FastSOA:
Achieving High-Performance Service Oriented Architectures

Michel Kruiskamp

University of Twente
Enschede - The Netherlands

R e l e a s i n g   y o u r   p o t e n t i a l

University of Twente
Enschede - The Netherlands

Master's Thesis

FastSOA: Achieving High-Performance Service Oriented Architectures

Author:	Michel Kruiskamp
University:	University of Twente
Master:	Computer Science
Supervisors University of Twente:	Harold van Heerde & Maarten Fokkinga
Track:	Information Systems Engineering
Organisation:	Logica Arnhem
Division:	Public Sector
Coordinators Logica:	Raynni Jourdain & Martijn Vlietstra
Mentor Logica:	Jan Wolsink

Arnhem. August 2008

Management Summary

Service Oriented Architecture (SOA) is an architectural style that aligns IT with business processes in an enterprise. Currently, Logica is involved in multiple SOA projects. As SOA projects grow, so does the need for more employees that have experience with large SOA environments. In SOA, there is taken care of scalability, accessibility, extensibility and maintainability. However, load and performance often are not taken care of. Inexperience with and unclear use of large grown SOA environments may rise the question what happens with the performance of such SOA environments. This was the reason for the following research:

"Determine whether a Service Oriented Architecture environment suffers from load and performance issues by looking at it from multiple (4) perspectives.

Research how the environment can cope with these possible issues. The results can be used to build high performance SOA environments."

In order to answer the questions that this research goal rises, we looked from four perspectives at SOA, theoretical as well as practical. The theoretical perspectives comprised a literature study and an existing research on SOA performance, the practical part comprised a questionnaire and tests. These observations showed that there is a need for improvement of SOA performance. According to these observations, a lack of good performance is occasionally experienced during SOA projects. The results of our own findings show that the following recommendations can be made:

- *Use SOAP Document Literal Encoding for SOAP messages.*
- *Use DOM for the parsing of small XML document. As the size of the XML documents grow, it is recommended to look at other solutions, such as SAX or StAX. When dealing with large to huge XML documents, it is best to use XML Binding techniques.*
- *Use tool kits such as Visual Studio .NET for developing services. However, mind the choices tool kits automatically make, such as XML parser and encoding style. Where possible, change the default choice according to this advice.*
- *Try to use middleware, e.g. data grids, when reliability and speed is a critical aspect.*
- *Evaluate more often on projects. The questionnaires showed that only a bit more than half of the SOA projects are evaluated. Evaluations can lead discovery of best practices, that can in turn lead to better knowledge of SOA and performance.*
- *Make use of caching. Our proof of concept for caching in ASP.NET C# shows that it is to implement caching in Webservices. Pay attention to the different aspects of caching, such as stale data and time-to-live (TTL) values.*

Using these recommendations for future SOA projects it is possible to achieve higher performance SOA environments than before. For even more insights in high performance SOA environments, this thesis also mentions suggestions for future research.

Preface

For the final part of my study Master Computer Science (MCS) at the University of Twente in Enschede, I have conducted a research leading to the document you are reading now: my final thesis. This research was conducted at Logica Arnhem and started at the first of February 2008. It was quite an endeavour to find a suitable research for the track Information Systems Engineering that I am following for my master title, but looking back at the past half year, I can conclude that I combined the best parts of this track in this research. Using the knowledge I have learned during my study, it was great to be able to put all of it in practice.

The result of this research and this thesis would not have been possible without various people, so I would like to thank all of them for giving me insights, access to information and valuable feedback.

I would like to start by thanking my supervisors at the University of Twente. Harold van Heerde, my first supervisor, really gave helpful feedback during all parts of the research and advice for bringing structuring the thesis. My second supervisor, Maarten Fokkinga, helped me looking thoroughly at the results from the test, so nothing important could be missed.

Besides the supervision at the University of Twente, I was supported by various people at Logica. Raynni Jourdain and Martijn Vlietstra, project leader and project architect of Working Tomorrow Arnhem, guided me during the overall progress of the research, pointed me at critical aspects of this research and helped me finding my own way within Logica. Without their help, a lot of useful information and knowledge would have remained hidden for me. Next, I would like to thank my mentor Jan Wolsink for giving useful feedback , literature and insights into Service Oriented Architectures. Also, I would like to thank my fellow Working Tomorrow students at Logica Arnhem for having a great time. Finally, I would like to thank my girlfriend Karin and my family for supporting me during these six months.

Arnhem, August 2008
Michel Kruiskamp

Table of Contents

List of Abbreviations

ICT	Information and Communication Technology
SOA	Service Oriented Architecture
XML	eXtensible Markup Language
EA	Enterprise Architecture
EAI	Enterprise Application Integrations
BPEL	Business Process Execution Language
WSDL	Web Service Description Language
UDDI	Universal Description Discovery and Integration
SOAP	Simple Object Access Protocol
ESB	Enterprise Service Bus
SAX	Simple API for XML
StAX	Streaming API for XML
DOM	Document Object Model

1 Introduction

This chapter presents the research setting at which the research takes place, gives an analysis of the problem, explains what is meant with performance and introduces Service Oriented Architectures.

1.1 Research Setting

This research takes place at Logica Netherlands, Arnhem. Logica is providing IT and business solutions at international level, employing around 39,000 people across 36 countries. The current mission statement of the company is "*To help leading organisations worldwide achieve their business objectives through the innovative delivery of information technology and business process solutions*".

Their main activities are providing business consulting, systems integration and IT and business process outsourcing across diverse markets, including telecoms, financial services, energy and utilities, industry, distribution and transport and the public sector.

Logica focuses on differentiated, value-added activities critical to the business success of its clients for a competitive advantage. It provides companies with a competitive advantage through the adoption of the following core strengths (Logica, 2008):

- Strategic consultancy
- Systems integration
- Products
- Project management
- Business process outsourcing
- Applications and infrastructure management
- Payroll and human resources outsourcing

Logica consists of six divisions, as can be seen in the lowest part of Figure 1-1. This research project will be done for the division Public Sector.

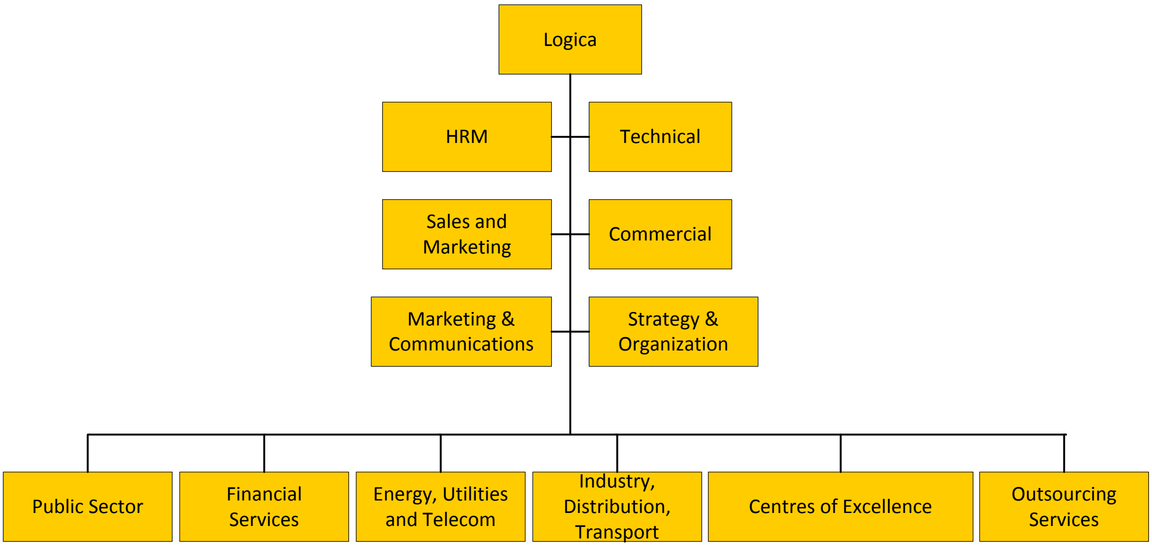


Figure 1-1: Organisation structure, Logica has six different divisions: the lower six blocks.

1.2 Problem Analysis

A quickly emerging development in the field of ICT is Service Oriented Architecture (SOA). The main goal of SOA is serving customers' needs by increased flexibility of IT systems, while reducing costs. SOA gives companies a way to quickly react to the wishes and demands of their customers (SAP, 2006; Josuttis, 2007).

In rapidly changing markets, companies are sensitive to these arguments. SOA gives companies a possibility to rapidly switch of strategy. This gives them a large advantage over other companies.

However, there are also downsides to SOA. There are SOA components that take care of scalability, accessibility, extensibility and maintainability. However, these components do not take care of load and performance. This should not be a problem, if the SOA projects are only used internally: you can make calculations what the maximum load will be. On the other hand, if the SOA project is exposed to an amount of traffic that you cannot calculate in advance (e.g. external web users), it is hard to see what load can be expected, possibly leading to a decrease in performance. Therefore, it is important to try to anticipate to these issues in another way. Since an SOA environment is heavily dependent on XML messaging, it is logical to find a solution for these load and performance issues at the messaging and data part of an SOA environment.

This research will look at SOA and it will examine from four perspectives whether there are any performance related problems in SOA environments:

- A literature study, of which the results are shown in this thesis
- An existing research on SOA performance (Cohen, FastSOA, 2007)
- Questionnaire with employees at Logica
- Tests

Furthermore, this research will look into solutions for such possible performance related issues.

1.3 Conceptual Design

1.3.1 Research Goal

The goal of this research is:

To determine whether a Service Oriented Architecture environment suffers from load and performance issues by looking at it from multiple (4) perspectives. Research how the environment can cope with these possible issues. The results can be used to build high performance SOA environments.

1.3.2 Research Model

To achieve the research goal, it is useful to get an overview of the steps that have to be taken. This can be achieved by the research model. A research model is a schematic overview that shows how a research is structured (Verschuren & Doorewaard, 2000). Furthermore, a research model is important for determining the theoretical background for the research, such as main concepts and theoretical framework. Figure 1-2 shows the research model for this research, according to the guidelines given by Verschuren and Doorewaard.

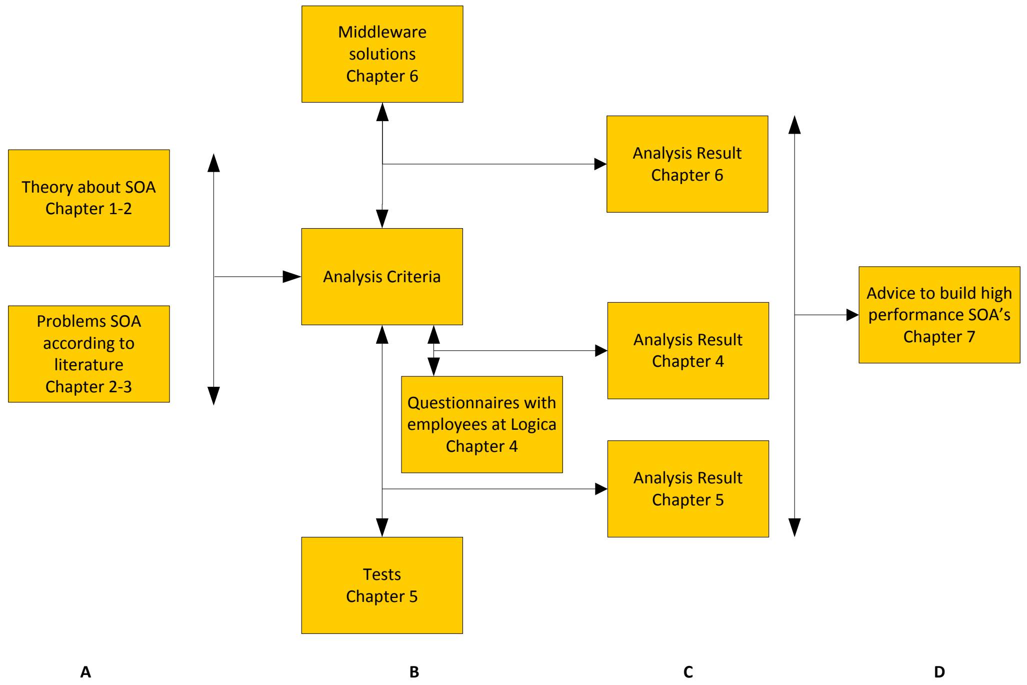

Middleware solutions Chapter 6
Theory about SOA Chapter 1-2
Analysis Result Chapter 6
Problems SOA according to literature Chapter 2-3
Analysis Criteria
Advice to build high performance SOA's Chapter 7
Analysis Result Chapter 4
Questionnaires with employees at Logica Chapter 4
Analysis Result Chapter 5
Tests Chapter 5
A
B
C
D

Figure 1-2: The Research Model that is used during this research. It consists of four stages. Each topic is handled in chapters of this thesis.

To clarify the research model, each step will be explained below.

In the first stage of the research model (A), different literature sources are used: theory about SOA (typical components, advantages, disadvantages, results of existing researches) and problems that can arise in SOA environments. These resources will give enough information to determine the analysis criteria for the rest of the research. Chapters 1 and 2 will show this literature study.

These analysis criteria are later on (B) used for taking a questionnaire with employees within Logica that are or have been involved in SOA projects to get an understanding of, if and how Logica sees the SOA problems and how it handles them. The results of this questionnaire can be found in Chapter 3.5.

As can be seen, stage B involves tests. These tests will be run to see if any possible performance issues can be solved, by carefully making decisions on implementing an SOA environment. Chapter 5 will show these tests and the results. The interviews and tests can also be used for getting practical knowledge about topics being researched and for validating own ideas. After stages A and B are completed, the results from these tests are analysed (C) in Chapter 5.2.1 and 5.3.1. Furthermore, a literature research and a practical research will be performed to see whether there are any middleware solutions for solving possible SOA performance problems in stage C (see Chapter 6). Using the results from stages A, B and C, an advice can be given to Logica that can be used to build high performance SOA environments. This advice can be found in Chapter 7. By giving this advice, the research goal will be achieved. Chapter 8 gives a discussion on SOA and this research. Finally, Chapter 9 does suggestions for future research on SOA performance.

1.3.3 Research Questions

The previous sections gave a quick overview of the structure that the research project will have. This section will describe in more details which questions have to be answered to reach the research goal. In order to reach this goal, the most important question will be the following:

"Does an SOA infrastructure suffer from load and performance issues?"

Since this is a very broad question, the main question can be split into several sub-questions, that can in turn be split into yet other sub-questions. Each question is answered in a chapter of this thesis.

1. What is Service Oriented Architecture?
    a. What are the basics of an SOA environment? (Chapter 2.2)
    b. What are the capabilities of an SOA environment? (Chapter 2.1)
    c. How do organisations benefit from an SOA environment? (Chapter 2.3)
2. Does an SOA environment suffer from load and performance issues? (Chapters 3, 4 and 5)
    a. How are performance and load defined and how can it be measured?
    b. When and where possibly do these issues arise?
    c. What are the possible consequences of these issues?
3. How can XML technology be used in SOA environments, leading to larger load tolerance and reaching higher performance? (Chapter 6)
    a. What options can be given to increase the performance of an SOA environment and load tolerance?
    b. How can performance increase be measured and how much performance increase can be achieved by using this XML technology in comparison with SOA projects that build upon typical current designs and components?

