform. The context identifier generator. generates a unique identifier for each context. The context manager consults the context service to determine if an accessed context is still open.

5.1.1 HTTP Server

This implementation of the WS-Context specification uses HTTP as the transport mechanism for interacting with the context service and context manager. The HTTP server hosts the context service, and context manager. The HTTP server dispatches the incoming SOAP messages to the appropriate handler function, and returns the resulting message to the client.

The HTTP server used in this case was an altered version of an example server implemented in Erlang (`http://www.trapexit.org/A_fast_web_server_demonstrating_some_undocumented_Erlang_features`).

5.1.2 Context Service

The Context Service manages the structure and lifecycle of the contexts. The *begin* operation creates an new context, or, a new child context if a the begin message contains an existing context. The *begin* operation may also include a timeout value indicating when the context's state should automatically be set to complete. The *complete* operation sets the internal state of the context to completed. At which point, further attempts to interact with the contest will generate a fault. The *getTimeout*, and *setTimeout* operations provide a way to retrieve a context's timeout, or update it. The *getStatus* operation returns the status of the context if such a thing exists for the particular context in question.

The context service creates a context identifier for each new context. The context identifier is a URI which uniquely identifies the context. In this implementation, a pseudo unique URI is generated using the current time and a randomly generated number. This context identifier is returned to the caller of the begin operation. It can then be shared by the caller with other participants within the same context, and used refer to that context in future interactions with the context service, and context manager.

```
-record(context, {id = [],                  % The context identifier.
                  has_timeout = false,      % Whether or not the context
                                            % has an associated expiry time.
                  timeout = 0,              % The context will expire at
                                            % this time.
                  parent = [],              % The context identifier of this
                                            % context's parent if it has one.
                  children = [],            % A list of contexts which are
                                            % children of this context.
                  has_completed = false,    % Whether or not this context has
                                            % completed.
                  has_status = false,       % Whether or not this context has
                                            % an associated status value.
                  status = []}).            % The status of the context (this
                                            % value is specific to this
                                            % implementation).
```

Listing 5.1: The context service record format used in the Mnesia database

The context data is stored as a set of records in a mnesia database (`http://www.erlang.org/doc/apps/mnesia/index.html`). The stored record consists of the context identifier for the context, the context identifier of the parent context if one exists, the context identifiers of any child contexts if they exist, the timeout value if it exists, a status value if it exists, and a flag indicating whether the context has completed. Listing 5.1 shows the erlang record definition for the context record.

5.1.3 Context Manager

The context manager is where information added to the context is stored. The context is transferred as a whole. The getContent operation returns the whole context for provided context identifier. The setContent operation sets the whole context to the one provided in the set Content message.

```
-record(contents, {id,          % The context identifier for this content.
                   content}).   % The content as a xmlElement record
                                % containing the content of the context.
```

Listing 5.2: The context manager record format used in the Mnesia database

The content data in the context manager is stored as a set of records in mnesia database (`http://www.erlang.org/doc/apps/mnesia/index.html`). The records consist of the context identifier, and the XML representing the context. Listing 5.2 shows the erlang record definition for the content record

5.1.4 Context Server Interactions

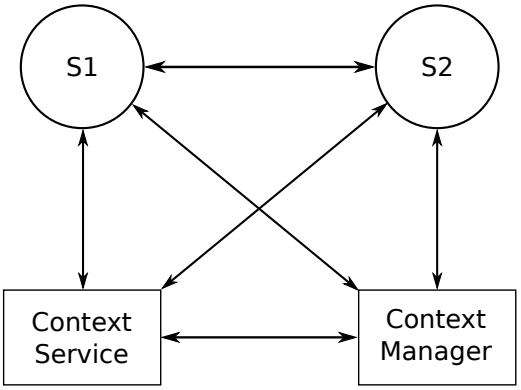


Figure 5.2: Services S1 and S2 interact with a Context Service and Manager

Figure 5.2 illustrates the basic interactions between the participants in a web services process which uses a context service. Initially, service S1 would use the context service to create a context and obtain a context identifier. Service S1 could then store some information in the context manager. The context manager would contact the context service to verify that the context which service S1 was trying to write to existed, and had not completed. Service S1 would then consume Service S2 passing it the context identifier. Service S2 could then obtain the context contents from the context manager, and/or create a subcontext using the context service.

5.1.5 Context Propagation

The context is propagated by passing the context identifier to other participant services in a process. The context identifier is included in the header of a SOAP message during the invocation of a service's operations.

An example of context propagation

In this example (fig. 5.3), a service (S1), creates a new context by consuming the *begin* operation of the context service (listing 5.3). The context service creates a new context, and returns its context identifier to the service in the context section of the message's SOAP header (listing 5.4). The service S1 then consumes an operation of service S2, passing the context identifier, and the locations of the context service, and context manager in the message to S2, again as part of the context section of the SOAP header (listing 5.5).

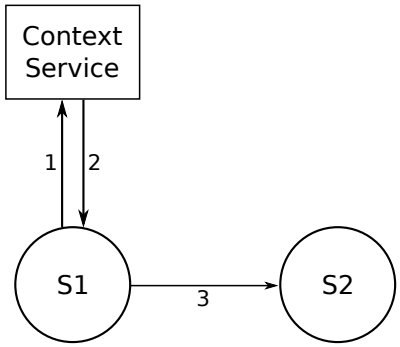


Figure 5.3: 1. Service S1 invokes begin operation. 2. Context Service returns a new context identifier. 3. S1 invokes an operation of S2, includes context identifier in SOAP message.

```
<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <wsctx:begin xmlns:wsctx="http://docs.oasis-open.org/ws-caf/2005/10/wsctx">
    </wsctx:begin>
  </S:Body>
</S:Envelope>
```

Listing 5.3: 1. The SOAP document sent by S1 to obtain a context.

```
<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <wsctx:context xmlns:wsctx="http://docs.oasis-open.org/ws-caf/2005/10/wsctx">
      <wsctx:context-identifier>http://example.org/contexts/789739443585</wsctx:context-identifier>
    </wsctx:context>
  </S:Header>
  <S:Body>
    <wsctx:begun xmlns:wsctx="http://docs.oasis-open.org/ws-caf/2005/10/wsctx"/>
  </S:Body>
</S:Envelope>
```

Listing 5.4: 2. The SOAP document returned to S1 containing the context identifier.

```
<?xml version="1.0"?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <wsctx:context xmlns:wsctx="http://docs.oasis-open.org/ws-caf/2005/10/wsctx">
      <wsctx:context-identifier>http://example.org/contexts/789739443585</wsctx:context-identifier>
      <wsctx:context-service>http://example.org/ContextService</wsctx:context-service>
      <wsctx:context-manager>http://example.org/ContextManager</wsctx:context-manager>
    </wsctx:context>
  </S:Header>
  <S:Body>
    <question/>
  </S:Body>
</S:Envelope>
```

Listing 5.5: 2. 1. The SOAP document sent by S1 to S2 propagating the context.

5.1.6 Context Structure

The use of subcontexts allows the process participants to structure the context as a tree of related contexts. A subcontext is created by including the context identifier of an existing context in the soap header of a call to the *begin* operation of the context service. The context service will create a new context and return it's context identifier.

According to the WS-Context specification, the structure of the context is recorded in the content stored in the context manager. The WS-Context specification includes the <parent−context> tag which can be used to indicate the current context's parent, by referencing its context identifier. The specification does not mention a <child−context> tag, but as the standard is meant to be extended to fit the needs of particular applications, child relations could be included as well.

This implementation of the context server also stores the context structure in the context service record, by recording a context's parent, and its children. This would allow the context service to

complete the child contexts of a context if that were appropriate to the particular application at hand. Storing the structure of the context in the context service would also allow for the structure stored in the context manager to be rebuilt if it were lost.

Process participants can only access one context per operation, so if a participant wants to view the contents of a related context (parent, or child), it must invoke the *getContent* operation using the appropriate context identifier (parent's, or child's) from the current context's contents.

5.1.7 Using the context

The process participants can make use of the context in one of two ways. It can be aware of the context, or it can be unaware of the context.

Service is Aware of Context

There may be cases where the awareness of a context is an integral part of a service. It these cases, the web server passes the entire soap message to the web service operation being invoked. The web service operation is responsible for interpreting the context portion of SOAP messages it receives, and propagating the context with messages it sends. Figure 5.4 illustrates this process.

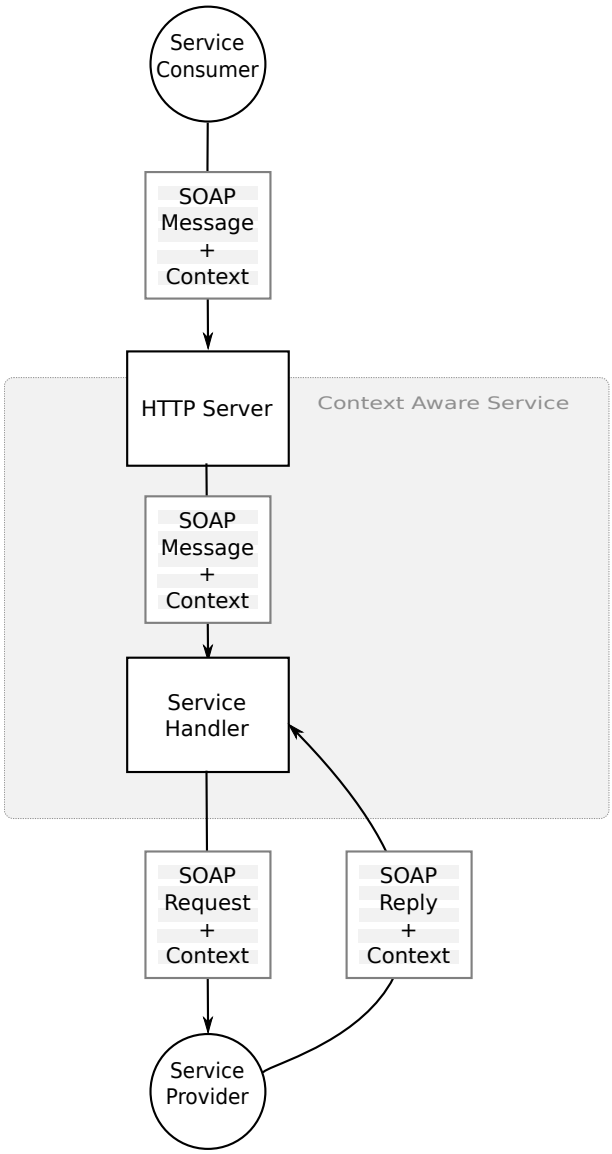


Figure 5.4: The service is responsible for handling context information.