1.4 Technical Design

1.4.1 Knowledge Sources

In order to perform a research in a structured way, it is also important to look at the knowledge sources that will be used in the research. According to Verschuren and Doorewaard there are five different types of knowledge sources, namely people, media, reality, documents and literature (Verschuren & Doorewaard, 2000). This research project will use some of these resources, as mentioned in the following sections.

1.4.2 Literature

The research model shows us that literature will be used for this research. Using literature it is possible to get an understanding of what SOA is and whether it has to cope with load and performance issues. Literature is the first perspective (see the research goal in Chapter 1.3.1). Chapters 1 and 2 show this literature study.

1.4.3 Existing Research

For this research, the results of an existing research by Frank Cohen(Cohen, FastSOA, 2007) will be used to look at the possible problems of SOA. This is the second perspective. Chapter 3 shows this existing research.

1.4.4 Questionnaire/People

As can be seen in the research model, a questionnaire will be used. This questionnaire will give a clear overview of the problems that Logica possibly has faced in the field of SOA. Different SOA project members have to fill in the questionnaire to get sufficient information. This questionnaire is the third perspective and it will be shown in Chapter 3.5.

1.4.5 Testing/Reality

Another way that will be used to perform research for this project, is actual testing. This will take place at stage B of the research (see Figure 1-2). These tests make up the fourth perspective. Chapter 5 shows these tests.

1.5 Performance of Information Systems

This research is about performance of SOA environments. Performance is also an important aspect of information systems in general. It can be seen as an indication of how the system performs and it refers to a system’s ability to achieve a response within a certain amount of time. A clear way to show performance of an information system is the throughput of the system. This can be measured by the transactions per second (TPS) measurement. Using the TPS measurement, it is possible to show how well an information system is able to perform as it receives more and more requests (or: if the load on the system increases) (Cohen, FastSOA, 2007).

In the ideal situation, an information system is capable of handling requests regardless of the amount that comes in at the same time. However, typical information systems have a bottleneck: additional requests they receive, slow down the speed at which the system responds to the requests, decreasing the transactional speed. For this reason, it is important that organisations are able to predict how an application will behave under increasing load. One may think that this behaviour can be calculated by multiplying the resources one user uses by the number of users that actually are using the system at the same time. However, this *prediction by multiplication* is usually not applicable due to concurrency: a user action (e.g. a write request) can have effect on other user action whether negative (e.g. the write request can slow down other read requests) or positive (e.g. a read request can speed up other similar read requests).

1.6 Enterprise Architecture

An Enterprise Architecture (EA) is a conceptual method that organisations can use to understand their own structure and the way they work. It provides a map of the enterprise and can be used as a for future decisions about changes in business and technology. Enterprise Architecture is about understanding all of the different elements that make up the enterprise and how those elements interrelate. To visualise the relations between these elements, various frameworks have been developed. One of the most famous frameworks is the Zachman Framework, developed by John Zachman in 1987 (Zachman International, 2008).

A common term in IT community that can be heard often in combination with Enterprise Architecture is SOA. Major IT vendors are talking about SOA, promoting their view and products based on SOA. The following sections and chapters will explain what SOA is.

1.7 SOA and EAI

Many people consider SOA as the successor of Enterprise Application Integration (EAI). EAI is an integration technology that supports enterprise business processes with the existing application portfolio. The goal of all EAI products is interoperability: making it possible to send data and commands between applications. The applications get more possibilities and become part of a bigger system. On the other side, the goal of SOA also is to let applications communicate with each other, but SOA does not use the current applications in order to reach its goal: it hides the existing applications and exposes a set of application-independent services. SOA projects often start with EAI, the longer an SOA environment exists, the more functionality will end up in the services itself (van der Lans, 2006).

1.8 SOA Definitions

Before SOA was a common term in the IT community and broadly used, enterprises were mainly using siloed, closed end-to-end applications: each IT-part of a business process was accomplished by a specific application. However, as markets change, increasing the need for flexibility, reuse, and fast time-to-market delivery, another approach for alignment between business and IT was needed . This is where SOA comes into play. SOA is a new paradigm in the world of IT that allows this alignment between business and IT. It provides a way to integrate dissimilar applications within an enterprise, improving reuse of applications logic while eliminating duplication of production environments. All this is possible through the use of single or combined service delivering units. There are various kinds of communications and coordination, restricted by agreements and contracts between these units.

The OASIS (Organisation for the Advancement of Structured Information Standards) group, a non-profit consortium that aims for open standards for the global information society, has given a clear formal definition for SOA (OASIS, 2006):

> *"Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations".*

The Open Group, a consortium neutral in both vendor and technology that aims for information integration within and among enterprises using open standards and global interoperability, also gives a definition for SOA (The Open Group, 2008):

> *"Service-Oriented Architecture is an architectural style that supports service orientation. Service orientation is a way of thinking in terms of services and service-based development and the outcomes of services. An architectural style is the combination of distinctive features in which Enterprise Architecture is done or expressed."*

Plenty other definitions can be found, but they tend to differ a lot, focus on different aspects and the question remains whether SOA can stick to the promises implicitly made in these definitions. To give a better view on what SOA is, we will explain the different aspects of SOA in the next chapters.

2 SOA Levels and Technologies

The previous chapter gave some definitions of SOA. In these definitions various terms (such as services) were used. This chapter will show us how SOA can be presented in different levels of abstraction. It gives an example of each abstraction level and an example of all three abstraction levels combined and It explains some of the terms that appeared in the SOA definitions from the previous chapters. Furthermore, it will show what are the typical components of SOA.

2.1 SOA Levels of Abstraction

An SOA environment can be presented in three levels of abstraction: operations, services and business processes (Zimmerman, Krogdahl, & Gee, 2004). Operations are transactions that represent single logical units of work. An execution of such an operation will typically cause one or more persistent data records to be read, written or modified (e.g. *'read number of items in stock'*). Services represent logical groupings of operations (e.g. *'ship product'*). Various definitions can be found for services within the scope of SOA. Some of these can be found in Table 2-1.

Similarly to objects and components, a service is a fundamental building block that combines information and behaviour, hides the internal workings from outside intrusion and presents a relatively simple interface to the rest of the organism.	(Sprott & Wilkes, 2004)
Services are self describing, platform-agnostic computational elements that support rapid, low-cost composition of distributed applications.	(Papazoglou, 2003)
A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners.	(Hao, 2003)
A service represents some functionality (application function, business transaction, system service, etc.) exposed as a component for a business process.	(Dodani, 2004)

Table 2-1: Various definitions can be found for services (within the scope of SOA)

It is important to note that a service is often in alignment with a business service of the company that uses the SOA environment: most business services have an SOA equivalent in terms of one or multiple services.

Generally, the following rules apply for services (Evdemon, 2005):

- A service has a clear boundary
  Services are separated from each other and the outside world by well-defined boundaries. A boundary represents the border between a service's public interface and its internal implementation. Interaction between services takes place by passing messages over these boundaries: the only way to interact with a service from the outside is by sending a message and the only way a service can interact with the outside is by sending a message, as shown in Figure 2-1.

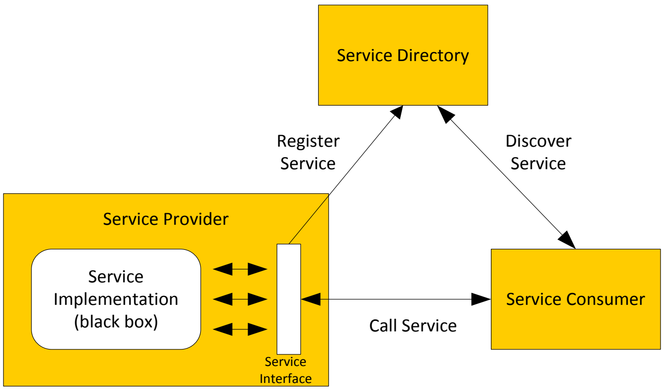


Figure 2-1: Each service has a clear border. As can be seen, interaction with the service (provider) has to pass a certain boundary and is only possible through its interface.

- A service is autonomous
  Services are units that are independently deployed, versioned, and managed. Even the further development of a service is independent of other services. While services are in this way designed to be autonomous, no service is isolated from the other services because interaction between autonomous services is possible.
- Services only shares schema and contract
  Service interaction should be based only upon a service's policies, schema, and contract-based behaviours. It is not of any use that a service knows all 'ins and outs' of other services. A service's contract is generally defined using a language called Web Service Description Language (WSDL), which will be explained later on. Contracts for aggregations of services can be defined using a language called Business Process Execution Language (BPEL). BPEL uses WSDL for each service aggregated (see Chapter 2.2.1). Services communicating using XML schema-based messages are non-dependent of both programming languages and platforms, increasing the level of interoperability. Schema defines the structure and content of the messages, while the service's

contract defines the behaviour of the service itself. Service consumers will rely upon a service's contract to invoke and interact with a service. This has as a consequence that a service's contract must remain stable over time: contracts should be designed as explicitly as possible. Once such a contract has been published, it becomes very difficult to modify it, while minimising the consequences for any service consumer that is currently working with the contract.

- Collaboration between services is based on policies

  It is not possible to communicate some requirements for service interaction in WSDL alone. Policy expressions can be used to separate structural compatibility (what is communicated) from semantic compatibility (how or to whom a message is communicated). Note that there is a structural difference in knowing what a service actually does with a message you sent (generally, a service consumer does not need to know this) and knowing the level of quality of the service and the operational requirements of the service (a service consumer needs to know this), such as under which conditions will the service accept the sent message. Such operational requirements for service providers can be in the form of machine-readable policy expressions.

The third and last level of abstraction are business processes: a long running set of actions or activities performed with specific business goals. Typically, such processes consist of multiple service invocations (e.g. *'Customer Order'*).

Figure 2-2 shows a visualisation of the levels of abstraction using a business process called 'Customer order'. This figure gives a good view of the hierarchical structure of the levels of abstraction.

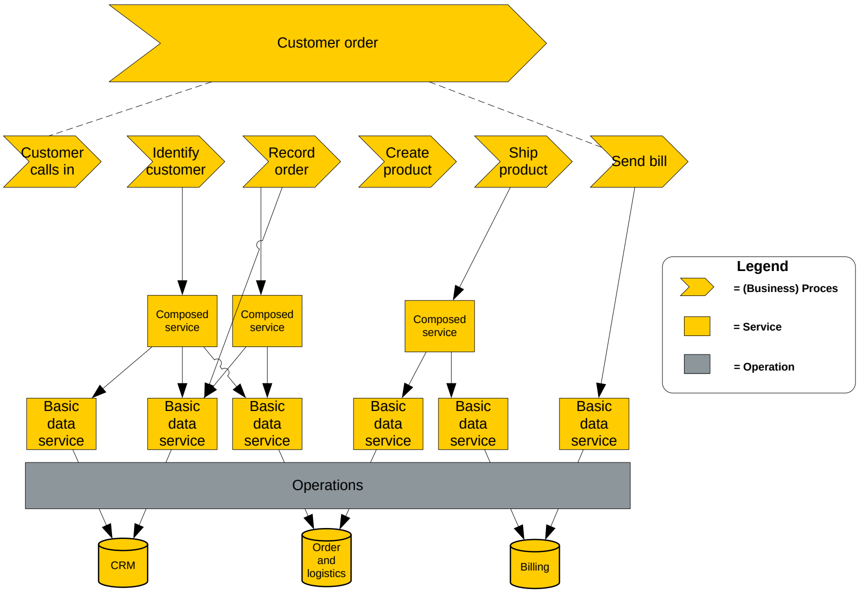


Figure 2-2: An SOA environment can be presented in three levels of abstraction: operations, services and business processes.

2.2 SOA Technologies

Since there is no standard for SOA environments, it is hard to explain how an SOA environment can be implemented. Two things that people often think are the same, are SOA and Web Services. What must not be forgotten, is that SOA is an architecture and Web Services are just one of the many (for example, other ways are REST and CORBA) ways to implement SOA. However, WSDL, SOAP and UDDI are the most popular type of technologies today to build services for an SOA environment and Web Services (Papazoglou, Mike P.; Heuvel, Jan-Willem van den, 2007). WSDL is used to describe services, UDDI to find and register the services and SOAP a transport mechanism to send messages between the service consumer and service provider. Figure 2-3 shows the relation between the service roles. The following sections will explain each component in more detail.

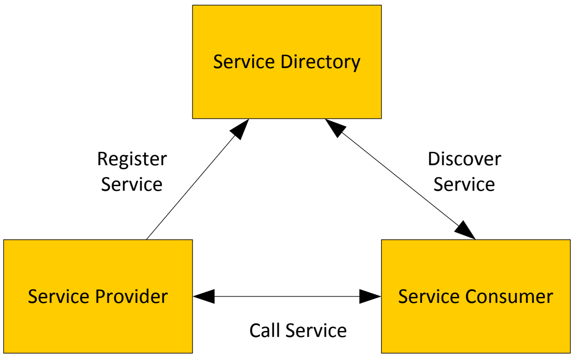


Figure 2-3: The relations between Service Consumer, Service Provider and Service Directory. Services register themselves at the service directory, service consumers call service providers and services are discovered by service consumers through the service directory (Josuttis, 2007).

2.2.1 Description: WSDL

The Web Service Description Language (WSDL), a standard that is submitted to the World Wide Web Consortium, describes (web) services by using a model and an XML format (World Wide Web Consortium, 2007; Alonso & Casati, 2004). These specifications are made of two parts: an abstract part and a concrete part (see Figure 2-4). Separating the abstract part (service interface) from the concrete part, allows the reuse of these interfaces: different services are able to implement the same interface while providing the service at different addresses.

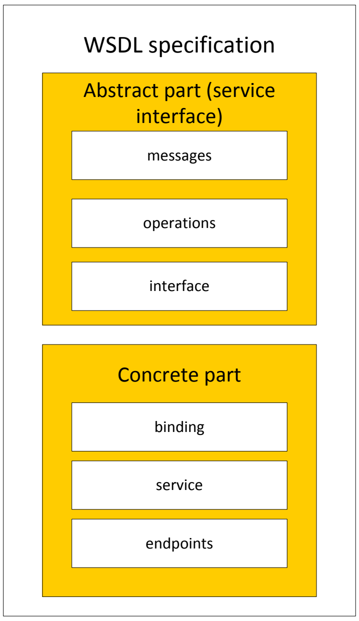


Figure 2-4: The structure of a Web Service Description Language file. It consists of two parts (abstract and concrete), enabling reuse of the interface.

2.2.2 Discovery: UDDI

One of the fundamental components of an SOA environment is a mechanism for services to be discovered by potential service consumers. Figure 2-3 shows this relationship between service consumers and the actual services. For the discovery of a service by a service consumer, a central directory to host service descriptions is required. The Universal Description Discovery and Integration (UDDI) specification has takes care of this. The primary goal of UDDI is the specification of a framework for describing and discovering services, maintained by OASIS (Alonso & Casati, 2004). The core of an UDDI registry (implementation of the UDDI specification) is made of a business registry, a naming and directory service. UDDI defines data structures and APIs, which can be used to publish service descriptions and to query the registry to look for descriptions.

2.2.3 Messaging: SOAP

As mentioned above, the WSDL describes the messaging protocol(s) that must be used to access the service. In an SOA environment, this is most of the time the Simple Object Access Protocol (SOAP), a platform independent standard messaging protocol for Application-to-Application interactions. SOAP defines how to organise information using XML in a structured way so that it can be exchanged between peers. SOAP specifies the following (Alonso & Casati, 2004):

- One-way communication message format that describes how information can be packaged into an XML document.

- Descriptions for using SOAP messages to implement Remote Procedure Call. These descriptions explain how clients can invoke a remote procedure using SOAP messages and how services can reply using SOAP messages.
- Rules what any entity receiving the message should follow, such as the XML elements that an entity should or should not read and understand.
- Description of how to carry a SOAP message as payload by other protocols, such as HTTP.

A SOAP message is a SOAP XML document instance. An advantage of the fact that messages are transported over HTTP, is that this transport easily passes through firewalls (it can be also a disadvantage in terms of security). As can be seen in Figure 2-5, a SOAP message is used as an envelope in which the information to be communicated is enclosed. An envelope consists of two parts: a SOAP header (optional) and a SOAP body (mandatory). Both the header and body can have multiple sub-parts: header blocks and body blocks. Every message has a sender, an ultimate receiver and an arbitrary number of intermediaries (*nodes*). Each of these processes the message and routes it to the (ultimate) receiver. The header contains additional information primarily meant for the nodes, such as transactional interactions, security, routing and debugging. Intermediaries should only process and manipulate this header, and not the SOAP body, since the body is the payload: the actual message being conveyed. Note that if there are no nodes, the header can be omitted.

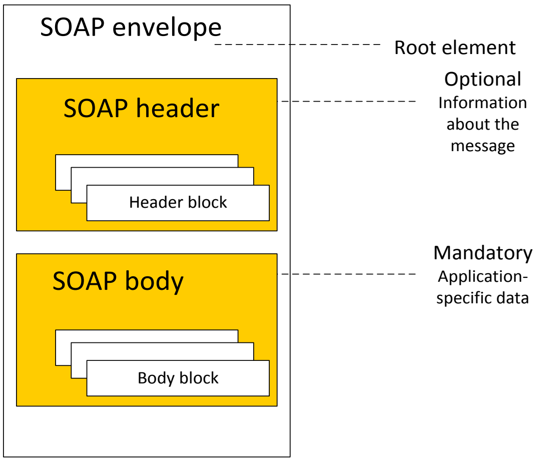


Figure 2-5: The structure of a SOAP message. It consists of two parts (header and body), the first is optional, the second mandatory (Alonso & Casati, 2004).

SOAP messages can be constructed in two ways: Document interaction style and RPC interaction style. When using the Document interaction style, the two interacting applications come to an agreement about the structure of documents exchanged. SOAP messages are used to transport these documents between these two applications. When using RPC interaction style, the messages are constructed in a different way: the body of a request message contains actual calls to a procedure (name of the procedure and input parameters), and the body of the response message contains the result and output parameters. See Figure 2-6 for a schematic overview of the different interaction styles.

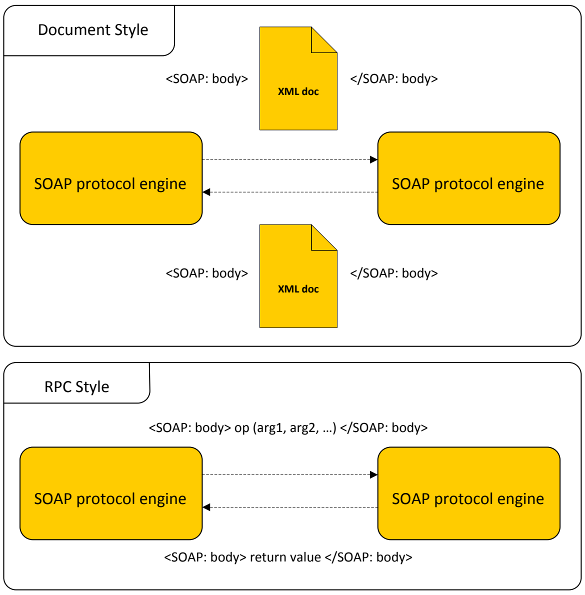


Figure 2-6: There are two SOAP interaction styles: document style and RPC style. The first sends complete XML documents, while the second only sends procedure calls (Alonso & Casati, 2004).

Encoding rules also influence the structure of a SOAP message. They define how a particular entity or data structure is represented in XML. SOAP 1.2 gives a particular form of encoding called ‘SOAP Encoding’ (also called ‘Section 5 Encoding’), that defines how data structures, including basic types and complex types, can be serialised into XML. SOAP 1.2 does not enforce any specific encoding form, so applications are free to choose their own encoding (Alonso & Casati, 2004).

2.2.4 ESB

The previous section explained SOAP: A messaging protocol for Application-to-Application interactions that can be used by service consumers and providers to communicate. There has to be some kind of infrastructure that makes all this possible. This infrastructure can be of the following form: Point-To-Point (see Figure 2-7) or connection through a bus (see Figure 2-8).

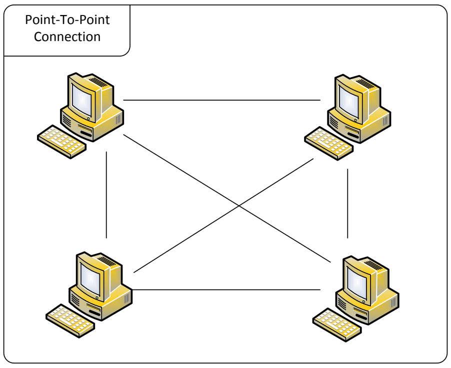


Figure 2-7: A Point-To-Point Connection between systems is very inefficient: each systems connects to each other system.

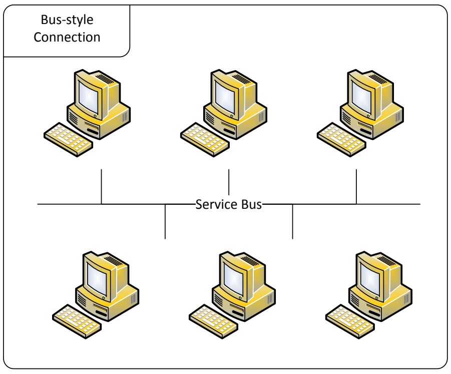


Figure 2-8: The Bus-Style Connection is efficient: a bus is used that routes the messages from one system to another system.

Traditionally, information system infrastructures used the Point-To-Point Connection method. With this connection method, every point in the information system is connected to every other point. This means that if a point, (e.g. an application) is added to the system, the number of connections grows very fast, since each point has to connect to the new point (see Figure 2-7). Clearly, this raises serious performance and scalability issues and the need for a 'backbone' that can manage the complexity of how a service consumer connects and communicate with a service provider. The Enterprise Service Bus (ESB) gives a solution for these problems. Providing interoperability and routing are the main roles of an ESB. Data transformation is an important aspect that is a direct result of the first role (providing interoperability). An SOA environment integrates different platforms and programming languages, so it is necessary the ESB is able to transform messages (data) to the right format.

There are two forms of routing, namely service routing and intelligent routing. Service routing means that the ESB is able to send a service call from a consumer to a provider, and then sending an answer back from the provider to the consumer. This way, the service consumer does not need to know the exact endpoint/location of the service provider. This is called location transparency.

Intelligent routing is a bit more complicated and is not a standard ESB capability (but a so-called value added service). From a business point of view, it is sometimes necessary to route messages in a different way. For example, different routing is needed if different messages have different priorities, or messages need to be dealt with according to their content. This type of routing is called 'content based routing'.

2.2.5 Service Composition

In an SOA environment, some services can be composed of other services. This is called *service composition*. One of the reasons for this is the need for a structured service hierarchy when the number of services in an organisation grows. In extensive projects, individual services are often orchestrated by a composition language, such as the Business Process Execution Language (BPEL). Business processes are often composed of individual services. An example of such composed services is a banking application: for each transfer of money between banks, the money has to be withdrawn from one account and be paid at another account. This can be accomplished by one service (called *transfer_money*) that is made of two other services (*withdraw_money* and *pay_money*).

Business processes are modelled and run by business process languages. BPEL is one of these languages. It is becoming the standard for defining and running business processes (OASIS, 2007). BPEL is an XML-based language for specifying executable business processes, that are themselves (composed) services. This can be achieved by using *basic description elements* and *advanced description elements.* Basic description elements are message exchange elements such as *invoke, receive, reply* and flow of control elements such as *sequence, switch, loop*. Advanced description elements offer for example compensation and event handlers, rollback operations, etc.

2.3 Advantages of SOA

As mentioned before, SOA has become an important term in the world of IT. According to the vendors of SOA solutions, SOA offers an enterprise lots of advantages (Arsanjani, Zhang, & Ellis, 2007):

- Reduced costs
  Reusability of services enables a reduce of costs. Costs for software development and maintenance can be kept low by reusing existing services for other applications and hence saving development and maintenance time.
- Increased flexibility
  Most of the business processes in an enterprise are supported by IT. The change of a business process always results in a change of the IT infrastructure. Consequently, enterprises try to keep business processes as long as possible without change. Building business processes using services as in SOA, the impact of a time-consuming business process change dramatically decreases. This way flexibility is increased.
- Added business agility
  SOA changes businesses to adaptive enterprises, where IT can respond to constantly changing business needs much more quickly and can do so with fewer resources. Business agility is increased because of the following reasons:
  - Loose coupling of services

The use of loose service coupling has service independence as a result. This independence frees a service from immediate links to other services. This way, reuse is easier to achieve.

    - Service abstraction

      Service abstraction enables reuse because it makes services appear as a black box: services are primarily viewed in terms of input and output. Processing details are hidden and service consumers only know how to access the service by a public service interface.

    - Service composition

      Reuse is primarily possible because of service composition. Because services are designed for reuse, it is possible to make new services by composing existing services.

    - Service statelessness

      The statelessness of services supports reuse because it enlarges the availability of a service and makes services more generic, reducing state management.

- Increased consolidation

  Often enterprises face the problem of how to handle with legacy systems that are vital for business processes. Using SOA, these legacy IT systems can be integrated across applications and organisations.

2.4 Disadvantages of SOA

The careful reader may have noticed the that the words “according to the vendors of SOA solutions” were used in the previous section and the question is whether SOA sticks to it promises that are made in several definitions. Reading the advantages of SOA in the previous section and the information about SOA published by the major IT vendors, it may seem that SOA is the holy grail in information system engineering. However, there is a downside to the SOA concept: a research performed by the UvA (University of Amsterdam), IBM and Sogeti shows SOA does not always stick to its promises(Berg, Hompes, & Truijens, 2007). This research involved taking interviews about SOA in practice at 12 large organisations, including KLM, Reed Elsevier and the Belastingdienst (Dutch IRS). All these organisations are currently involved or were involved in trying to make a small step towards an SOA architecture. These organisations make a good representation of the type of organisations that are using or planning to use SOA environments. Questions that were asked were about the increase in business flexibility, reusability, complexity, return on investment, consolidation of legacy systems and business/IT alignment. All the organisations had quite a high expectations about SOA: 10 of the 12 organisations expected large increases in flexibility and reusability beforehand. However, the results from the interviews showed otherwise: eight organisations found that complexity was increased by SOA (67%), while only four organisations (34%) experienced a decrease in complexity (see Chart 2-1).

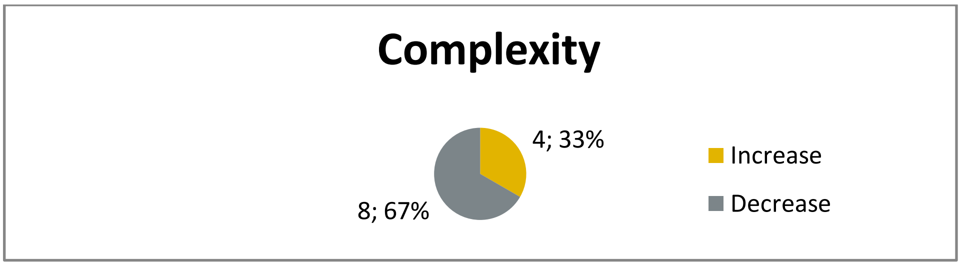


Chart 2-1: As expected, more than half of the organisations experienced a decrease in complexity

Another point worth mentioning is that technological complexity greatly increased: organisations were not familiar with SOA, so expertise was missing. Most of the organisations (seven out of 12, 58%) mentioned reuse was increased by SOA, so SOA keeps up with the promise of reuse (see Chart 2-2).

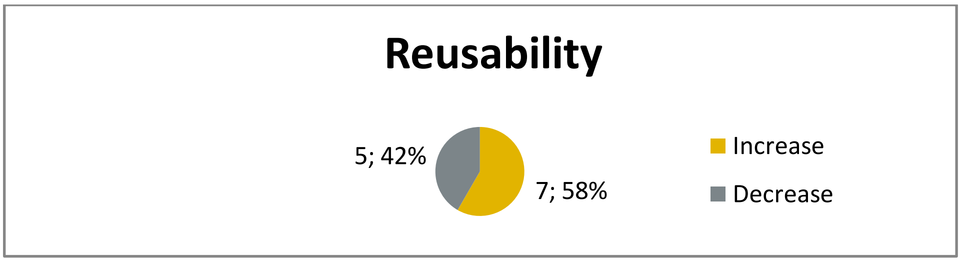


Chart 2-2: Seven organisations experienced an increase in reusability.

The research showed that 41% (five organisations out of 12) experienced an increase in flexibility, but 17% (two organisations) experienced a decrease in flexibility. The other five organisations did not notice an increase in flexibility or could not decide yet whether flexibility increased or decreased (see Chart 2-3).

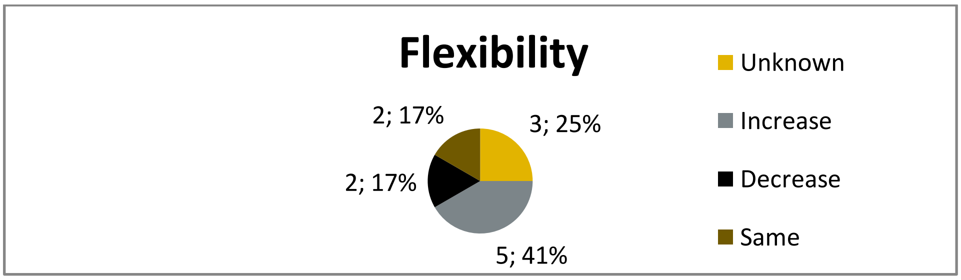


Chart 2-3: Less than half of the organisations experienced an increase in flexibility

From the technological point of view, the main advantages that SOA offers, according to eight organisations, is standardisation. However, the main features of SOA big vendors promote, such as scalability and interoperability, are only mentioned by a few as technological advantage.

As this research shows, SOA is clearly not the holy grail for information system engineering according to these 12 organisations. SOA does not keep up with the high expectations of all the promises SOA vendors make. It is important to note that all 12 organisations were only involved in SOA on a small scale: it is hard to already see the results of the promises SOA vendors make.

2.5 Summarizing SOA

Service Oriented architecture (SOA) is an approach to structure enterprises and aligns business and IT. It provides a way to structure the business processes and the IT components of enterprises.

Vendors of SOA solutions promise to offer enterprises a lot of advantages and benefits, like shorter time-to-market and increased flexibility. However, studies show that these promises are not always fulfilled.