Service is Unaware of Context

More commonly, it is unnecessary for the service to be aware of the context. The context is used solely as a tool to facilitate the management of the overall process, and is invisible to the service itself. Figure 5.5 illustrates this process.

In such cases, a context handler function is called by the http server. The context handler strips the context from the SOAP message, and stores the relevant context properties (*ContextId*, *ContextServiceURL*, and *ContextManagerURL*), and context handler locations in a per-thread *ets* data table (an erlang dictionary). The context handler calls the *onReceive* handler, which has been registered with the http service for this web service operation. The onReceive handler performs any operations related to the invocation of the web service operation. The context handler then returns the SOAP message, now stripped of its context, to the http server now invokes the web service operation requested by the service consumer, and passes it the contextless SOAP message.

During its completion, if the web service operation consumes some other web service, the *onSend* handler will be triggered, and passed the SOAP message. The *onSend* handler has an opportunity to complete any context related operations before it adds the context information to the SOAP header, and returns it to the message sending function.

If a reply is received from the web service which has been consumed, the *onReply* handler is triggered. The *onReply* handler has an opportunity to complete any context related operations, before stripping the context and returning the contextless SOAP message to the message receiving function.

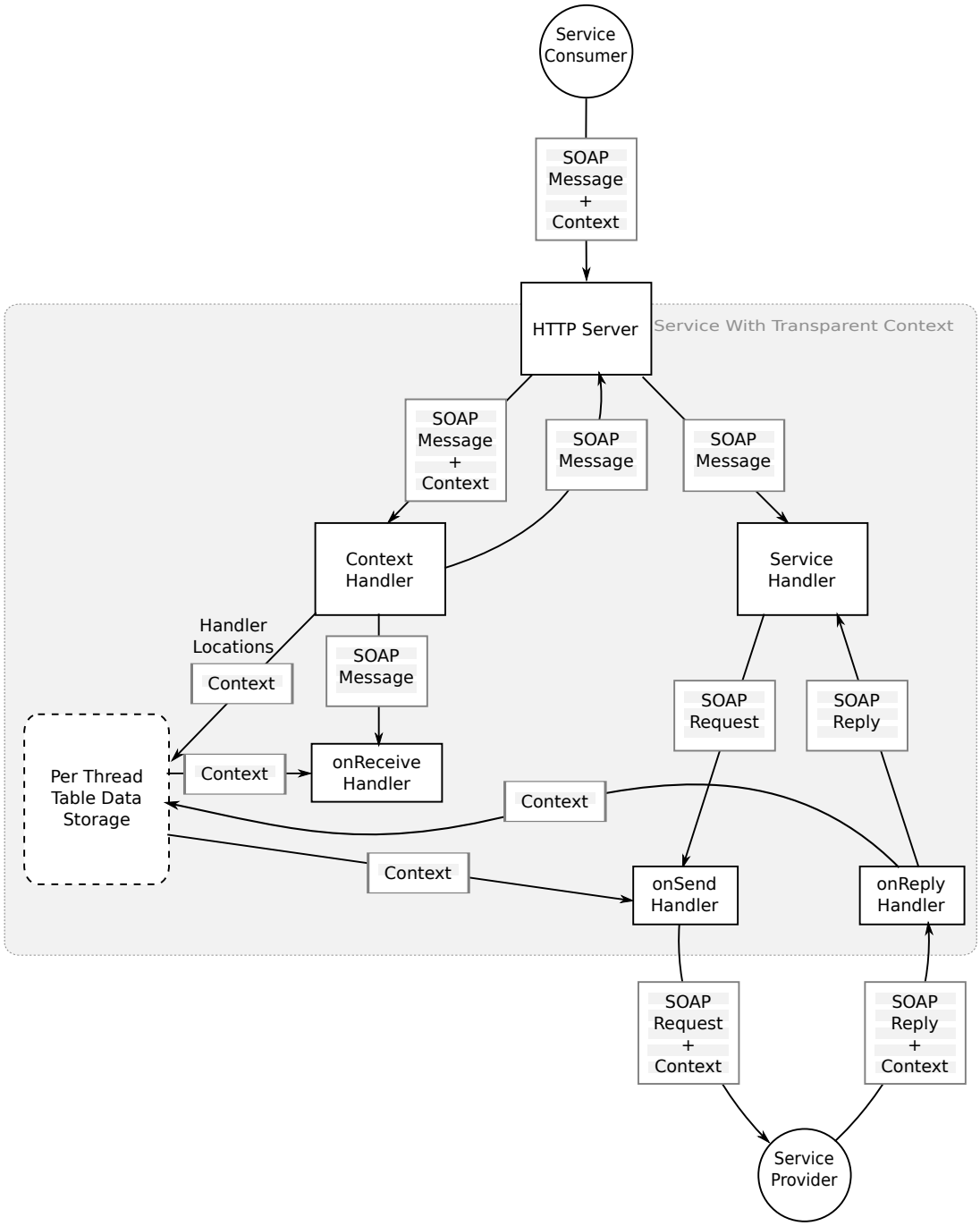


Figure 5.5: A set of handler functions process the context, and the service is un aware of its existence.

5.1.8 Monitoring Processes (An Example)

The process monitor service logs the consumption of services in the process. Each service in the process creates its own child context, and records its invocation, and consumptions.

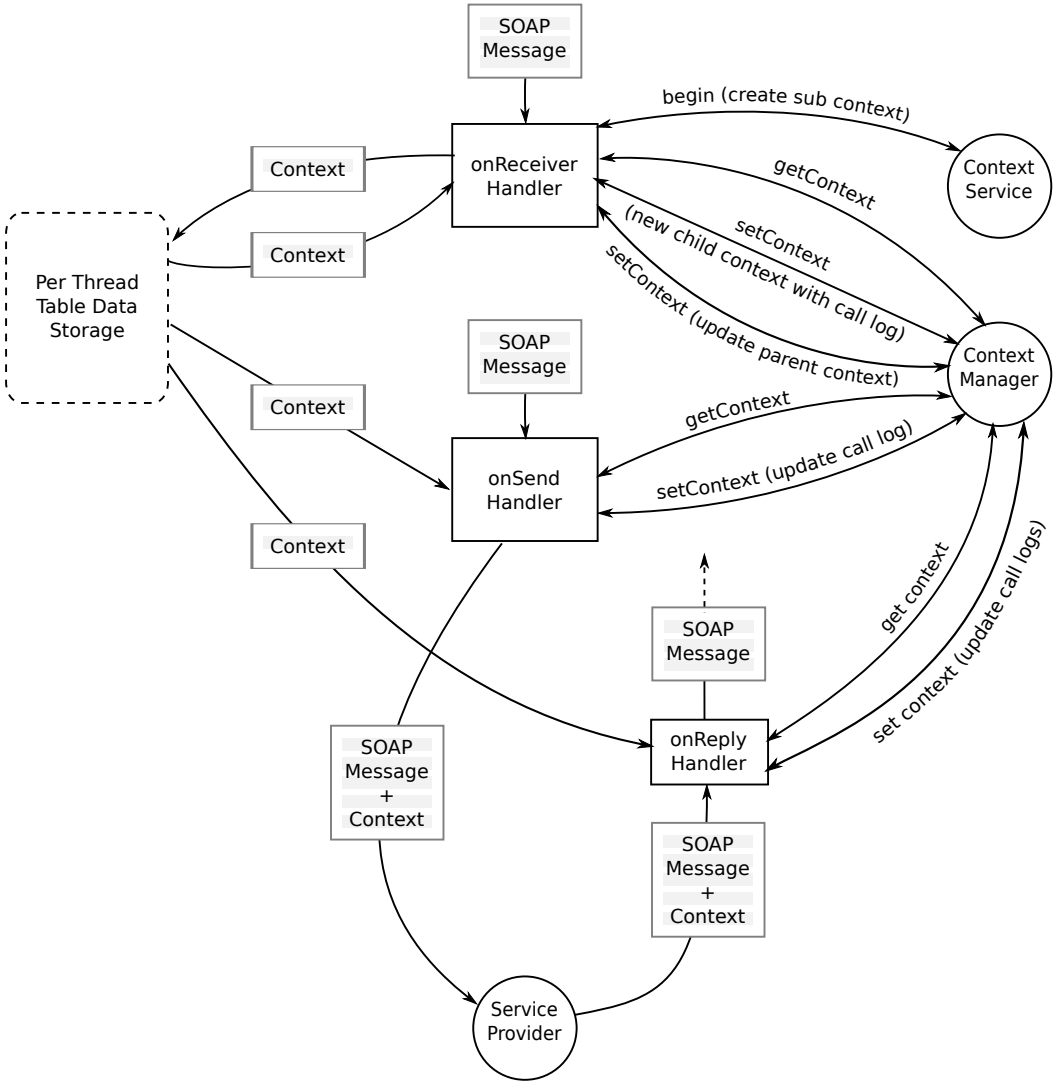


Figure 5.6: The function of the onReceive, onSend, and onReply handlers.

The monitoring management functionality is an example of a context unaware service. The *onReceive*, *onSend*, and *onReply* handlers record the service interactions to the context, and the service being monitored is unaware that it is being monitored. Figure 5.6 illustrates the function of the *onReceive*, *onSend*, and *onReply* context handlers.