An SOA environment can be represented in three levels of abstraction: operations (e.g. *'read number of items in stock'*), services (e.g. *'ship product'*) and business processes (e.g. *'Customer order'*). Services are made of the basic components, operations. Business processes are made of services. The focus of SOA is, like the abbreviation says, placed on services. Typically, each service in the SOA environment is similar to real services the enterprise offers. For example, a room booking service that a hotel offers to its customers will be implemented by one SOA service. Interaction between services and service consumers is based on exchanging messages. There are three service roles. These are the service provider, the service consumer and the service directory.

The following components are typically used to build an SOA environment:

- A description of the service, typically made in the Web Service Description Language (WSDL).
- A mechanism for services to be discovered, typically made using the Universal Description Discovery and Integration (UDDI) register.
- A protocol for messages to be exchanged, typically the Simple Object Access Protocol (SOAP).
- A mechanism to regulate the exchanged messages, typically the WS-Policy framework.
- An infrastructure for delivering and routing the messages, typically an Enterprise Service Bus (ESB).

Besides all the promises that SOA vendors make about the advantages of SOA, a recent study shows that SOA does not always come up to the expectations.

3 Possible causes for low SOA Performance

The previous chapters explained what SOA is, the advantages of SOA and how SOA works. This chapter looks at the performance of SOA. It gives three possible bottlenecks or causes for low SOA performance, according to an existing research.

3.1 XML and SOA according to Frank Cohen

This research focuses on this question by looking at it from four perspectives, as mentioned before in the Problem Analysis in Chapter 1.2:

- A literature study
- An existing research on SOA performance (Cohen, FastSOA, 2007)
- Questionnaire with employees at Logica
- Tests

The literature study on SOA can be found in the previous chapters: what is SOA, how is it achieved and what are the experiences with SOA?

The remainder of this chapter looks at the existing research on SOA performance by Frank Cohen, the author of multiple articles on IBM DeveloperWorks (IBM, 2008). Next chapters will look at the research question from the two remaining perspectives, namely a questionnaire within Logica and tests.

The self describing aspect of XML encourages the use of XML in information systems: to use XML, you are not bound to certain structural tags or other constraints, but can define your own tags and data definitions. As a result, when you say SOA and messages, you say XML. Most SOA environments use XML to exchange messages and access services. Business loves SOA for the advantages mentioned in Chapter 2.3 and developers love XML's simplicity. Unfortunately, the combination of SOA with XML can lead to serious performance and scalability problems, reducing the transactions per second of an SOA environment when load and requests increase. For this reason, it is important to ask some questions when looking at performance of SOA environments and its services, such as the following:

- Who or what will be accessing the services?
- When will the services be accessed?

- How often will the services be accessed?
- Where will the services be accessed from?

Frank Cohen looked into this subject of SOA and XML and wrote a book on his research that keeps these questions in mind, called FastSOA (Cohen, FastSOA, 2007). In this book he proposes a new approach to SOA and discusses the common traps than can be avoided by following a set of software development patterns when building an SOA environment. According to the FastSOA research, performance issues can emerge, as described in the following sections.

3.2 Cause 1: SOAP Encoding Styles

Chapter 2.2.3 introduced a type of encoding rules: SOAP Encoding (also called Section 5 Encoding). It defines how a particular entity or data structure is represented in XML. Messages that are encoded using SOAP Encoding, mainly use the RPC interaction style (see Chapter 2.2.3). The messages that are encoded this way and use this interaction style are called SOAP RPC Encoded messages.

Another kind of encoding is Literal encoding. An additional requirement to use this encoding is that the Body contents of the SOAP messages have to conform to an XML Schema. Messages that are encoded using Literal encoding mainly use the Document interaction style, and are called SOAP Document Literal Encoded messages.

According to Frank Cohen’s research on SOAP encodings, a developer’s choice of encoding style greatly determines the performance and scalability of a service. When the payload size of a SOAP message increases, this is shown even more. Chart 3-1 shows the difference according to Frank Cohen’s research and measurements.

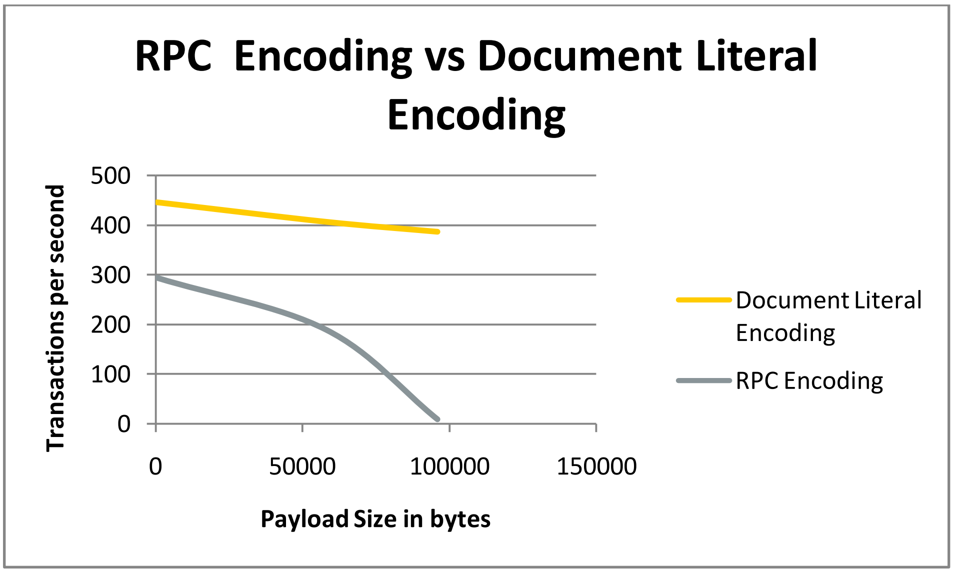


Chart 3-1: There is a large difference between the two types of SOAP encoding according to Frank Cohen's research. Where RPC Encoding almost plummets to zero when the payload size of a message increases, Document Literal Encoding only slightly decreases.

A question one can ask is why a developer does not always choose the best encoding style. There are multiple answers to this question. Often developers just choose the encoding style that is the easiest for them to implement. This saves the developer a lot of code-writing, time, and thus costs. Another reason is that vendors of SOA solutions, such as IBM and Microsoft, tend to favour one encoding style. For example, IBM WebSphere Application Developer uses SOAP RPC Encoding by default, while Microsoft .NET uses SOAP Document Literal Encoding by default (Cohen, FastSOA, 2007).

3.3 Cause 2: Java SOAP Bindings

Large enterprises often have spent a lot of money and time on their IT investments. As a result, as enterprises move on by extending their infrastructure with SOA, they want to keep and expand their existing IT investments. One of these investments most organisations have done, are application servers: a machine or its software that works in conjunction with a web server to deliver application services. The best known and most used application servers are Java Enterprise Edition (JEE) application servers. Among the vendors that support these JEE application servers are BEA, IBM, Oracle, SAP and Apache Tomcat. The widespread use and investments of these application servers make developers often use tools of these JEE application servers to build SOA environments. Most of these servers provide a utility that asks for a WSDL service definition file as input and outputs a Java class SOAP binding, also known as a proxy (see Chapter 3.3).

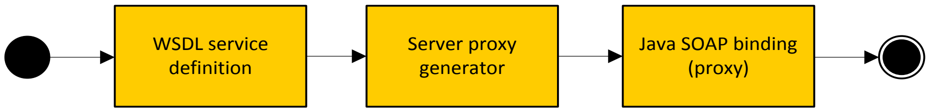


Figure 3-1: Some tools require a WSDL service definition as input and generate a Java SOAP binding (proxy) that can be used by service consumers to call a service (Cohen, FastSOA, 2007).

This binding or proxy is a reference to services and can be used to call these services as if the services can be found locally. The proxy extracts the XML content from the SOAP service call (deserialises the service call) that originates from the service consumer and instantiates a Java Request object that contains the SOAP message body contents. This Request object is handled by the Enterprise JavaBeans (EJB) to get a response to the actual service request (see Figure 3-2). EJB technology is the server-side component architecture for JEE (Sun Microsystems, 2008). It makes use of application modules and can be used by application servers.

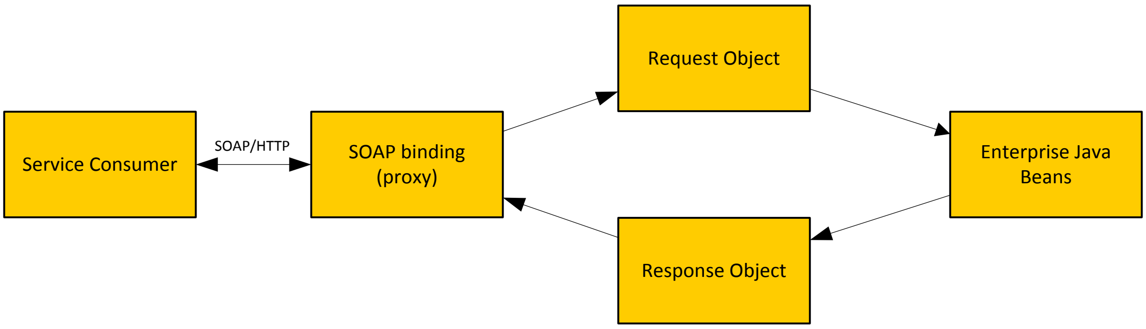


Figure 3-2: With a Java SOAP binding as generated in Figure 3-1, handling a SOAP request messages means lots of overhead transforming it into request and response object (according to Frank Cohen)(Cohen, FastSOA, 2007).

According to Frank Cohen’s research, SOAP bindings generated by application server tools are inefficient and very slow. His research shows that throughput went down from 15-20 TPS (at 10 kB payload) to 1.5 TPS (at 100 kB payload). The fact that services can be reused for service composition (see Chapter 2.2.5) decreases the performance of the SOAP bindings even more. For each consecutive service call, the request and response has to be deserialised and serialised. Java SOAP bindings can be said to increase user convenience, but at the same time decrease performance.

3.4 Cause 3: XML Parsers

Another choice developers have to make regarding building an SOA environment, is what XML parser to use. Every XML message that is sent and received first has to be parsed. After this, operations can be performed on it. Generally, there are two main technologies for XML parsing: event-based parsing and object model parsing (SAX, 2001). Event-based parsers, such as the

Streaming API for XML (StAX) and Simple API for XML (SAX) types, read an XML document from beginning to end and each time it encounters a syntax construction, it returns an event to the application that is running it. This application has to handle the event. There is a fundamental difference between StAX and SAX parsers. The first are called pull parsers and the latter are called push parsers (Sun Microsystems, 2005). The reason for this is as follows: pull parsers are called this way because client applications call the methods in the parsing library only when it needs to interact (the client pulls XML data when it explicitly asks for it). Push parsers are called this way because these kind of parsers send XML data to the client as the parser encounter elements (it pushes the data to the client, whether or not the client is ready to use the data at that time).

Object model parsers, such as DOM, do not only read the data but also construct an in-memory (tree) representation of the document, which can be altered (Intel Software Network, 2007). The choice when to use what type of parser, heavily depends on the type and amount of data to parse (we return to this in Chapter 5.)

Table 3-1 (Sun Microsystems, 2005) shows these and other differences between the types of parsers. Another approach, that has the same goal as parsing (breaking an XML document up in logical pieces), is the transformation of XML to (e.g. Java) content objects. Using this technique, it is possible to access and process XML data without having to know XML or XML processing (Ort & Mehta, 2003).

	StAX	SAX	DOM
API Type	Pull, streaming	Push, streaming	In memory tree
Ease of use	High	Medium	High
CPU and Memory Efficiency	Good	Good	Varies
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes

Table 3-1: Comparison of XML parser types

3.5 Summary

According to Frank Cohen, there are three possible causes for low SOA performance: SOAP Encoding styles, Java SOAP bindings and XML parsers. Frank Cohen states that by making careful decisions during the implementation of an SOA environment and considering each of these possible causes, a large performance increase can be gained.

4 The Logica Approach to SOA

In order to find out how Logica approaches SOA projects and which choices they make during these projects, a questionnaire about the outcome of Frank Cohen's research is taken with employees at Logica. This chapter explains the questions in the questionnaire, shows the and gives an analysis of these results.

4.1 Introduction

In order to get to know whether Logica sees the issues mentioned in Chapter 3, a questionnaire was placed on the Internet (see Appendix A). The results of this questionnaire will be used to validate or reject the possible causes of low performance in SOA environments according to Frank Cohen, namely SOAP encodings, XML parsers and Java SOAP bindings (see Chapter 3).The people that filled in this questionnaire are all employees at Logica that are or were involved in one or more SOA projects or projects in which XML messaging played a large role. A lot of attention was paid to the selection of the respondents. All the respondents were IT/software architect or software engineer. The reason for this selection, is that these employees have the most experience with and knowledge about SOA.

Before placing the questionnaire on the Internet and sending out the invitations to the respondents, the questions were validated by my supervisors and a Software Architect within Logica. By analyzing the answers to the questions in the interviews, I could get an overview of how Logica approaches such SOA projects. The structure of the questionnaire can be found in Appendix A.

In statistical researches, it is important to look at data reliability and validity. Reliability means that the answers that are given in the questionnaire will be the same when the questionnaire is repeated with the same respondents. Validity refers to the extent to which the data gives a true measurement: are the answers really true? (McNeill, 1990) However, the questionnaire part of this research is not a statistical research. The answers that are given in this research are used only to get an understanding about the view Logica and its employees have on the possible causes for the problems mentioned in this thesis. To get such a view, the proportion between the different answers will be enough. Therefore, it is not required that the given answers are 100% reliable and valid.

4.2 Respondent Projects

In total, 64 respondents have taken part in the questionnaire. Some of the answers that were given were not usable for this research due to being vague or not complete. As a results, answers of 52 different respondents were used. Examples are projects that required one or two Web Services, the coupling of various services offered by a Dutch railroad company, applications for selling insurances of various insurance companies and alarming systems for Dutch ministries.

4.3 SOAP Encoding Styles

There are two styles that can be used to encode a SOAP message: SOAP RPC Encoding and SOAP Document Literal Encoding (see Chapter 2.2.3). According to Frank Cohen, the SOAP encoding style that is used for an SOA environment can have a large impact on the performance (see Chapter 3.2). For this reason, a first question that was asked to the respondents was whether they have taken the different SOAP encodings into account (*During this project, were SOAP encoding styles taken into account?*) The answers that were given to these question were very different, so they had to be categorised. The following categories were made:

- No, we did not look at the various encodings at all (for an unspecified reason) (*NU*).
- No, the customer that ordered the project gave the preference for a specific encoding (*NC*).
- No, we used the encoding style that was 'default' for the tools we use (e.g. Microsoft BizTalk) (*NT*).
- Yes, the encoding was chosen according to our own preferences and expertise or the W3C standard WS-I Basic Profile 1.0 (*YO*).
- Do not know (*DNK*).

The distribution of the given answers can be found in Chart 4-1.

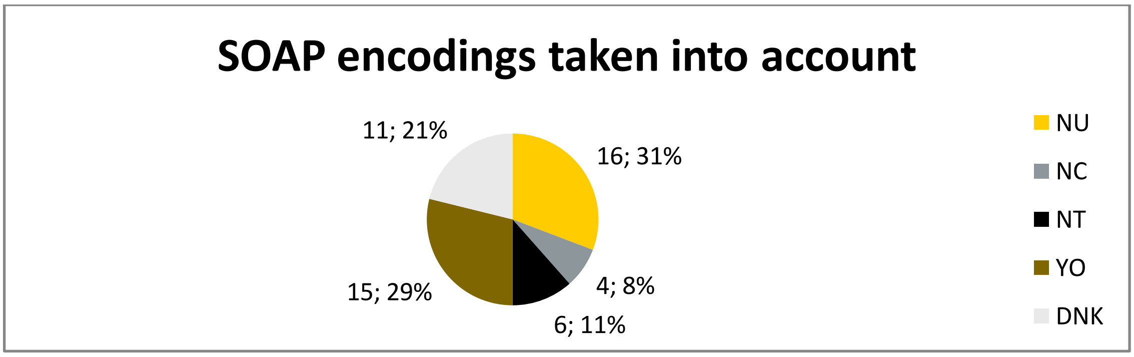


Chart 4-1: Answer distribution for question 1. The answers have been grouped into five categories.

As can be seen, the largest group of respondents answered to the question that they did not look to the different encoding styles at all (category NU): 16 out of 52, or 31%. Furthermore, the diagram shows that the three categories NU, NC and NT together comprise 50% (26 answers). This means that in 50% of the projects, SOAP encodings are not taken into account. For 21% (11 projects), the respondents do not know whether SOAP encodings are taken into account. A possible reason for this is that these respondents are not engaged in these decisions, because of their function within Logica. The remainder of the respondents answered positive to the question: the W3C WS-I Basic Profile 1.0 was used to determine the SOAP encoding (Document-Literal) or the expertise of the developers was the deciding factor for this choice. Because of the diversity of these answers, further research on the performance difference between the SOAP encodings is needed. In Chapter 5.2, tests are performed on the two types of SOAP encodings that will look at the performance of both.

4.4 Out-of-the-box Tools

The second question in the questionnaire was whether any out-of-the-box tools, that can be used to ease the development of services, were used in the project (*During the project, did you make use of any out-of-the-box tools to ease the development of the services? For example to build SOAP bindings (proxies)?*) . According to Frank Cohen's research, (Java) SOAP Bindings that are often generated by such tools, can be the cause of performance issues (see Chapter 3.3). The answers that were given were categorised into the following categories:

- Yes , because of the ease of development using such tools and the standard of using it.
- No, the project was too small of size.
- Do not know.

The distribution of the given answers can be found in Chart 4-2.

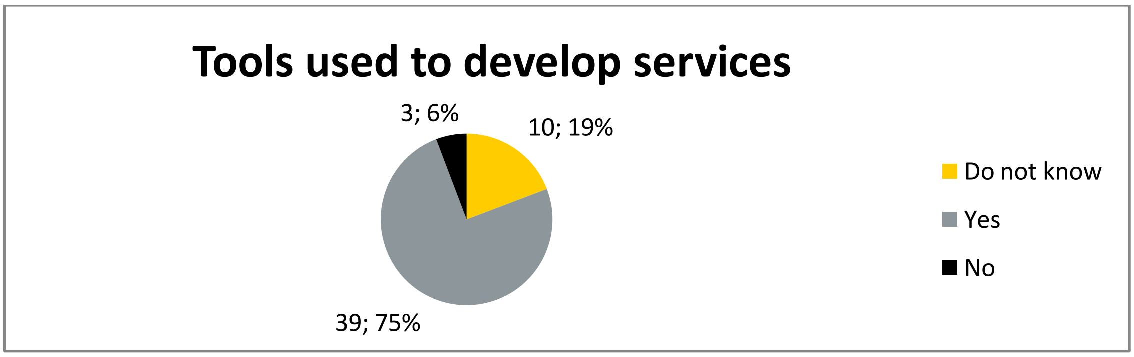


Chart 4-2: Most respondents answered 'Yes' to the question whether tools were used to develop services.

The largest part of the respondents gave a positive answer to the question whether they have used or are using tools to ease the development of services. When looking more precise at these answers, the following reasons were given:

- Better quality, when using tools, less coding errors are made
- Speed of development, using tools speed up the process of development

- Standard part of toolset (e.g. Microsoft Visual Studio)

Where these answers may differ at first sight, it is highly possible that the intention behind the answers is the same, e.g. a toolset may be used in order to develop better quality products or increase the speed of the development. Therefore, all these answers are grouped in the same group in the diagram. When looking closer at the three no answers, it becomes clear that the reason for not using these tools is the size of the project. With smaller project, less tools are used. As mentioned above, the largest part of the respondents gave a positive answer to the question whether they have used or are using tools to ease the development of service. Looking at the reasons that are given for this choice, it becomes clear that these tools really improve a lot of aspects. For this reason, this research does not look further into performance of SOAP Bindings/proxies that are generated and the recommendation is to stick to using such tool kits, as it drastically improves speed and decreases the amount of errors.

4.5 XML Parsers

As mentioned before in Chapter 3.4, according to Frank Cohen, the different types of parser deliver different performance, depending on the situation. Therefore, the third question in the questionnaire was whether the different types of XML parsers was taken into account and what the motivation was behind the choice of parser (*What considerations were made for choosing an XML parser?*). Respondents gave answers that are grouped into the following categories:

- No, not taken into account because the tool manages all the parsing (*NT*).
- No, we used our own parser (*NO*).
- No, the customer that ordered the project gave the preference for a specific parser(*NC*).
- Yes, and we choose a parser for specific reasons (*Y*).
- Do not know (*DNK*).

The distribution of the given answers can be found in Chart 4-3.

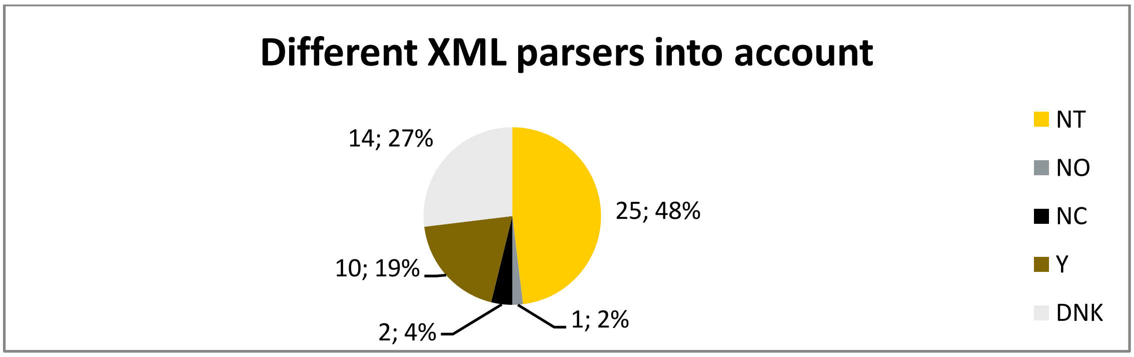


Chart 4-3: Distribution of the given answers for the question whether XML parsers were taken into account. As can be seen, most of the time they are not.

One answer really stands out among the others. 48% of the respondents answered that they did not look at the different type of XML parsers, but that they simply let the toolkit choose the parser and do the parsing. The reasons they gave for this were mainly the same as mentioned before in Chapter 4.4: better quality, speed of development and standard part of toolset.

When looking at the 'other no'-answers, two reasons can be seen. For one project, a self-made solution was used to parse data. The remainder of the respondents gave as reason that the customer decided on the type of XML parser.

Only 19% of the respondents answered that they did look at the different type of XML parsers. The motivations for and use of a specific parser that were given are the following:

- Speed of parsing, as mentioned in Chapter 3.4 and before in this chapter, different parsers perform different, depending on the situation.
- Expertise, developers have different experience with and knowledge of the different types. For example, developers prefer one type over another type, since the first is easier to implement.

Another kind of 'parsing' is transforming XML to content objects. For example, the Microsoft toolkit supports this, so for some answers this fits into the *NT* category. By using this XML to content objects transforming, a user does not have to have any knowledge about XML processing. There is also a Java approach for directly transforming XML to objects, this is called the Java Architecture for XML Binding (JAXB) (Ort & Mehta, 2003). Figure 4-1 shows how it works.

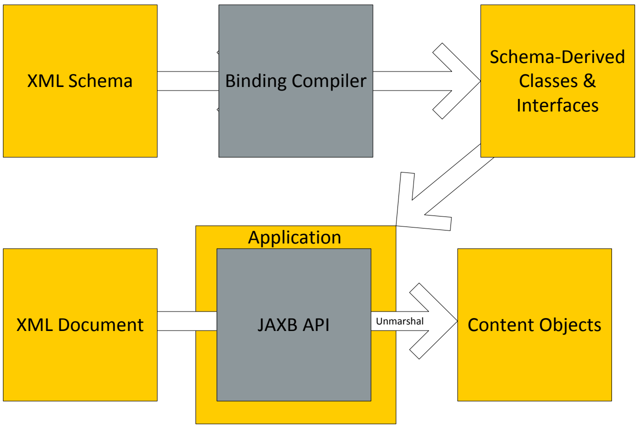


Figure 4-1: Mechanism of transforming XML to objects using JAXB

First, an XML schema (a document that defines the structure the XML document has to conform to) is used by the JAXB Binding Compiler to generate classes and interfaces that are derived from the XML schema. Next, the XML document is unmarshalled into Java content objects. These content objects are instantiations from the schema derived classes and interfaces. These objects can then be used in applications, as if the document is parsed using a SAX, StAX or DOM parser. The great advantage of all this, is that there is no need to create a parser (such as SAX, StAX or DOM) or write callback methods (Ort & Mehta, 2003).

As mentioned above, 48% of the respondents answered that they did not look at the different type of XML parsers, but that they simply let the toolkit choose the parser and do the parsing. Looking at these numbers, it is worthwhile to look further into the difference in performance between the types of XML parsers. In Chapter 5.3, tests are performed on the performance difference between the different types of XML parsers. In this section, recommendations will be given when to use what type of XML parser.

4.6 Middleware

The literature study in the early chapters of this thesis showed that there are various middleware products that can be used to develop an SOA environment (see Chapter 2.2). To see whether there were any middleware products used to increase the performance of such an SOA environment, the questionnaire contained a question whether some of these middleware solutions were used in the projects that respondents participated in (*Did you make use of any middleware to improve performance during the project?*) Only two types of answers were given:

- No
- Do not know

The distribution of the given answers can be found in Chart 4-4.

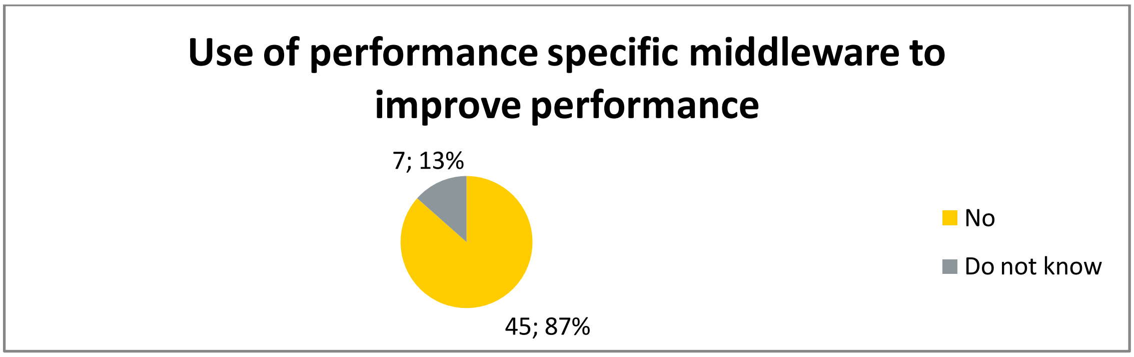


Chart 4-4: Distribution of the given answers for the question whether performance specific middleware was used. As can be seen, most of the time it is not the case.

It is clear that most of the respondents answered that they did not or do not use any performance specific middleware. When looking at the complete answers including explanation for the answers, it becomes clear that lots of other middleware was and is used in most projects: almost every respondent answered that middleware such as application servers and Enterprise Service Buses were used. However, these kinds of middleware are used to ease the development, deployment and maintainability of the SOA, and not to improve the performance in a way like data grids and caching do. None of the respondents answered ‘yes’ to the question. This gives an option to increase the performance of the SOA, e.g. by using solutions as mentioned in Chapter 6.

4.7 Implementation Problems

It is important to know whether any implementation specific problems were encountered during the projects the respondents were or are involved in. Using this knowledge, advice can be given on certain choices that are encountered during the implementation. Furthermore, the result of this question can be used to determine whether the statements that Frank Cohen made in his research are true or not. The following question was included in the questionnaire: *Did any problems arise during the project in terms of implementation technical issues (performance, reliability and scalability)?*

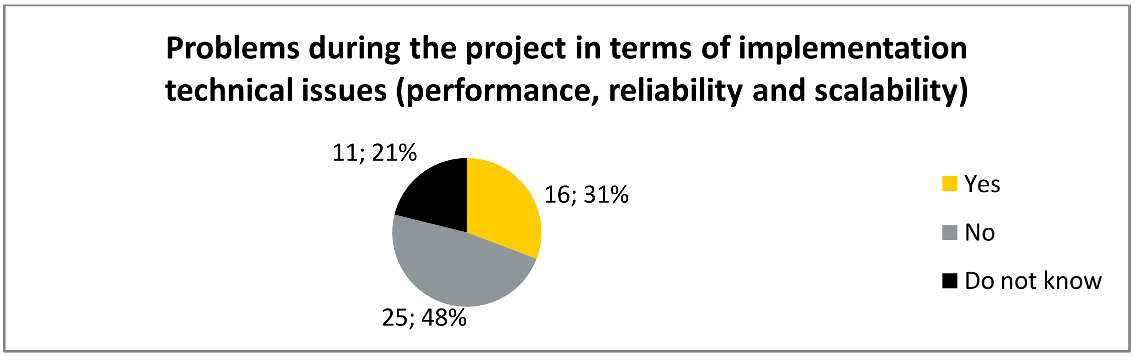


Chart 4-5: Distribution of the answers to the question whether any problems were encountered during the projects. For just only 31%, problems were not encountered.

The distribution of the given answers shows that nearly one third (31%) of the projects encountered implementation technical issues in terms of performance and scalability. Almost half of the respondents (48%) answered that no implementation technical issues were encountered. The remainder of the respondents could not say whether any problems were encountered.

When looking closer at the 'yes' answers, the following causes for these problems were given:

- Large and too many XML messages
- Processor intensive services (calculations)
- Thoughtless system architecture

The first cause was mentioned most often and the third cause was mentioned least.

When looking at the causes for low performance according to Frank Cohen, we can see a correspondence with the causes for the low performance as given by the respondents: both Frank Cohen and the respondents point at XML for being a cause of low SOA performance. Moreover, these observations point out that the XML part of SOA can be important for performance.

4.8 Evaluation Moments

At the end of a project, it is a good thing to evaluate. Using evaluations, new insights and new information can be gathered to improve the project (The National Science Foundation, 2002). For this reason, the respondents were asked whether there had been any evaluation moments after the project they were or are involved in using the following question: *Have there been any evaluation moments after the project?* Chart 4-6 gives an overview of the results of this question.