The onReceive Handler

The *onReceive* handler invokes the context service's *begin* operation, creating a new sub-context. Next, the context manager's *getContext* operation is invoked to retrieve the parent context. Then the context manager's *setContent* operation is invoked to add a new context content to the context manager. The new context includes a reference to the parent context, and a callLog with a record of the invocation of this operation. Listing 5.6 illustrates the content of the new context. Finally, the *onReceive* handler invokes the context manager's *setContent* operation to add the new child context to the parent context's content. This is illustrated in listing 5.7.

```
"<content>
  <wsctx:parent-context xmlns:wsctx="http://docs.oasis open.org/ws caf/2005/10/wsctx">
    <wsctx:context-identifier>
      http://example.org/Contexts/514094808
    </wsctx:context-identifier>
  </wsctx:parent-context>
  <callLog>
    <receivedCall>
      <thisEndpoint>
        http://example.org/AnswerService
      </thisEndpoint>
      <thisOperation>
        answer
      </thisOperation>
      <atTime>
        2010-12-06T12:00:00
      </atTime>
      <caller>
        192.168.0.123:65049
      </caller>
    </receivedCall>
  </callLog>
</content>",
```

Listing 5.6: The contents of the new context

```
"<content>
  ...
  <wsctx:child-context xmlns:wsctx="http://docs.oasis open.org/ws caf/2005/10/wsctx">
    <wsctx:context-identifier>
      http://example.org/Contexts/23789837498
    </wsctx:context-identifier>
  </wsctx:parent-context>
</content>",
```

Listing 5.7: The addition to the parent context.

The onSend Handler

The *onSend* handler is passed the outgoing SOAP message. First, it retrieves the context information from the *ets* table storage, then invokes the context manager's *getContent* operation to retrieve the context's content. The handler then invokes the context manager's *setContent* operation to update the context's content with the addition of a record recording this service consumption. Listing 5.8 illustrates the added record. Finally, the *onSend* handler adds the context information to be propagated to the SOAP message, and returns it to the message sending function.

```
"<content>
  ...
  </wsctx:parent-context>
  <callLog>
    ...
    <sending>
      <thisEndpoint>
        http://example.org/AnswerService
      </thisEndpoint>
      <thisOperation>
        answer
      </thisOperation>
      <atTime>
        2010-12-06T12:01:00
      </atTime>
      <toEndpoint>
        http://example.org/SomeOtherService
      </toEndpoint>
    </sending>
  </callLog>
</content>",
```

Listing 5.8: The callLog record added by onSend

The onReply Handler

The *onReply* handler is passed the incoming SOAP reply message. First, it retrieves the context information from the *ets* table storage, then invokes the context manager's *getContent* operation to retrieve the context's content. Then, it invokes the context manager's *setContent* operation updating the content with a record in the callLog indicating that a reply was received. The addition to the context's content is illustrated in listing 5.9. Finally, the *onReply* handler strips the context from the SOAP reply, and returns it to the message sending function.

```
"<content>
  ...
  </wsctx:parent-context>
  <callLog>
    ...
    <gotReply>
      <thisEndpoint>
        http://example.org/AnswerService
      </thisEndpoint>
      <thisOperation>
        answer
      </thisOperation>
      <atTime>
        2010-12-06T12:02:00
      </atTime>
      <replyFromEndPoint>
        http://example.org/SomeOtherService
      </replyFromEndpoint>
    </gotReply>
  </callLog>
</content>",
```

Listing 5.9: The callLog record added by onReply

Chapter 6

Use Scenarios of Context in SOAP Web Services

6.1 Monitoring

The context may be used to store data about the status of the process. At least it may be used to record the services which have been involved in the process, and the hierarchy of execution. If a service fails this may be recorded in the context.

The process could record when a consumer consumed a service, when the service consumption completed, what the completion status was (successful, unsuccessful). While a process is active, the various service providers may note processing milestones, and may post expectations of completion times.

While the process is active, a monitoring agent (human or software), may inspect the context to observe the status of the process. The context may be saved, and analyzed, or played back at a later date.

Being able to analyze process performance at a later date may reveal problem which may not be otherwise immediately obvious. Situations such as a complex failure where the process fails only when a certain service provider is selected on a certain day, may become apparent. This provides a heuristic which may then be encoded in the context, which will avoid this failure in the future.

6.2 Reacting to Changing Requirements

The context may be updated with new or changed requirements. The service providers may inspect the context and react to these changes, or a monitoring agent may be employed to notify the service providers of the change.

The parameters which are supplied to a service provider on service consumption should be recorded in the context, and as they pass down through the process, providers which depend on them should register to be notified if they change. This provides the process with a mechanism for reacting quickly to changing requirements.

6.3 Avoiding Self Competition

In a process which employees parallelism, the prospect of self competition arises. Separate branches of the execution may compete for resources, and may influence the overall cost of the process. By using information in the context it may be possible to avoid this self competition.

If an area of self competition is identified in the process, an exclusion scheme such as a context based semaphore [51] may be employed.

6.4 Fault Handling

6.4.1 Terminating Redundant Branches

If a process employs parallelism, and a failure has occurred on one branch. It is possible using the context to terminate the entire process. Instead of letting the other branches run to completion.

A value could be stored in the context which indicated the status of the process, or of a sub component of the process. Interested participants could register to receive notifications if this value changed. This could be used to pause or halt processing of the interested portions of the process.

6.4.2 Extending Runtime On Timed Out

In the case of processes which have failed due to some constraint, for example time-out failure. It is possible to use the context to relay the cause of the failure to the originating entity, which may then relax the operating constraints, and continue the process.

This is really a special case of changing requirements. A service provider has reached a timeout. By pausing the sub process, it is working on, and registering a fault, referencing the requirement that it has violated, and noting that it could complete if the requirement were relaxed, it gives the specifier of the requirement the opportunity to relax that constraint, and continue the process.

6.5 Implicit Parameters

It is unreasonable in the construction of composite services to account for every possible variation in the needs of potential consumer. The composite service will likely make arbitrary parameter choices of the services it consumes. It could register these decisions in the context, and similar to the case of changing requirements, a service consumer may have an opportunity to alter these arbitrary, pass-through requirements.

Chapter 7

Performance Evaluation

The additional overhead of a context service definitely adds to the completion time of a process, but it is envisaged as an aid to long running processes, which are modelling real world processes. As such, the performance overhead should be more than acceptable.

7.1 Testing the Basic Context Service

Any context service developed following the WS-Context specification will have at a minimum these basic operations: *begin*, *complete*, *setTimeout*, and *getTimeout*. If a context manager is employed, then the operations *getContent*, and *setContent* would also be available.

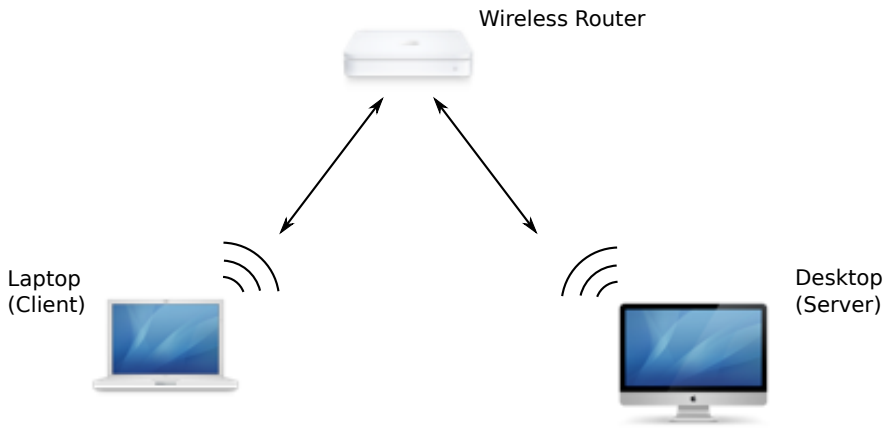


Figure 7.1: The test hardware and network configuration.

The performance of these basic operations was tested using a desktop computer running the context server, and a laptop running test client software. The desktop computer is an Apple Macintosh with a 3.06 GHz Intel Core 2 Duo processor, and 4 GB of memory, running OS X version 10.6.5. The laptop is an Apple iBook with 1.2 GHz PowerPC processor, and 768 MB of memory, running OS X version 10.5.8. The computers communicated over a wireless LAN connection. The wireless router is a D-Link DI-534 802.11g. The basic hardware setup is illustrated in fig. 7.1.

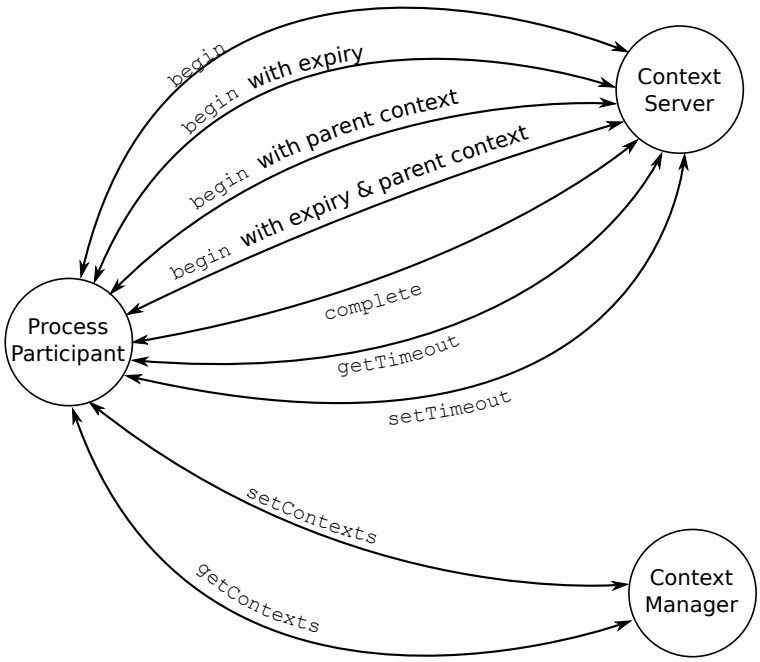


Figure 7.2: The test of the basic context operations.

The speed of the wireless LAN connection was estimated using the scp utility to transfer a large file, and found to average approximately 472 kb/s with a high degree of variance. The variance in the network performance is assumed to be a result of interference with the wireless network signal, possibly by other devices operating in the same frequency range or some physical obstruction.

A Test of the Basic Operations Over a Network Connection

The test client software invoked each of the basic operations of the context service, and those of the context manager (illustrated in fig. 7.2). The four forms of *begin* operation were all invoked. These operations were all invoked in order sequentially. The test client blocked waiting for each response. The invocation times were recorded for each operation. The sequence of invocations was repeated 50 times. The number of repetitions (50) was selected with the objective of capturing a representative sample of server performance.