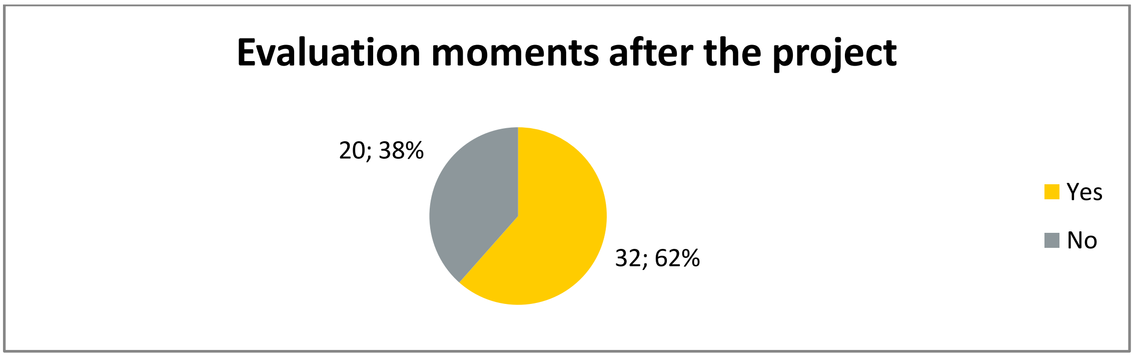


Chart 4-6: Distribution of the answers to the question whether there have been any evaluation moments after the project. For most of the projects, this was the case.

For most of the projects that the respondents were involved in, there have been evaluation moments (62%). Still, for more than one third of the projects, there have not been evaluation moments, so improvement is certainly possible. Using evaluation moments, one can get to know best-practices, than later on can be used in other similar projects. As a results, these similar projects can show higher performance.

4.9 Best-Practices

Another question that was asked to the respondents was whether any best-practices (methods and procedures found to be most effective) came forward out of the projects (*Did any best-practices come forward out of this project?*). Using these best-practices, it is possible to learn from the projects and take the findings along to other projects. Chart 4-7 gives an overview of all the answers.

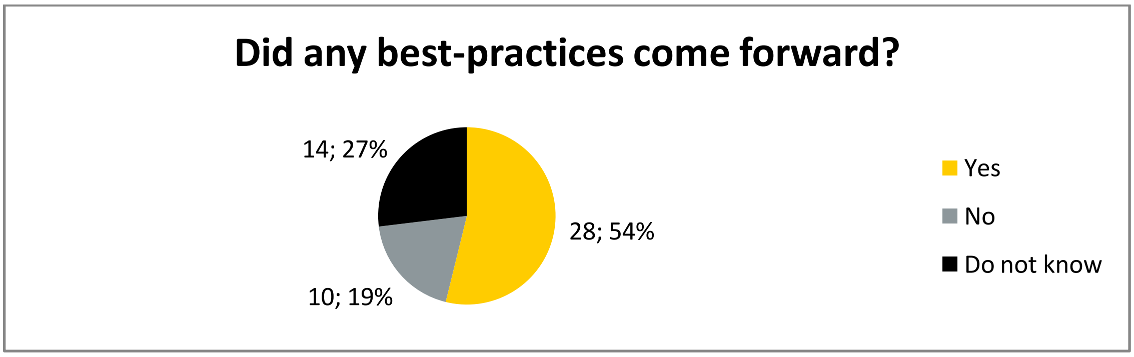


Chart 4-7: Distribution of the answers whether best-practices came forward out of the project. For most of the projects, this was the case.

For more than half of the projects, best-practices were found, such as:

- Configuration of the hardware environment
- Testing should start early

- SOAP Document Literal Encoding should be used, because of interoperability
- Keep it simple, use the business as SOA initiator, not IT

Another large group of respondents could not give answer to the question whether any best practices came forward: 27% or 14 out of 52 respondents. A possible reason for this, is that the respondents that gave this answer don't have the appropriate knowledge about the project to give a positive or negative answer to this question.

4.10 Questionnaire Conclusion

When looking closer at all the questions and the corresponding answers, the following can be concluded:

- *The different SOAP encodings are most of the time not taken into account, if they are, most of the time SOAP RPC Document Literal Encoding is chosen. According to the best practises, this also is the best choice, because of interoperability.*
- *For almost every project, frameworks and tools are used to implement SOA environments.*
- *For most projects, the different XML parsers are not taken into account. If they are taken into account, the size of the messages is often the deciding factor for choosing the parser.*
- *Middleware to specifically increase the performance of SOA environments and services is almost never used.*
- *For about a third of the projects, implementation issues occur. The main reasons for these issues are large and too many XML messages.*
- *For more than half of the projects, there have been evaluation moments afterwards. This could*

5 Tests

In Chapter 3, Frank Cohen mentions three possible reasons for low SOA performance, namely SOAP Encoding styles, Java SOAP bindings and XML parsers. The previous chapter showed the questionnaire and its results that were used to find out whether Logica sees these SOA performance problems.
This chapter looks from a different perspective at SOA performance, by performing tests.
This chapter shows how the tests are structured, the results of these tests and analyses of these tests. These analyses will look at the results of the tests and will compare these results with the results of Frank Cohen's research, this way validating or rejecting his reasons for low SOA performance. Furthermore, by looking at the results of these tests and the questionnaire, advice can be given on how to make certain decisions during SOA projects.

5.1 Introduction

The results of the questionnaire that was taken with employees at Logica, showed that projects implementation problems arise when working with SOA and/or XML messaging. Furthermore, Frank Cohen found out in his research that XML messaging in SOA environments can lead to serious performance issues (see Chapter 3).

Using these results and the possible reasons for performance problems as stated by Frank Cohen, the following tests were selected:

- Testing performance difference between the SOAP encodings
- Testing performance difference between the XML parsers

There is a specific reason why the performance of Java SOAP Bindings/Proxies is not tested: the results of the questionnaire show that Logica uses toolkits and application servers (so also Java SOAP bindings) for almost every (Java) project. The reasons that were given for this, are mainly speed of development and better quality. For this reason, the probability that such toolkits and applications servers won't be used anymore, is very small, so Java SOAP Bindings/Proxies are not tested in this research.

The tests have been performed on two personal computers with the following specifications:

Personal Computer 1:

- CPU: Intel Pentium 4, 2.80GHz

- Memory: 2GB DDR RAM
- Operating system: Microsoft Windows XP Professional, SP2

Personal Computer 2:

- CPU: Intel Pentium 4, 2.80GHz
- Memory: 1GB DDR RAM
- Operating system: Microsoft Windows XP Professional, SP2

5.2 SOAP encodings

As mentioned before in Chapter 2.2.3, there are two styles of SOAP encoding: SOAP RPC Encoding and SOAP Document Literal Encoding. Furthermore, we saw in the results of the questionnaire that in most projects, the fact that there are multiple SOAP encodings is not taken into account. However, the choice of SOAP encoding can have large impact on the SOA performance according to Frank Cohen. To see whether the encoding of SOAP messages has any influence on the performance of services, performance tests were performed, so that the results of these tests can be used later on for SOA projects. The following components were involved in these tests:

- BEA WebLogic Server 7.0. This server is used to deploy a test Web Service that is extensively requested by concurrent test agents, running on Personal Computer 2.
- BEA WebLogic Workshop. This program that is part of the BEA WebLogic Product Family is used to create and deploy the test Web Service that is deployed on the server, running on Personal Computer 2.
- Test Web Service for WebLogic Workshop. This service is deployed on Personal Computer 2.
- PushToTest TestMaker 4.3.1. This is a free, open-source, Java testing automation program that can be used to record user actions that can be later on reproduced in order to perform tests. I used TestMaker to run agents that request the Test Web Service (PushToTest, 2008). TestMaker ran on Personal Computer 1.
- WebLogic Workshop Encoding Kit Test Agent. This kit that consists of test agents is used by PushToTest TestMaker 4.3.1 to request the Test Web Service, measure the number of transactions (performance). This test agent ran on Personal Computer 1.

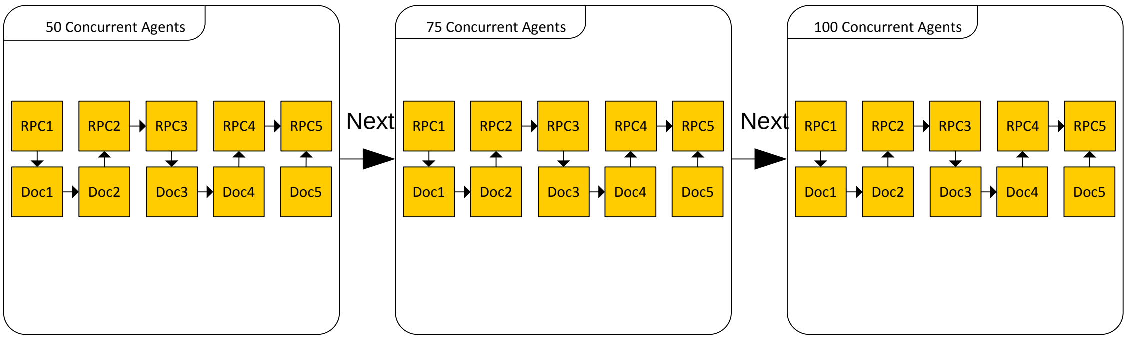


Figure 5-1: Test scheme for the SOAP encoding tests. As can be seen, the tests are run for three amounts of agents (50,75 and 100), each test ten times.

The Test Web Service is a service that returns a number of words, that are randomly chosen from a set of words. This service was deployed at the WebLogic Server 7.0 and accepts messages (requests) encoded in both the SOAP RPC style as the SOAP Document Literal style. When such a message arrives at the Test Web Service, it inspects the content of the request. This content has to be the number of words that the service consumer wants to retrieve from the Web Service.

As can be seen in Figure 5-1, the performance tests that were run were done in three phases: 50 Concurrent Agents, 75 Concurrent Agents and 100 Concurrent Agents. These three phases give sufficient different results. Each phase consists of 10 test runs, five SOAP RPC Encoding test runs and five SOAP Document Literal Encoding test runs (resulting in average values). Each test run started with requesting a relatively small number of words (500) and ended by requesting relatively large number of words (4500), increasing four times by 1000 words. By measuring the number of transactions that are completed in each test run and dividing this number by the total time each test took in seconds, we get the transactions per second (TPS). A transaction in this case is seen as the complete request and respond cycle and it is measured as can be seen in Figure 5-2. The results and an analysis of these tests can be found in Chapter 5.2.1.

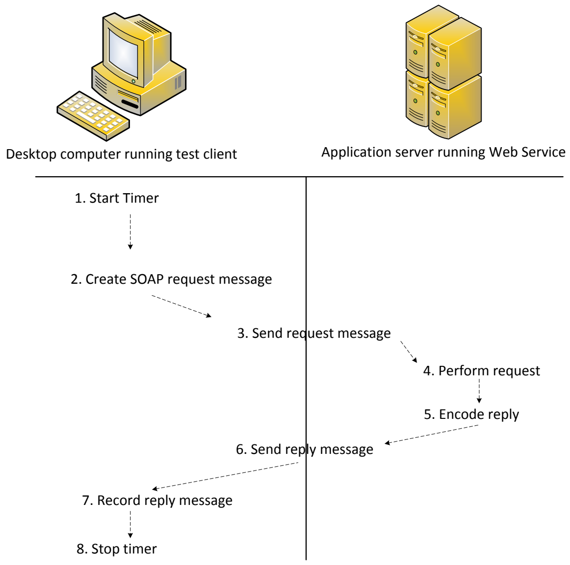


Figure 5-2: Timing measuring for the encoding tests. The timer is started at the very beginning and it ends after the results have been logged.

5.2.1 SOAP encodings Test Result Analysis

Chart 5-1 shows an overview of the SOAP encoding performance test that is explained in the previous section. These results are for 50 concurrent agents, running both the WebLogic Server and the test client on a local machine. The remainder of the results can be found in Appendix B.

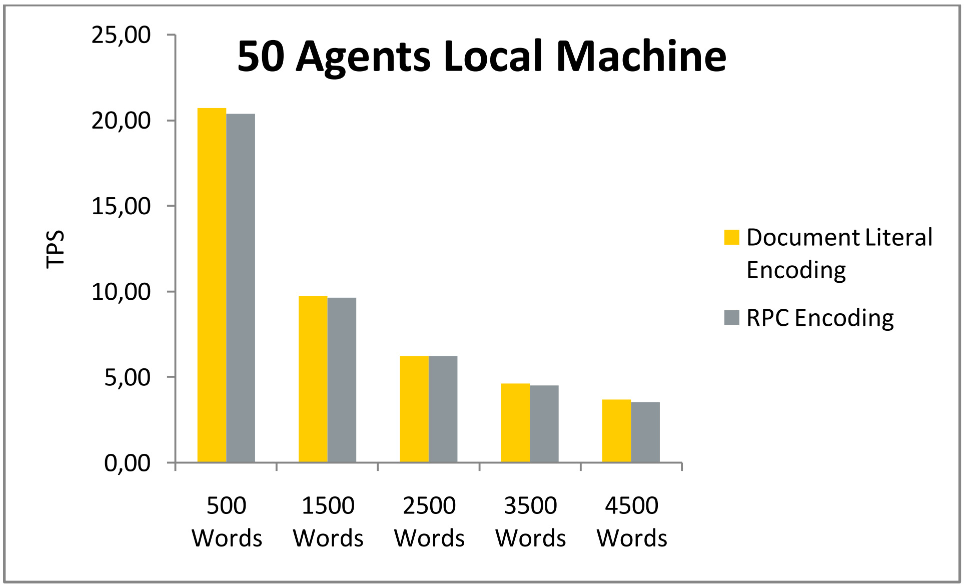


Chart 5-1: Results of the performance tests for 50 agents, running the server and the test client on one machine. The difference between SOAP Document Literal and SOAP RPC are negligible

The performance of the service decreases very fast as the number of requested words grows for both SOAP encodings. Moreover, these results of the performance tests show that the differences between performance using SOAP RPC encoding and SOAP Document Literal encoding are negligible, though document style seems to lead to slightly better performance. There are some points (mainly when using 75 agents, running all components of the test on a local machine), where SOAP RPC encoding performs better than SOAP Document Literal encoding.

Frank Cohen shows otherwise, as Chapter 3.2 and Chart 3-1 show. According to his research, the encoding style of SOAP messages has a large impact on the performance of the service , where SOAP Document Literal Encoding performs much better when the payload increases. This in contrast with the results of our own tests, where both encoding styles deliver decreasing performance. In order to rule out any mistakes/errors that were made during the tests, such as network traffic, computer usage and configuration of the test client, the test has also been performed in two other settings/test environments:

- Separating the WebLogic Server and the test client, by using two different machines and testing the services through a normal network connection.
- Separating the WebLogic Server and the test client, by using to different machines and testing the services through a direct UTP connection, outside of a network.

One could expect that different test environments will lead to other results, since the resources for running the WebLogic Server can be spent on other tasks. However, as the numbers show (see Appendix B), there is no significant difference between the results of either of these two test environments and the environment where the WebLogic Server and the test client run on the same machine, still getting complete different results than the results of Frank Cohen.

Looking closer at the results, it looks like that using a direct UTP connection results in better performance for SOAP RPC when the number of words (SOAP message payload) increases. For this reason, 10 more test runs have been performed (five for SOAP Document Literal and five for SOAP RPC, all requesting 5500 words). This upper value was chosen for a specific reason: the higher the number of words that are requested, the more instable the test client (PushToTest TestMaker 4.3.1) became. More specifically, the test client crashed all the time when using a word value of 30.000 for the SOAP RPC encoding style.

The average TPS values for these runs show the aforementioned statement (SOAP RPC performs better when the number of words increases) does not hold: SOAP Document Literal performs better than SOAP RPC (3,21 TPS versus 3,14 TPS).