The average invocation time (in microseconds) for each operation is reported in table 7.1, along with their standard deviations. Histograms of the call times for the various operations are illustrated in figures 7.3 - 7.11.

The standard deviations are quite large relative to the mean times. For example, the *begin* operation had a mean time cost of 90,011 microseconds with a standard deviation of 252,292. Possible explanations for the large standard deviations are the poor wireless network performance or context switching on either the client or server. There is also an unexpected difference between

the performances of the different forms of the *begin* operation, which one would expect to be almost identical.

Operation	**Mean Time to Complete (microseconds)**	**Standard Deviation**
begin	90,011	252,292
begin with expiry	45,743	129,835
begin subcontext	41,927	141,074
begin subcontext with expiry	27,492	20,175
complete	91,391	227,632
setTimeout	32,416	29,354
getTimeout	57,286	159,350
setContents	33,356	29,332
getContents	46,748	135,087

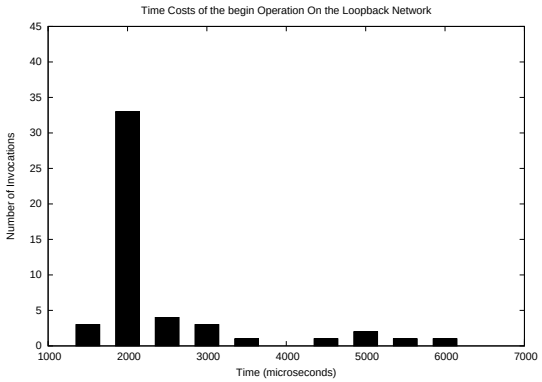
Table 7.1: Sample time costs of basic WS-Context operations over a LAN connection

A Test of the Basic Operations on the Same Machine

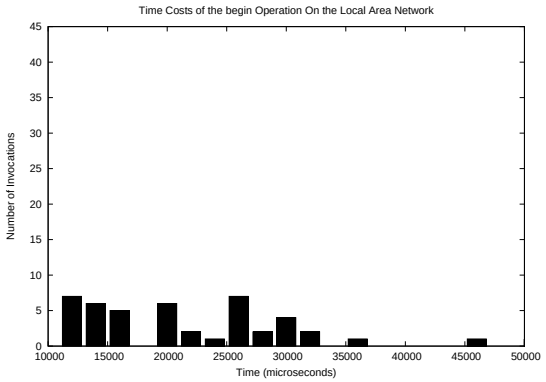
In an attempt to isolate the impact of the network connection on the performance of the basic operations, the test client software was run again; this time on the same machine as the context server. Again, the basic operations were invoked in sequence and their invocation times were recorded. The sequence of invocations was again repeated 50 times.

The average invocation times (in microseconds) for each operation is reported in table 7.2, along with their standard deviations. Histograms of the call times for the various operations are illustrated in figures 7.3 - 7.11.

Overall, the variances are considerably smaller relative to the mean, with the exceptions of the first and fourth forms of the *begin* operation, and the *getContents* operation. The first invocation of the *begin* operation took 188,539 microseconds, and must suffer from a cold start problem skewing the results. The other high invocation times must represent interference from other processes on the machine most likely operating system context switches.



(a)

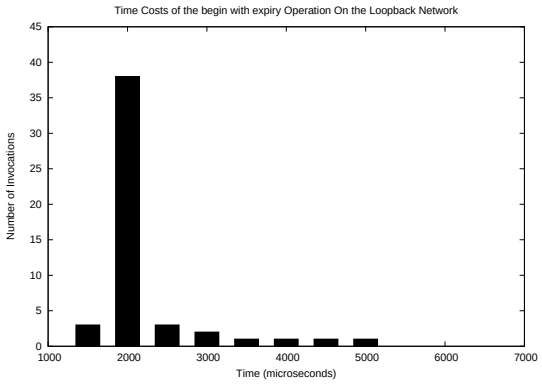


(b)

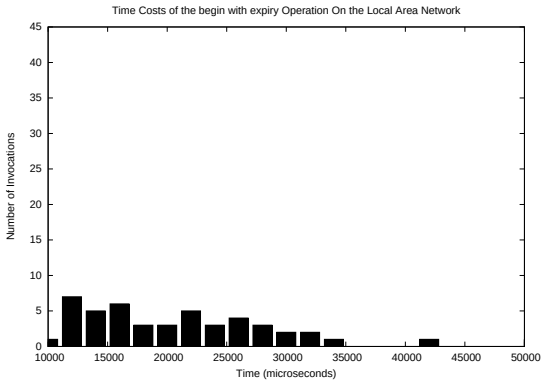
Figure 7.3: A histogram of the call times for the begin operation on the loopback network (a) and the local area network (b).

Operation	**Mean Time to Complete (microseconds)**	**Standard Deviation**
begin	6,320	26,315
begin with expiry	2,416	687
begin subcontext	2,638	926
begin subcontext with expiry	3,292	3,532
complete	2,540	594
setTimeout	2,303	351
getTimeout	2,528	678
setContents	2,660	915
getContents	3,265	4,001

Table 7.2: Sample time costs of basic WS-Context operations on the local machine

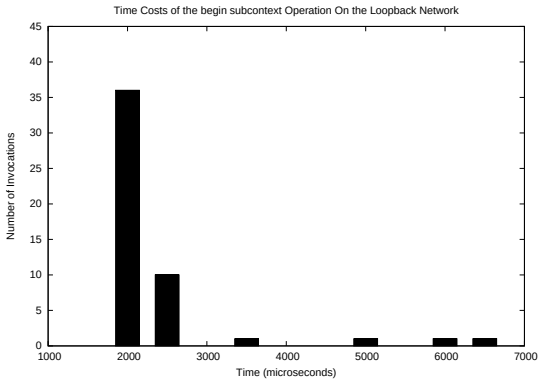


(a)

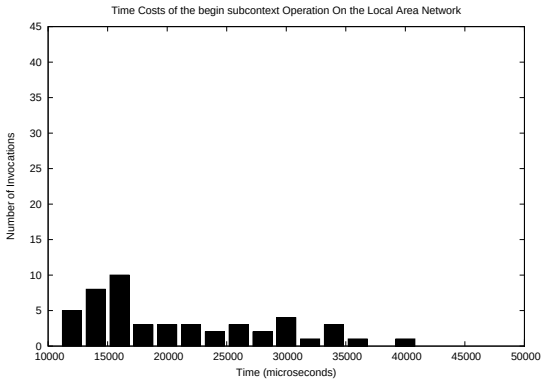


(b)

Figure 7.4: A histogram of the call times for the begin with expiry operation on the loopback network (a) and the local area network (b).

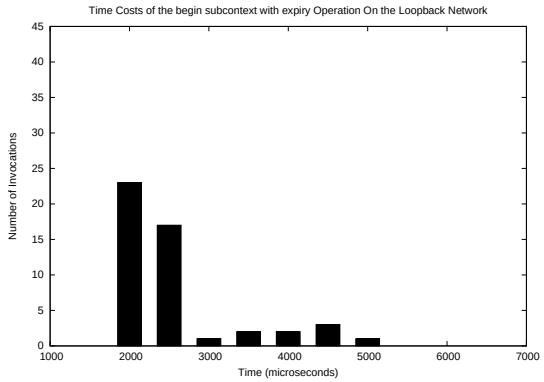


(a)

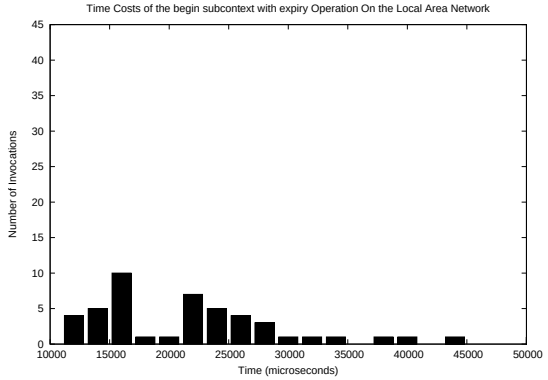


(b)

Figure 7.5: A histogram of the call times for the begin subcontext operation on the loopback network (a) and the local area network (b).

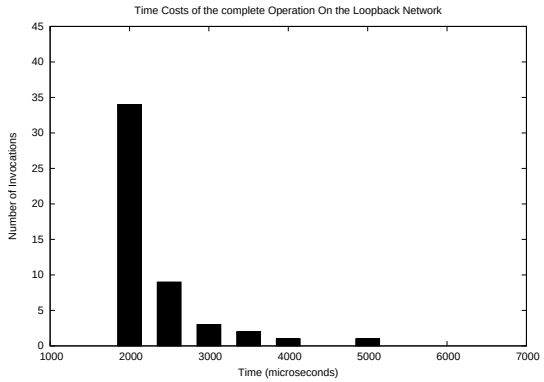


(a)

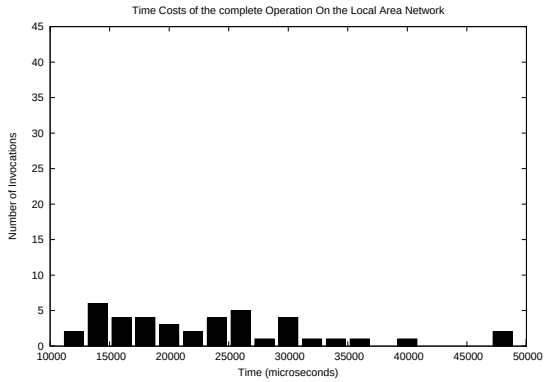


(b)

Figure 7.6: A histogram of the call times for the begin subcontext with expiry operation on the loopback network (a) and the local area network (b).