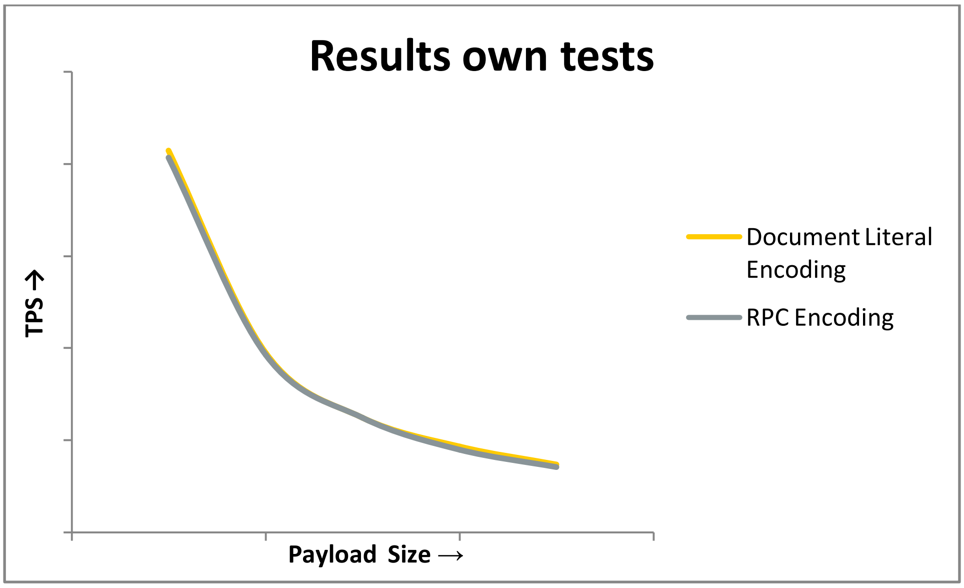


Chart 5-2: Our own results of the SOAP encodings performance tests. As can be seen, both styles show similar behaviour.

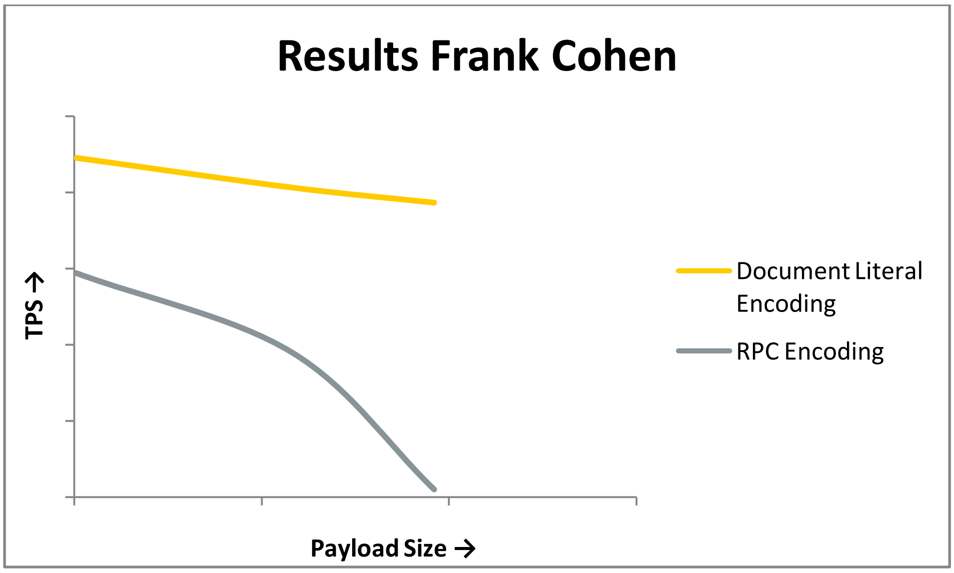


Chart 5-3: Frank Cohen's results of the SOAP encodings performance tets. As can be seen, There is a lot of difference in performance between the two encoding styles.

The main question that remains, is where the difference between Frank Cohen's research results and the research results on SOAP encodings in this thesis may come from (see Chart 5-2 and Chart 5-3). The document Frank Cohen provided for IBM DeveloperWorks (Cohen, Discover SOAP encoding's impact on Web service performance, 2003) explains that for his tests, he used Sun Solaris E4500 servers with 6 CPUs and 4 GB of RAM. Clearly, this hardware is very different that the hardware that was used for this research: two personal computer with one CPU each and 1 or 2 GB of RAM, being less powerful than the servers Frank Cohen used. This difference may also be an explanation for the TPS difference between the results. As can be seen in Chart 3-1, Frank Cohen's test results start with a TPS value of over 250, where our results start with a TPS value of only 20. Where the Sun Solaris servers are powerful enough to keep SOAP Document Literal Encoding at a high TPS value as the payload size increases and let the SOAP RPC Encoding start of with a high TPS value, our personal computers are not powerful enough to keep either encodings at a reasonable TPS value. This difference in hardware may be the reasons for the large difference. Another reason for the difference might be faulty implementation of either Frank Cohen's setup or the setup used for this research. However, since the same testing code and web service is used for both environments, this chance is rather small. When looking closer at these results and the results that were achieved by Frank Cohen, the recommendation that is given is to use SOAP Document Literal Encoding. Our results showed that there is almost no difference using a personal computer. Therefore, it is best to use the encoding style according to the WS-I Basic Profile 1.0 standard, namely SOAP Document Literal Encoding.

This analysis leads to the following conclusions, that is somewhat in contrast with Frank Cohen's research:

The encoding style of a SOAP message does not have a significant impact on the performance of services in a Service Oriented Architecture when run on a personal computer. However, when run on a professional server, the encoding style possibly can have a large impact. Further research is needed on this matter. Until then, choose the encoding style according to the production environment.

5.3 XML Parsers

As mentioned before in Chapter 3.4, there are three types of XML parsers: Streaming API for XML (StAX), Simple API for XML (SAX) and Document Object Model (DOM). Table 5-1 gives an overview of the differences between these types of parsers (Sun Microsystems, 2005).

	StAX	**SAX**	**DOM**
API Type	Pull, streaming	Push, streaming	In memory tree
Ease of use	High	Medium	High
CPU and Memory Efficiency	Good	Good	Varies
Read XML	Yes	Yes	Yes
Write XML	Yes	No	Yes

Table 5-1: Comparison of XML parser types

According to Frank Cohen, there is a lot of performance difference between the types of parsers, as mentioned in Chapter 3. In order to validate or reject this and, tests were conducted. Furthermore, the results can be used to give advice to Logica for future projects, since the results of the questionnaire showed, that in most of the project the different types of parsers are not taken into account. The following components were involved in these tests:

- XMLGen. A benchmark data generator that generates XML documents of various sizes. The size of the XML document to generate can be provided by the user of XMLGen (CWI, 2003).
- Four different XML documents that were generated by XMLGen, varying in size:
    - xmark_0.00001.xml, 27KB
    - xmark_0.0001.xml, 34KB
    - xmark_0.001.xml, 116KB
    - xmark_0.01.xml, 1.155KB
- A SAX implemented parser.
- A StAX implemented parser.
- A DOM implemented parser.
- A Java class that did the actual comparison of the parsers.

The first tests that were performed was the actual parsing of XML documents of varying size mentioned above. As can be seen in Figure 5-3, the performance tests that were run were done in two phases: first , the smallest document (27KB) was parsed by the StAX, SAX and DOM parsers in the warm up run. Next, all the documents generated by XMLGen were parsed by the three parsers. To keep in mind that the parse time may be influenced by other factors, such as computer use and other running programs, these documents are parsed five times. Computing the average values of these five runs results in a reliable measurement. Furthermore, these entire tests were performed three times. The results of these tests can be found in Appendix C.

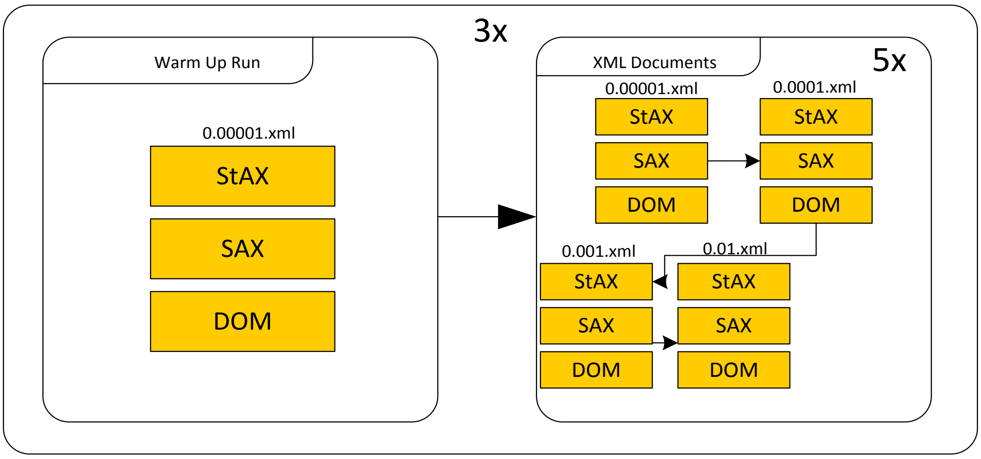


Figure 5-3: Testing the parsing of XML documents was done using a warm up run parsing the smallest document and 5 test runs for each type of parser parsing all four documents.

However, to make a fair comparison between the parsers, the next phase (performing actions on the parsed data) also has to be taken in account. For this reason, other tests were performed. By looking at the XML schema that is used by XMLGen to generate the five different XML documents, it was possible to get to know the structure of the XML documents. The XML documents represent on-line auctions, like eBay. Each XML document consists of one or multiple articles that are offered at the auction by users. For each article, the XML documents keeps track of various kinds of information, such as the highest bidder and the quantity of the offered article. Using this knowledge, it was decided to test the different parsers at the speed of counting the number of products that are offered once, since this requires the parser to look at all elements in the document. This operation is the same as counting the occurrence of the following text in each XML document.

```
<quantity>1</quantity>
```

The results and an analysis of these tests can be found below.

5.3.1 XML Parsers Test Result Analysis

The previous section explained how the tests to measure the difference between the StAX, SAX and DOM were structured. When using our intuition to predict the performance of each parser, one would think that the fastest parser would be SAX, then StAX and DOM would be the slowest kind of parser. As mentioned before, StAX is a pull parser: it does not generate events that call the user’s callback methods like push parsers do, but the returning of events is requested by the program. When you want to parse whole documents, it would not be logical that StAX would be faster than SAX. If this would be the case, SAX parser could be made using a while loop (e.g.

`while(getNewEventsFromStAX) do SAXcallback`). Looking at this reasoning, one would expect for StAX only to be useful when you want to skip unwanted events. Furthermore, DOM will most of the time be the slowest type of XML parser, since it generates an in-memory tree representation of the XML document. Frank Cohen's research on parsing does not state concrete results, but states each parser is best suitable for a specific situation. When dealing with moderate to huge documents, he states to use a XML Binding compiler, such as JAXB, to address elements directly. For medium sized documents, he proposes to use DOM parsers and for documents that contain elements that are not nested deeper than two or three elements, he proposes to use StAX parsers for skipping unwanted elements.

The actual results from our own tests that did not include XML operations (only the parsing itself), can be seen in Chart 5-4. Results from the second type of tests (including of XML operations on the documents) can be seen in Chart 5-5. As can be seen both charts, the expectations of the DOM parser are true: for each XML document, DOM was slowest in parsing. As the size of the XML document grows, the time DOM takes to parse it, grows even more. Clearly, it is not wise to choose DOM for parsing large documents. Furthermore, the expectations about SAX and StAX are also true: for each XML document, SAX was faster than StAX. The difference between SAX and StAX performance, grows as the size of the XML documents grows. However, these two parsers perform much better than DOM, so it would be wise to choose one of these parsers if you just have to parse XML documents. The choice between one of these two parsers can depend on various things, such as ease of implementation and expertise.

When looking at the results of the tests (see Chart 5-5) that included the counting of the `<quantity>1</quantity>` element, the results are different. As can be seen, the time the SAX parser took has increased in comparison with the first test, because of checking if each event that is thrown by the parser is a `<quantity>1</quantity>` element. Opposite to this, the time StAX took to parse the document has in most cases decreased. The reason for this can be found in the power of StAX. Using StAX, it is possible to skip elements, so every time an element not equal to `<quantity>1</quantity>` is encountered, it is not processed, saving parsing time. The DOM performance is increased only a bit. A reason for this might be that the largest part of DOM parsing time is creating the tree that represents the XML document. Searching the tree is only a small task for DOM parsers. Chart 5-5 shows two types of DOM parsers, DOM 1 and DOM 2. The reason for this, is that there are two ways to retrieve elements from a DOM tree, namely using XPath (XML Path Language (XPath) Version 1.0, 1999) (DOM 1) and using a specific function, `getElementByTagName` (DOM 2). As Chart 5-5 and the figures in Appendix C show, there is not much difference between the two ways. However, once familiar with XPath, the XPath way is much more powerful in use.

When looking at these results and the recommendations by Frank Cohen, similar conclusions can be made. DOM is always the slowest of the three types of parsers, so it would be wise not to use DOM for parsing large documents. However, DOM is the easiest type of parser to use and implement. When there is a need for skipping unwanted elements in a small to moderate XML document, StAX would eventually the fasted solution. This is not the case when searching for the `<quantity>1</quantity>`, but we can see that StAX will eventually be faster, when the complexity of the search query increases. Both Frank Cohen and some respondents of the questionnaire mention the use of XML Binding for parsing XML. Frank Cohen recommends the use of XML Binding for parsing moderate to huge XML documents. Respondents answered that they used XML Binding because it's often part of the toolkit they use. When combining Frank Cohen's recommendation and the respondents answers, the recommendation that is given in this thesis is to use XML Binding when using large XML documents and a toolkit (such as the Microsoft Visual Studio that is often used in SOA projects) because of the complexity when dealing with such large XML documents. The toolkit can handle all the XML parsing using XML Binding techniques.

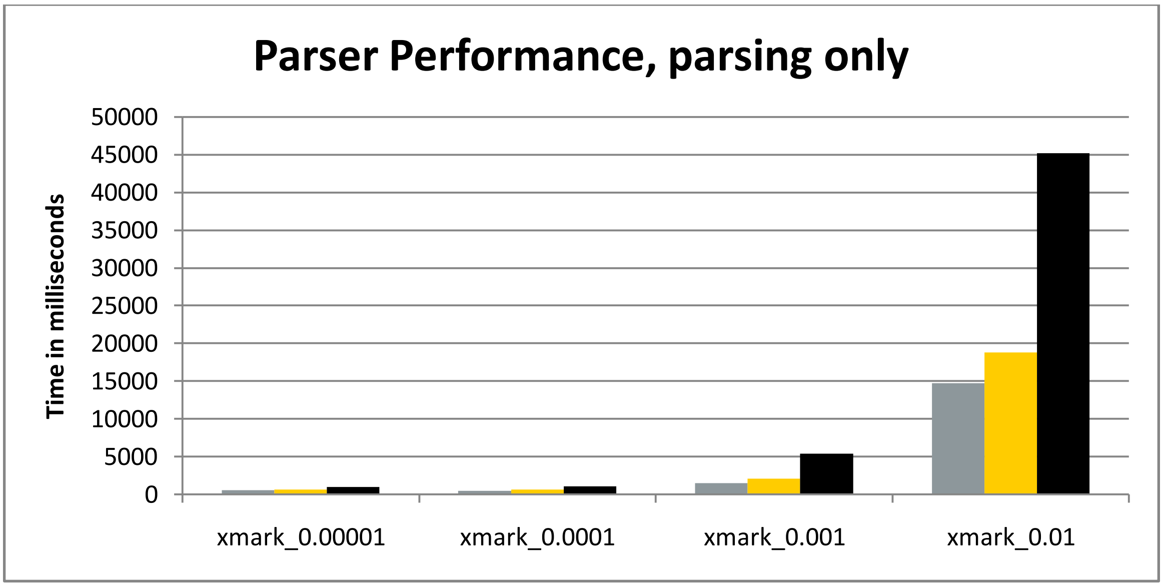


Chart 5-4: Results of the performance tests for the three XML parsers, parsing only. The difference between the three parser types is noteworthy.