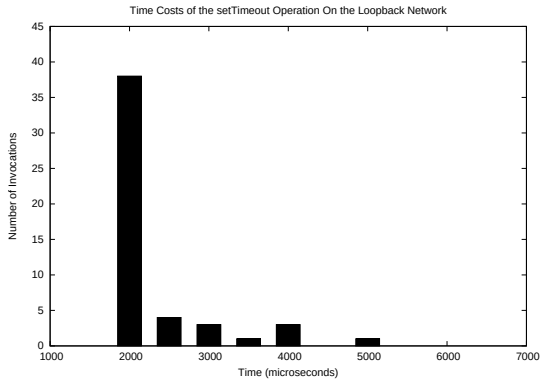


(a)

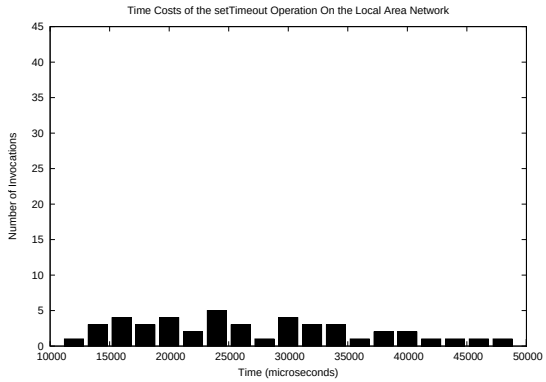


(b)

Figure 7.7: A histogram of the call times for the complete operation on the loopback network (a) and the local area network (b).

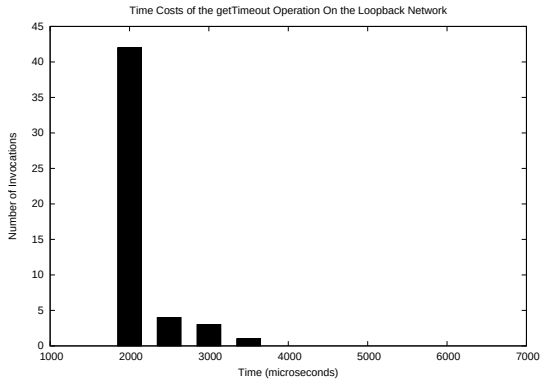


(a)

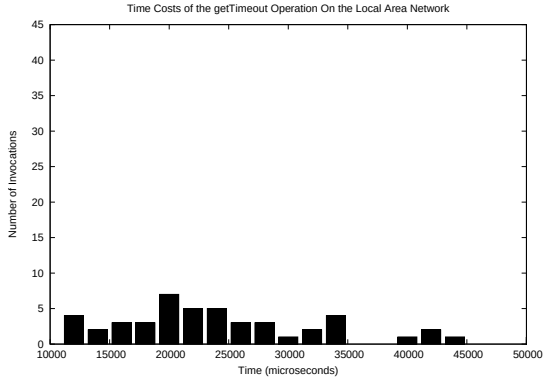


(b)

Figure 7.8: A histogram of the call times for the setTimeout operation on the loopback network (a) and the local area network (b).

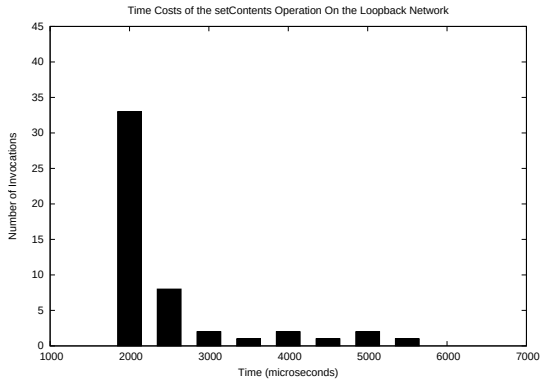


(a)

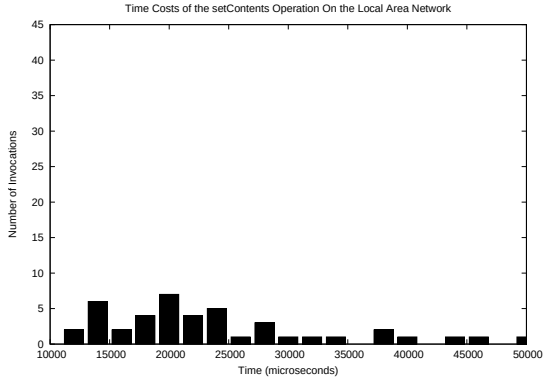


(b)

Figure 7.9: A histogram of the call times for the getTimeout operation on the loopback network (a) and the local area network (b).

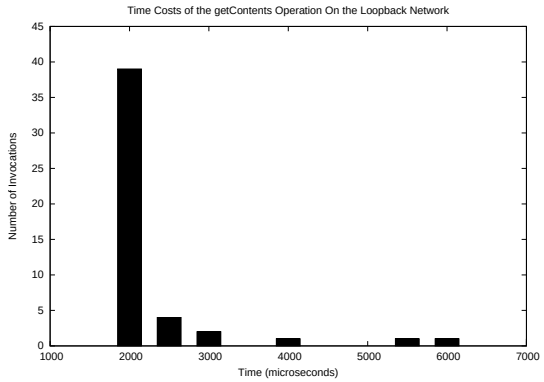


(a)

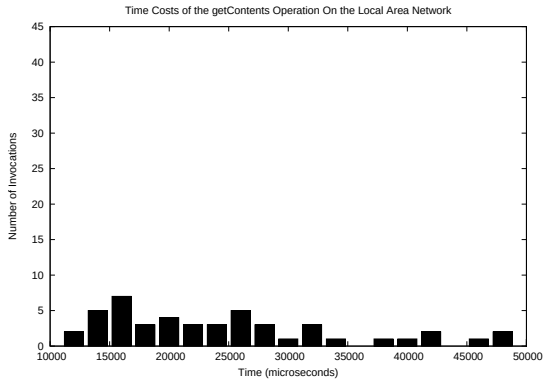


(b)

Figure 7.10: A histogram of the call times for the setContents operation on the loopback network (a) and the local area network (b).



(a)



(b)

Figure 7.11: A histogram of the call times for the getContents operation on the loopback network (a) and the local area network (b).

7.1.1 Testing the Performance of different Context Sizes

The time costs of all the basic operations, with the exception of *getContent*, and *setContent*, are constant. The *getContent*, and *setContent* operations time costs depend in some part on the size of the contest being transferred. A test was conducted, again on the desktop machine to isolate the network variance, to determine the impact of increasing context size on the time cost of the operations.

Tests were conducted for context sizes of 10 bytes, 100 bytes, 1,000 bytes, 10,000 bytes, and 100,000 bytes. It is not possible to anticipate how the context service will be used, so a broad range of context sizes were tested. For each test, a *setContent* operation set the content to the characters, and a *getContent* retrieved the characters. This operation sequence was repeated 50 times for each content size. The average time costs for each operation at each content size are reported along with their standard deviations in Table 7.3. Figure 7.12 illustrates the relationship between the size of the content and the time it takes to complete the *getContent* and *setContent* operations.

As expected, the bigger the content size, the longer the operations take to complete. For the smaller sizes context (10 to 1,000 bytes), the basic cost of doing a database read/write dominates. For the larger sizes (10,000 and 100,000 bytes), the size of the context takes over, and the time cost can be seen to grow linearly with size of the context.

Content Size	**Operation**	**Mean Time to Complete (microseconds)**	**Standard Deviation**
10	*setContents*	1,612	452
10	*getContents*	1,686	653
100	*setContents*	1,754	678
100	*getContents*	1,679	435
1,000	*setContents*	2,322	493
1,000	*getContents*	2,582	1,238
10,000	*setContents*	6,821	418
10,000	*getContents*	7,431	546
100,000	*setContents*	66,508	1,556
100,000	*getContents*	74,754	6,528

Table 7.3: Sample time costs for Context Manager operations, with varying content sizes, on the local machine

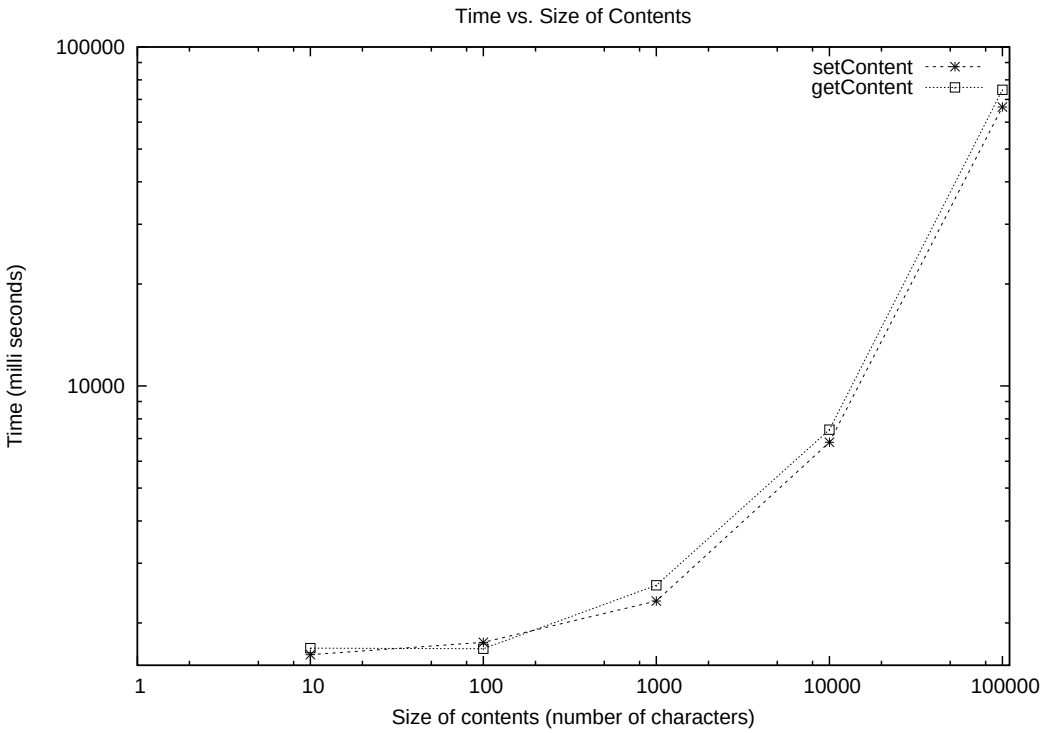


Figure 7.12: The time cost for getContent and setContent operations for various content sizes (number of characters).

7.2 Context Propagation

The most basic function of operating in a context is creating and propagating a context. These tests measure the cost of propagating a context through a process.

7.2.1 Context Propagation Over Services on the Same Machine

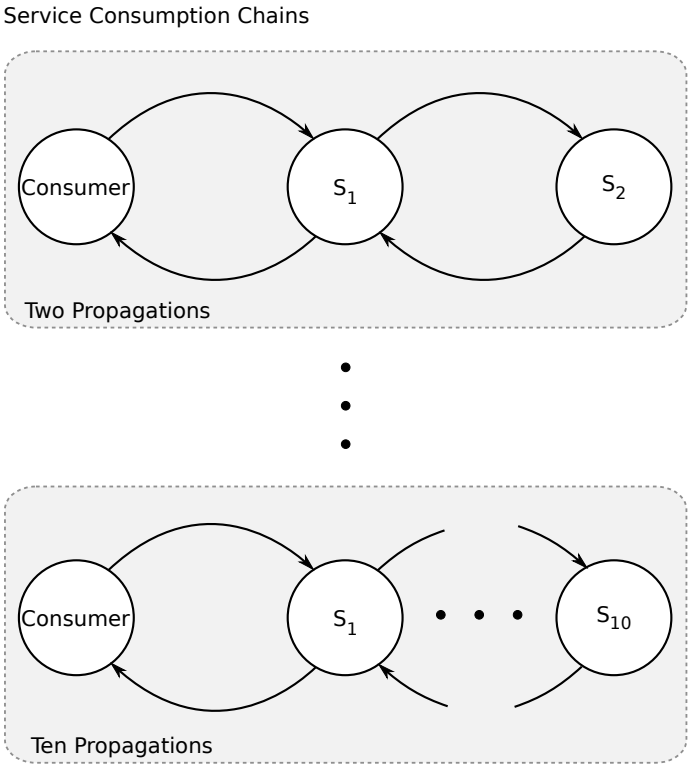


Figure 7.13: Service consumption chains with two to ten propagations were tested.

This test shows the overhead of propagating a context for chains of 2 to 10 service consumptions (Figure 7.13). The test was run on an Apple Macintosh with a 3.06 GHz Intel Core 2 Duo processor, and 4 GB of memory, running OS X version 10.6.5. This test has two parts: service consumption chains with context propagation, and service consumption chains without context propagation. In each case, a set of services was run, with a client service consumer initiating the first consumption. The service receiving the request would then consume another service, and so on, to form the chain. In the tests where contest was used, the client consumer creates the initial context, and passes it to the initial server. The initial, and subsequent services retrieve the context information from the incoming requests, and contact the context manager to retrieve the context, add to it, and return it to the context manager. Each test was run 50 times and the resulting mean times to complete, and standard deviations are presented in Table 7.4.

There is overhead to propagating the context and it increases approximately linearly to the number of propagations. The area of application of systems employing the context will determine whether this is an acceptable overhead.

The standard deviation for the test with the service chain length of 9 at 19,075 microseconds is high relative to the other standard deviations reported. One reason for this could be a context switch at the operating system level on the test machine.

Has Context	**Chain Length**	**Mean Time to Complete (microseconds)**	**Standard Deviation**
no	3	2,160	1,776
yes	3	18,118	4,721
no	4	3,490	2,262
yes	4	24,747	5,523
no	5	4,362	2,150
yes	5	32,357	7,928
no	6	5,580	2,606
yes	6	39,573	6,750
no	7	6.842	2,816
yes	7	46,001	7,411
no	8	7,791	2,629
yes	8	51,747	6,001
no	9	9,230	3,797
yes	9	63,796	19,075
no	10	10,361	3,006
yes	10	67,485	5,491
no	11	11,952	8,181
yes	11	73,950	4,646

Table 7.4: Sample time costs of service consumption chains with and without context propagation.

7.3 Stress Testing

These tests attempt to discover the performance limits of the server. In both tests, the context server was run on an Apple iBook laptop with a 1.2 GHz PowerPC processor and 768 MB of memory, running OS X version 10.5.8. The context server test software was run on an Apple Macintosh with a 3.06 GHz Intel Core 2 Duo processor, and 4 GB of memory, running OS X

version 10.6.5. The machines were connected by an ethernet LAN using a D-Link DI-534 router. The network connection speed was tested using the scp utility to send a large file between the machines, and was found to be 10.9MB/s with a low degree of variability.

7.3.1 Closed Loop Test

This test simulates a set of clients interacting with the server, and it attempts to determine how many clients can simultaneously interact with the server. The test software simulates n clients making k sequential requests, with a interrequest think time of t. For all tests, the interrequest think time (t) was set to one second. The test was run for 10 to 170 clients sending 10, 20, and 30 requests.

The results of the tests are summarized in Figures: 7.15 (minimum response times), 7.14 (maximum response times), 7.16 (mean response times), and 7.17 (mean duration to complete all requests). All figures show plots for tests where the clients each send 10, 20, and 30 requests.

For the mean duration plots, the mean duration rises slowly from 10 to 70 clients, then begins to rise more quickly from 80 to 170 clients as the server begins to saturate, and arriving clients are must wait in a queue. There is a peak in the plots of the clients sending 10 requests (Figures 7.16 and 7.14 subfigure a). This was most likely a result of some network, or operating system interference.

The mean response times plots, as expected, follow a similar, pattern; rising slowly to 70 clients, then rising more quickly to 170 clients.

7.3.2 Open Loop Test

This test simulates bursts of arriving requests, and attempts to determine the maximum throughput of the system. The test software makes b bursts of r requests (five bursts were used for all tests), with an interarrival time i (times of 500 to 0 milliseconds were used). For each burst, requests are sent every i milliseconds, without waiting for the response of the previous request. When all the requests have been sent, the software waits for all requests to complete. The software records the time it takes to complete all the requests d and how many requests were successful s. The throughput is calculated by dividing the number of successful requests by the time it takes to complete all the requests.

The results of the test is summarized in figures: 7.18 (throughput for 100 requests sent at decreasing interarrival times), 7.19 (throughput for 100 requests sent at decreasing interarrival times), and 7.20 (throughput for 100 requests sent at decreasing interarrival times). Each figure contains two plots. The first plot (a) shows the full test. The second plot (b) focusses on the point around the peak throughput.

All figures follow a similar pattern. The throughput remains low for interarrival times between 500 to 150 milliseconds where the server is under utilized and waiting between requests. Then, the throughput begins to climb sharply from 150 to 4 milliseconds as the server becomes fully utilized. Finally, the throughput falls precipitously from 4 to 0 milliseconds as the server becomes overloaded, and queues incoming requests. It is notable that the server did not fail, but eventually completed all requests successfully. For the tests with 100 and 200 requests, the maximum throughput achieved was 190 requests per second. For the test with 300 requests, the maximum throughput achieved was 160 requests per second.

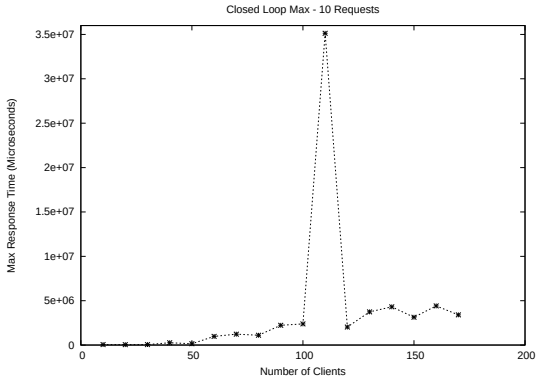
7.4 Summary

The context server was evaluated to determine the performance implications of employing a context server, and the server it self was stress tested to determine its performance.

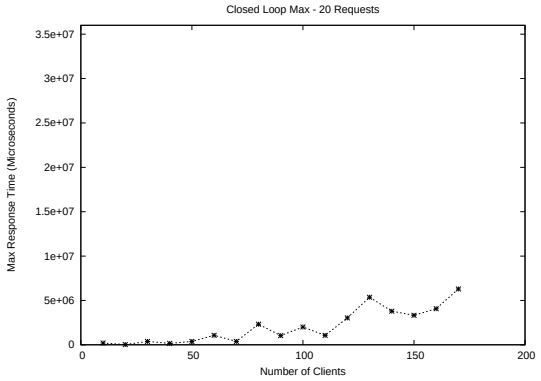
Testing the basic operations of the context service showed that they had an effectively constant time cost in the 10s of milliseconds. The operations for setting and retrieving a context's content were tested and observed to change linearly with the change in the size of the context. The cost of propagating the context along chains of service invocations was tested and observed to have a time cost overhead which grew linearly with the length of the service invocation chains.

Stress testing the context server found that for the closed loop tests, which simulated the interaction of concurrent clients, response times remained relatively unchanged up to 75 concurrent clients, and increased slowly for increasing numbers of concurrent clients. The open loop tests, which simulated bursts of arriving requests, showed a maximum throughput of between 160 and

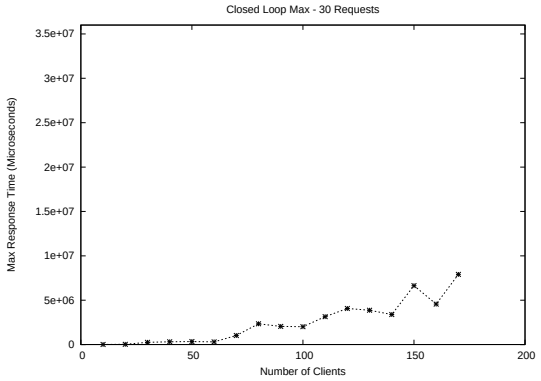
190 requests per second with an interarrival time of 4 milliseconds.



(a)

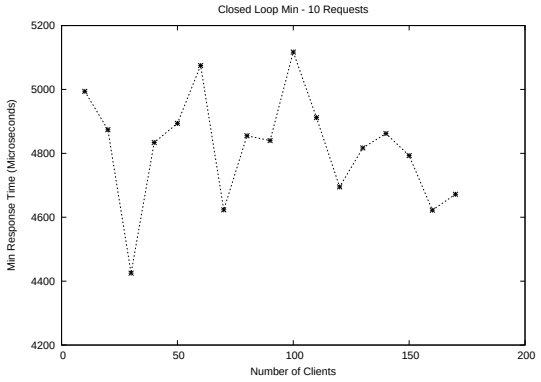


(b)

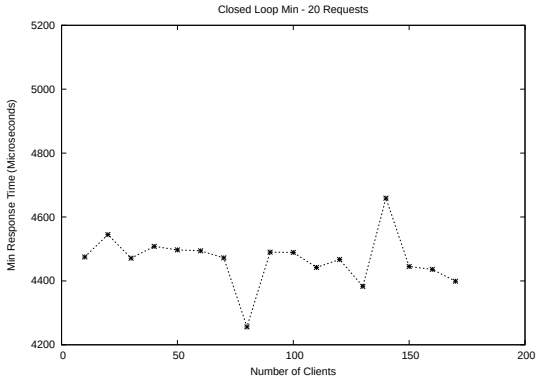


(c)

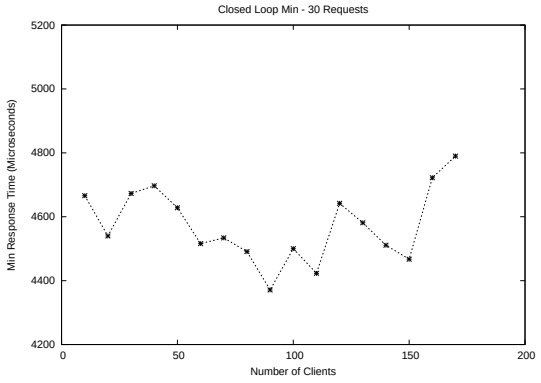
Figure 7.14: The maximum observed response times at each number of clients, for clients sending 10, 20, and 30 sequential requests



(a)

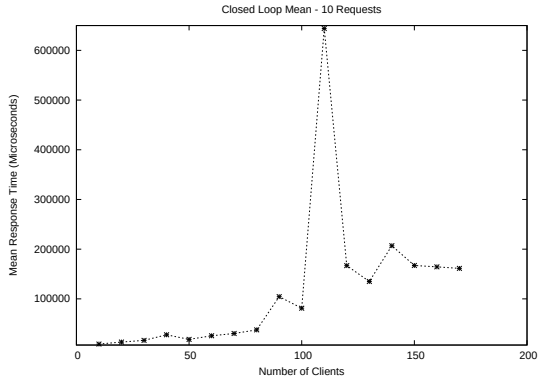


(b)

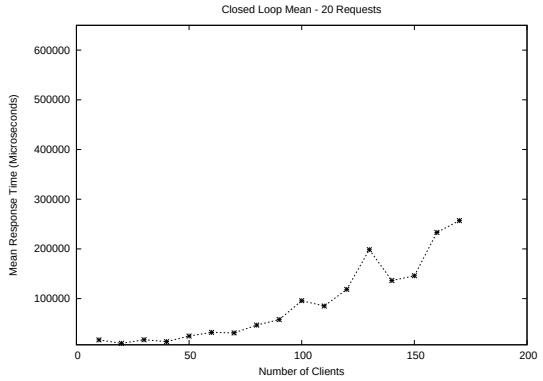


(c)

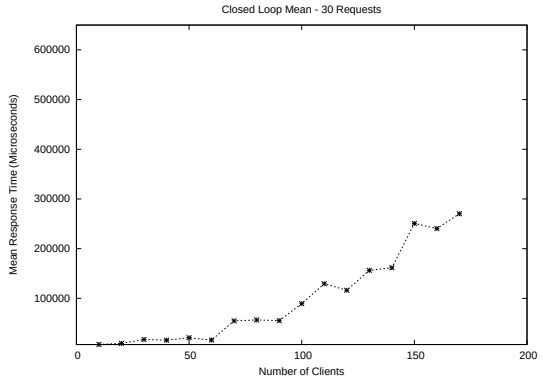
Figure 7.15: The minimum observed response times at each number of clients, for clients sending 10, 20, and 30 sequential requests



(a)



(b)



(c)

Figure 7.16: The mean response times at each number of clients, for clients sending 10, 20, and 30 sequential requests

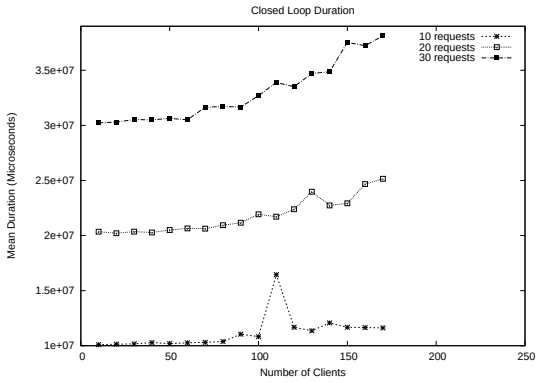
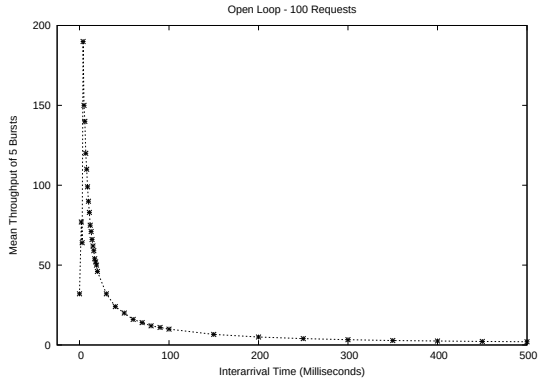
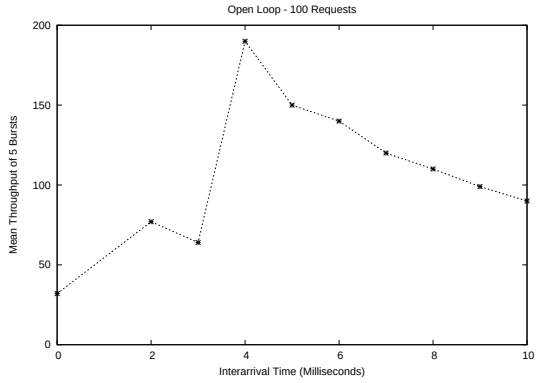


Figure 7.17: The mean duration of time for all the client's requests to be processed at each number of clients, and for clients sending 10, 20, and 30 sequential requests

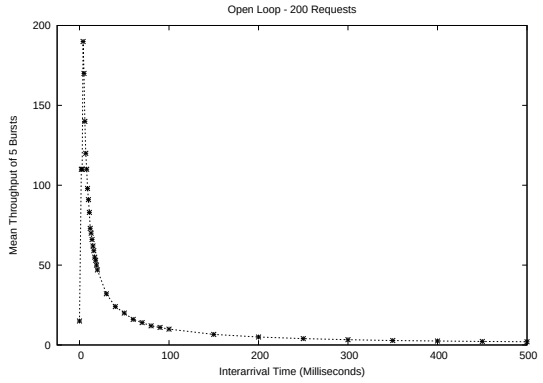


(a)

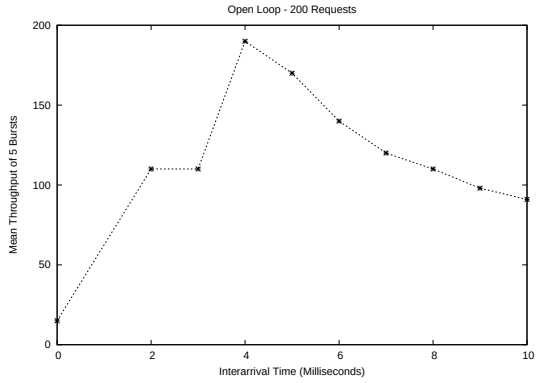


(b)

Figure 7.18: (a) The context server throughput for interarrival times from 500 to 0 milliseconds for a burst of 100 requests. (b) A close up of the area of maximum throughput.

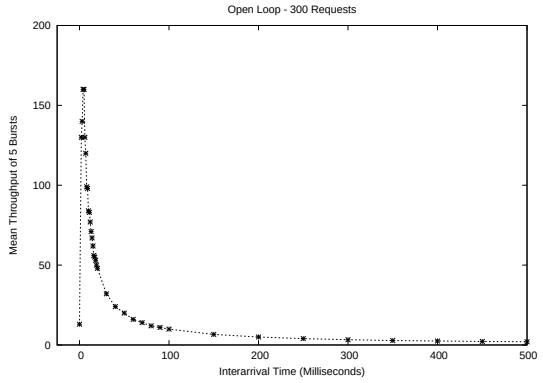


(a)

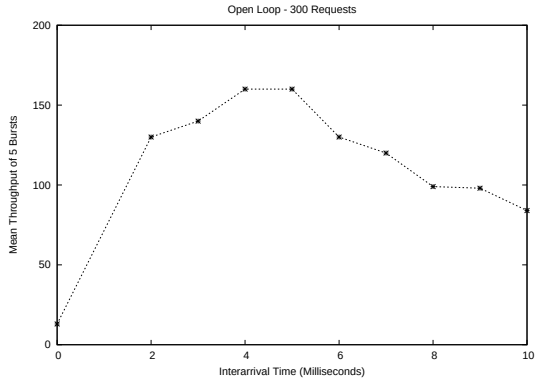


(b)

Figure 7.19: The context server throughput for interarrival times from 500 to 0 milliseconds for a burst of 200 requests. (b) A close up of the area of maximum throughput.



(a)



(b)

Figure 7.20: The context server throughput for interarrival times from 500 to 0 milliseconds for a burst of 300 requests. (b) A close up of the area of maximum throughput.

Chapter 8

Conclusion

Problems with SOA

This thesis identified some potential management problems associated with processes over web services. These problems stem somewhat from the fundamental principles of Service Oriented Architecture, that service participants interact through a contracted interface and ignore what lies behind it. While this approach significantly reduces the complexity of system development, it pushes the system complexity to runtime system management. Most of the identified problems deal with the communication of information relevant to the successful completion of the process between process participants which are not directly connected. These communications could be viewed as providing the context in which the web service process executes. Using a context service is a potential solution to the identified communication problems.

Context Options

There is only one existing context standard in the web services area; WS-Context. It describes a basic context scheme, prescribing the structure and encoding of the context, along with the architecture of a context structure service, a context storage service, and the operations they must provide. The WS-Context specification is spartan, and meant to be extended to implement a particular context type.

WS-RF provides a more comprehensive set of operations, which would be useful in the implementation of a context service, but it is not in and of itself a context specification.

There are context service examples in the literature which are convenience components of the particular system being studied, and attempts to create complete solutions for context aware services, but none have been widely accepted as standards.

WS-Context Implementation and Performance

A basic WS-context server was implemented, and evaluated. The evaluation looked at: the time costs of the basic operations provided by the context service and context manager, the cost of propagating the context between process participants, and the performance of the context server

under stress.

The basic operations test found that there was a time cost in the tens of milliseconds per operation. Most basic operations have a constant time cost, however the *getContent* and *setContent* operations for setting and retrieving the context's content are dependant on the size of context being transferred.

A context propagation test simulated the propagation of the context to two or more process participants. The test ran all services on the same machine, but with separate http servers. There is an appreciable difference (8 milliseconds per propagation) between a service consumption which propagates a context and one which does not. However, the overhead was for a dummy service which did no work.

The stress tests simulated loads on the context service, and attempted to determine the maximum throughput, and the effects multiple concurrent clients on response times. The maximum throughput of the context server was found to be 190 requests per second. The response times were generally unaffected by other clients up to 75 concurrent clients, beyond which, response times began to slowly increase.

Outlook

In the future, the WS-Context specification could be extended to include support for concurrent access to a single context. It could also include a notifications scheme, and context searching facilities. Some hybrid of WS-RF and the WS-Context specifications might be best. The outlined uses for the context service could implemented and evaluated against the identified problems with web services. The context service could be reimplemented as a cloud computing based service to improve its scalability.

While a context is one way to deal with some of the problems associated with processes over web services, it does not seem to be the direction that industry is going. Management systems such as ESBs, rely on observing the documents in transit and inferring their context, rather than having it explicitly communicated. It could be possible to use a context server as an addition to the ESB as there are many cases such as nonfunctional requirements in which useful management directives would not be inferable.

The abstraction of context itself might not be the best approach to this problem of communication between process participants. Perhaps a scheme involving direct communication between process participants, or agents acting on their behalf, would be better.

References

[1] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical report, OASIS, October 2006.

[2] Casey K. Fung, Patrick C. K. Hung, Richard C. Linger, and Gwendolyn H. Walton. Extending business process execution language for web services with service level agreements expressed in computational quality attributes. *Hawaii International Conference on System Sciences*, 7:166a, 2005.

[3] D.A. Menasce and V. Dubey. Utility-based qos brokering in service oriented architectures. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 422 –430, 2007.

[4] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (wsdl) 1.1. W3C Recommendation, mar 2001.

[5] Oasis uddi specification tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec`.

[6] Hugo Hass. Web services message exchange patterns. Technical report, W3C, July 2002.

[7] Diane Jordan and John Evdemon. Web services business process execution language version 2.0. Published on Web Site, April 2007.

[8] Mule esb management console — mulesoft. `http://www.mulesoft.com/management-console-mule-esb`.

[9] Mary Bazire and Patrick Brézillon. *Understanding Context Before Using It*, volume Volume 3554/2005 of *Lecture Notes in Computer Science*, pages 29–40. Springer Berlin / Heidelberg, July 2005.

[10] Mark Little, Eric Newcomer, and Greg Pavlik. Web services context specification (ws-context) version 1.0. Technical Report 1.0, OASIS, April 2007.

[11] Oasis web services composite application framework (ws-caf) tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf`.

[12] OASIS. Oasis web services composite application framework (ws-caf) tc. web, April 2007.

[13] Mark Little, Eric Newcomer, and Greg Pavlik. Web services coordination framework specification (ws-cf). Technical report, OASIS, 2005.

[14] Mark Little, Eric Newcomer, and Greg Pavlik. Web services acid specification (ws-acid). Technical report, OASIS, August 2006.

[15] Mark Little, Eric Newcomer, and Greg Pavlik. Web services long running action specification (ws-lra). Technical report, OASIS, August 2006.

[16] Mark Little, Eric Newcomer, and Greg Pavlik. Web services business process specification (ws-bp). Technical report, OASIS, August 2006.

[17] Mark Little, Jim Webber, and Savas Parastatidis. Stateful interactions in web services - a comparison of ws-context and ws-resource framework. Web Site, apr 2004.

[18] Soraya Kouadri Mostèfaoui, Hannes Gassert, and Bèat Hirsbrunner. Context meets web services: Enhancing wsdl with context-aware features. In *1st International Workshop on Best Practices and Methodologies in Service-Oriented Architectures*, pages 1–14, 2004.

[19] Irene Y.L. Chen, Stephen J.H. Yang, and Jia Zhang. Ubiquitous provision of context aware web services. In *Services Computing, 2006. SCC '06. IEEE International Conference on*, pages 60–68, Sept 2006.

[20] Markus Keidl and Alfons Kemper. Towards context-aware adaptable web services. In *WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 55–65, New York, NY, USA, 2004. ACM Press.

[21] Aisha M. Salama Elsafty, Sherif G. Aly, and Ahmed Sameh. The context oriented architecture: Integrating context into semantic web services. *Semantic Media Adaptation and Personalization, 2006. SMAP '06. First International Workshop on*, pages 74–79, 2006.

[22] Ikuo Matsumura, Toru Ishida, Yohei Murakami, and Yoshiyuki Fujishiro. Situated web service: Context-aware approach to high-speed web service communication. *Web Services, 2006. ICWS '06. International Conference on*, pages 673–680, 2006.

[23] Oasis web services resource framework (wsrf) tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`.

[24] Oasis web services distributed management (wsdm) tc.

[25] Vaughn Bullard and William Vambenepe. Web services distributed management: Managment using web services (muws 1.1) part 1. Published on Web Site, August 2006.

[26] Vaughn Bullard and William Vambenepe. Web services distributed management: Management using web services (muws 1.1) part 2. Web Site, August 2006.

[27] Kirk Wilson and Igor Sedukhin. Web services distributed management: Management of web services (wsdm-mows) 1.1. Web Site, August 2006.

[28] Akil Arora. Web services for management (ws-management). On Web Site., April 2006.

[29] Oasis web services transaction (ws-tx) tc. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-tx`.

[30] Eric Newcomer and Ian Robinson. Web services coordination (ws- coordination) version 1.2. Technical report, OASIS, February 2009.

[31] Eric Newcomer and Ian Robinson. Web services atomic transaction (ws-atomictransaction) version 1.2. Technical report, OASIS, February 2009.

[32] Eric Newcomer and Ian Robinson. Web services business activity (ws-businessactivity) version 1.2. Technical report, OASIS, February 2009.

[33] David Burdett and Nickolas Kavantzas. Ws choreography model overview. Technical report, W3C, March 2004.

[34] Nickolas Kavantzas, David Burdett, Gregory Ritzinger, Tony Fletcher, and Yves Lafon. Web services choreography description language version 1.0. Technical report, W3C, December 2004.

[35] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46 – 52, 2003.

[36] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web services business process execution language version 2.0. Technical report, OASIS, April 2007.

[37] S. Pokraev, J. Koolwaaij, M. van Setten, T. Broens, P. D. Costa, M. Wibbels, P. Ebben, and P. Strating. Service platform for rapid development and deployment of context-aware, mobile applications. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pages –646, 2005.

[38] A. Sashima, N. Izumi, and K. Kurumatani. Location-mediated web services coordination in ubiquitous computing. *Services Computing, 2004. (SCC 2004). Proceedings. 2004 IEEE International Conference on*, pages 109–114, 2004.

[39] M. Brian Blake, Daniel R. Kahan, and Michael Fitzgerald Nowlan. Context-aware agents for user-oriented web services discovery and execution. *Distrib. Parallel Databases*, 21(1):39–58, 2007.

[40] Quan Z. Sheng and Boualem Benatallah. Contextuml: A uml-based modeling language for model-driven development of context-aware web services development. In *ICMB '05: Proceedings of the International Conference on Mobile Business (ICMB'05)*, pages 206–212, Washington, DC, USA, 2005. IEEE Computer Society.

[41] Maria Riaz, Saad Liaquat Kiani, Sungyoung Lee, Sang-Man Han, and Young-Koo Lee. Service delivery in context aware environments: Lookup and access control issues. In *RTCSA '05: Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 455–458, Washington, DC, USA, 2005. IEEE Computer Society.

[42] Joachim Wolfgang Kaltz and Jürgen Zeigler. Supporting systematic usage of context in web applications. In Geoff Sutcliffe and Randy Goebel, editors, *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference*, page 643. AAAI Press, May 2006.

[43] J. Wolfgang Kaltz. Using context-awareness to support user interaction withweb services. In *AICT-ICIW '06: Proceedings of the Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services*, page 155, Washington, DC, USA, 2006. IEEE Computer Society.

[44] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language. Technical report, W3C, February 2004.

[45] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s: Semantic markup for web services. Technical report, W3C, November 2004.

[46] Harry Chen, Tim Finin, and Anupam Joshi. *The SOUPA Ontology for Pervasive Computing.* Whitestein Series in Software Agent Technologies. Springer, July 2005.

[47] Michiaki Tatsubori and Kenichi Takashi. Decomposition and abstraction of web applications for web service extraction and composition. *Web Services, 2006. ICWS '06. International Conference on*, pages 859–868, 2006.

[48] Andrew Harrison and Ian Taylor. Service-oriented middleware for hybrid environments. In *ADPUC '06: Proceedings of the 1st international workshop on Advanced data processing in ubiquitous computing (ADPUC 2006)*, page 2, New York, NY, USA, 2006. ACM Press.

[49] Bart Orriens and Jian Yang. A rule driven approach for developing adaptive service oriented business collaboration. *Services Computing, 2006. SCC '06. IEEE International Conference on*, pages 182–189, 2006.

[50] Zakaria Maamar, Djamal Benslimane, Philippe Thiran, Chirine Ghedira, Schahram Dustdar, and Subramanian Sattanathan. Towards a context-based multi-type policy approach for web services composition. *Data Knowl. Eng.*, 62(2):327–351, 2007.

[51] Edsger Wybe Dijkstra. Cooperating sequential processes, technical report ewd-123. Technical report, 1965.