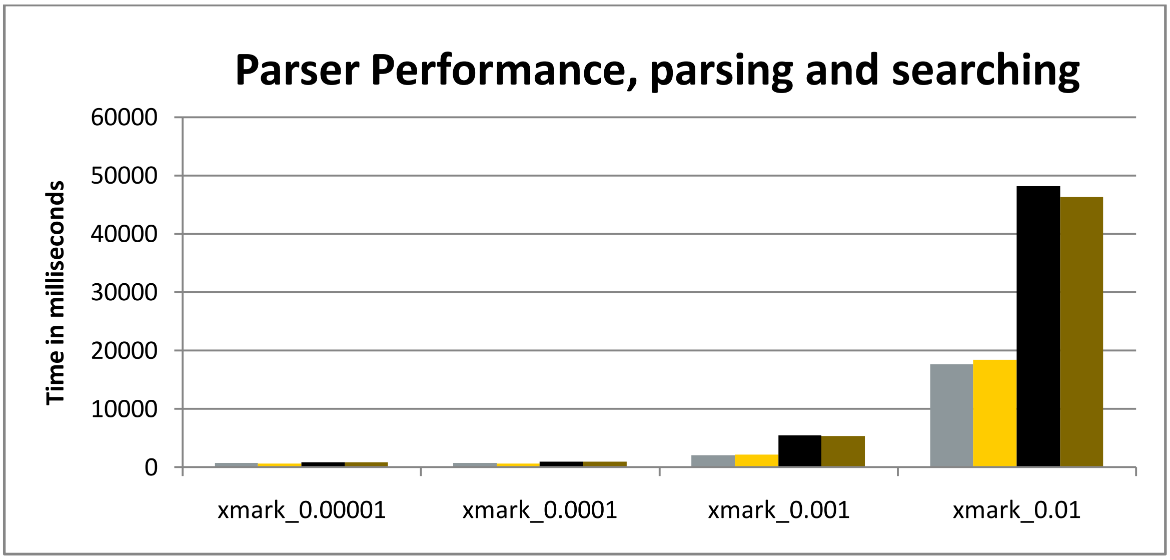


Chart 5-5: Results of the performance tests for the three XML parsers. As can be seen, the difference between the SAX and StAX parsers decreased.

SAX, StAX and DOM differ a lot in performance. When considering the differences in performance and implementation, using DOM would be the best for small documents. DOM is easiest to use and implement. However, as the size of the XML documents grow, it is recommended to look at other solutions, such as SAX or StAX. When dealing with large to huge XML documents, it is best to use XML Binding techniques.

6 Speeding up SOA

6.1 Introduction

The previous chapter showed the results of tests that were conducted for this research. The results of these tests validated, as well as rejected some results of the research by Frank Cohen. We did also see big performance differences between the different types of XML parsers. Our own tests did not lead to big differences when using the two different type of SOAP encodings like Frank Cohen's tests did. According to these two perspectives (existing research and our tests) and the other two perspectives (questionnaire and literature study), one can see that the extra increase in performance would be welcome, since the questionnaire respondents do sometime experience performance problems.

Besides careful considering the implementation specific choices, such as the XML parsers and encoding styles, there are some other options that can greatly increase SOA performance. This chapter will focus mainly on one of those options, namely caching.

6.2 Option 1: In-Memory Data Grids and Caching

As mentioned in Chapter 3.1, there are four fundamental questions to be asked when evaluating the performance of services in an SOA environment:

- Who or what will be accessing the services?
- When will the services be accessed?
- How often will the services be accessed?
- Where will the services be accessed from?

When looking closer at these four questions, one can conclude that all four questions are about availability, reliability and scalable performance. One technique that can be used to ensure high availability, reliability and scalable performance are data grids. Before explaining what is meant in this research with *in-memory data grid and caching*, it is important to know what is meant with the separate parts of this term (Purdy, 2008).

- A data grid is a system composed of multiple servers that work together to manage data and related operations in a distributed environment.
- An in-memory data grid is a data grid that stores data in the memory of the servers in the data grid.

- Caching is a particular technique to mitigate an application's load on a database without violating data-correctness if this data changes (Oracle, 2007).

6.2.1 Caching

A technique that is often used in combination with a database is caching. It is derived from the French word for hiding: "*cacher*". Caching is a technique to increase the performance of an information system by mitigating the load on a database. It can be seen as if caching hides the database in which the cache is placed in front of. Generally, it works as follows: if a program often tries to access the same set of data that is stored in a database, a lot of performance increase can be achieved by storing this data in a place that can handle the data requests much faster than the database. If the data is needed once more, the data can be retrieved from the cache, mitigating the load on the database.

The careful reader might notice that this description of caching is similar to the description of an in-memory data grid that can found above. However, there is one main difference that can be found by looking at the literal French meaning of *"cacher"*. Since caching is supposed to hide something (a database), actual caching always involves a database. In the case of in-memory data grids, there does not have to be a database: The data that is stored in the memory of the nodes in the grid, makes up the database.

There is one important aspect to caching: there has to be a certain degree of tolerance for out-of-date (stale) data. During the time that frequently accessed data is stored in the cache, it is possible that another process might have written other values to this data in the actual database. This invalidates the values of the data that is stored in the cache, leading to incorrect value retrievals: the data becomes out-of-date, or stale. A solution for this is to set a time to live value (TTL) for each cached value (Jung, Berger, & Balakrishnan, 2003). Once this TTL value has expired, the data will be discarded from the cache. Note that it still is possible that an application retrieves stale data from the cache, but this data has been stale for at most the value of the TTL.

6.2.2 Caching in combination with a data grid

By building a data grid that consists of the nodes where services are running and loading often requested data into the memory of each of these nodes (resulting in an SOA in-memory cache data grid), it is possible to avoid sending lots of data and thus increasing the response time/performance of the SOA environment (Gray & Prashant, 2000). This way, services that need data can get it from memory instead from getting it in a message or from a database (Oracle, 2008; Microsoft, 2003).

The question that remains is how such a solution ensures availability, reliability and scalable performance. In traditional systems that do not use a grid structure for storing and retrieving data, there can be so-called *single point of failures (SPOFs)*. An SPOF is a part of a system that when it fails, causes the complete system to fail. For information systems this can be for instance a database. If this database fails and there is no mechanism to switch to another redundant database, it becomes impossible to retrieve any data that is stored in the database. Especially in Service Oriented Architectures, it is important that all data services rely on are available all the time, since each service needs to have access to the data.

Caching in combination with a data grid as explained in this chapter increases the reliability and availability of the data in SOA environment. By copying the data to all of the nodes in the data grid and synchronizing these nodes with each other every time the data changes, each node cannot be seen anymore as a SPOF: the data is available at all nodes. When one node goes down, it can still be retrieved from the data grid (Allcock, Bester, & Bresnahan, 2001; Karczewski & Kuczynski, 2007).

The question that remains is how caching in combination with a data grid ensures scalable performance. There are two ways to scale systems as demand increases: scaling up and scaling out (Devlin, Gray, Laing, & Spix, 1999). Scaling up involves replacing servers with larger servers (buying bigger 'boxes') and scaling out involves adding extra servers (buying more 'boxes' of the same type). Caching in combination with a data grid falls in the category of scaling out. By using caching in combination with a data grid, it is possible to extend the data grid, as the demand increases, this way increasing scalability. The use of the caching (layer) reduces the load on the database, reducing the need for the database itself to scale along with the data grid(Oracle, 2008).

6.3 Option 2: The FastSOA Solution

As Frank Cohen performed his research, he also came up with a solution to speed up SOA, called the Fast Service Oriented Architecture, or FastSOA. As can be seen in Figure 6-1, FastSOA adds an XML Mid-Tier to the SOA environment. FastSOA provides a mid-tier service SOAP binding, XQuery processor (W3C, 2008) and native XML database.

By keeping the binding native to XML (so Java SOAP Bindings/proxies are not used anymore), no transformations have to be made on the incoming XML message to e.g. Java Request Objects. The binding calls a query to handle the XML request document in the XQuery processor. This XQuery processor parses the incoming document and checks the XML database to see whether the request was previously received. This XML database functions as a cache for commonly requested data. If a request was previously received, the FastSOA service is able to return the response from the XML database without having to go through the ESB to make the actual request to the service. By returning such cached values, service consumers often can get responses much faster. However, for such caching to work, it is important that the real data does not change, since this would invalidate the cache (see Chapter 6.2.1). Furthermore, the XQuery processor can be used to transform requests and responses between incompatible schema types. For example, the XQuery processor can be used for transformation when a service consumer uses Schema 1.0 to request a response and a service uses Schema 1.1 to provide responses.

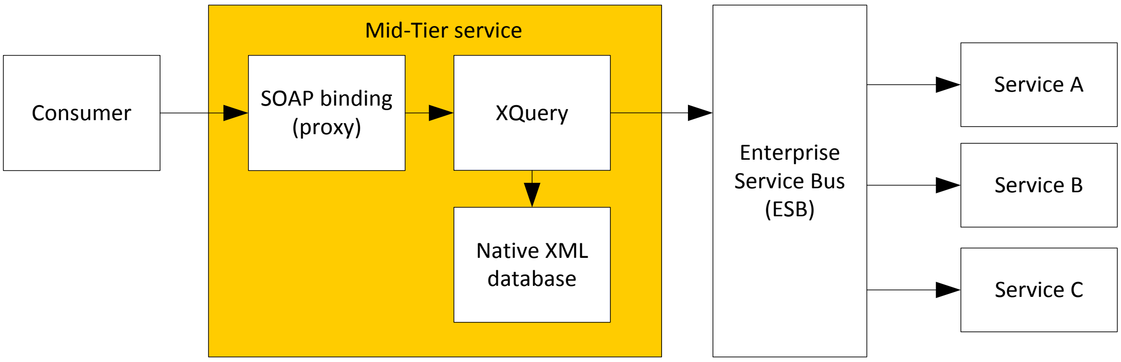


Figure 6-1: The solution as proposed by Frank Cohen for the issues mentioned. This solution involves a native XML database in the Mid-Tier for caching service responses.

6.4 Option 3: Mid-Tier Service Response Caching

Chapter 6.3 mentioned the use of a native XML database to store service responses for caching purposes as part of the FastSOA solution. This option suggests only the caching part of the FastSOA solution, since the results of the questionnaire showed that not using toolkits for creating SOAP Bindings/proxies is not a valid option. This caching is called Mid-Tier Service Response Caching.

Below are given two examples to clarify this type of caching. The first is without Mid-Tier Service Response Caching and the second is with Mid-Tier Service Response Caching.

- A data request message from a service consumer enters an SOA environment without mid-tier service response caching. This message is transported through all layers, all the way down to the appropriate service(s) that queries the resource management/data layer. After the requested data is retrieved from the resource management/data layer, it is returned to the service consumer by going once again through all the layers.
- A request message from a service consumer enters an SOA environment, with mid-tier service response caching. Before the message is routed by the ESB to the appropriate service(s) and querying the service to retrieve the requested data, the mid-tier service response caching comes into play. This technique queries a cache to see whether the request message was handled before. If this is the case, the requested data is retrieved from this cache and returned to the

service consumer. In order for this service response caching to work, it is necessary that the responses that are currently not in the cache are routed to the appropriate service as if the mid-tier service response caching is not present and that the data retrieved is then inserted into the cache (see Figure 6-2).

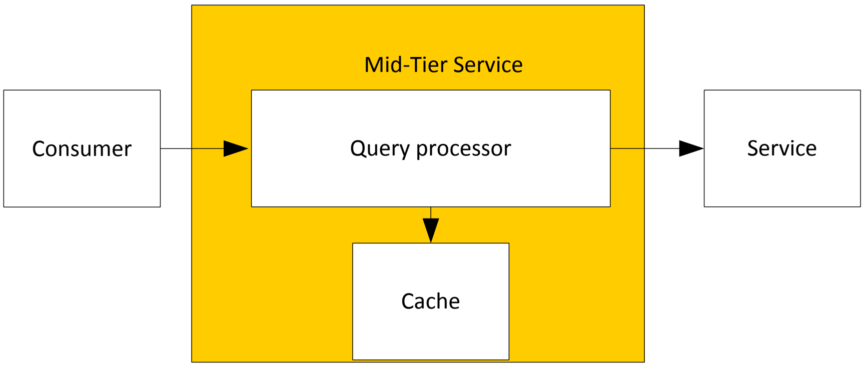


Figure 6-2: Mid-Tier Service Response Caching is used to increase the speed of querying a service by providing a cache to store service responses.

It is hard to test Mid-Tier Service Response Caching accurately in practice. In order to test it, there are two extremes:

- Testing it in a real, large SOA environment that uses a lot of service requests, service responses and data flows.
- Testing it using a single service that caches service responses.

The first of these two ways to test Mid-Tier Service Response Caching is the approach that requires a lot of work and knowledge in infrastructure, and vendor specific middle- and software that is not present for this research. An example is extensive knowledge of application servers and coding, since caching policies have to be set up and maintained in the SOA environment that often consists of application servers. For this reason, such an approach is beyond the scope of this research.

The second approach is also not a valid option for other reasons. The one and only service that is required for testing has to have enough complexity in order to be representative for an SOA environment. One can think of service that needs two parameters as input in order to query a search engine such as Google (Google, 2008) and structures the results in a clear way before showing it to the service consumer. However, building complexity in the service by using third party software introduces dependency on this third party software: once the third party software is not working or heavily used at a certain time, the service itself is either not working or very slow. Once the performance of this software decreases, the performance of the service also decreases. Another way is building a complete service yourself. A downside to this approach is the structuring, complexity and architecture of the service, because it has to be representative for an SOA environment. Other, less extreme approaches introduce a combination of these problems.

Another thing that makes testing Mid-Tier Service Response Caching nearly impossible, is the dataset that is used to query the service to be tested. Consider a service that has only one input parameter, that performs Mid-Tier Service Response Caching and that can hold at most one service response in its cache. Now assume that the service will be tested by using a large dataset that makes up the input parameter and that is made with a random data generator. Say that it comprises of 1000 input parameters of 10 different values. Because a random data generator is used to make the dataset, the dataset will have a discrete uniform distribution on the 10 possible values, so each different value will approximately appear 100 times in the dataset. When a specific parameter is given to a service that can hold one for the first time, the service response is cached into a database (see Figure 6-2). However, since the dataset is uniformly distributed, changes are that once in 10 times, a cache hit will occur. Clearly, this makes caching very trivial, so another approach is required.

For the reasons mentioned above, we will focus on a theoretical analysis of Mid-Tier Response Caching with a small proof of concept.

6.4.1 Mid-Tier Service Response Caching Advantages

Obviously, caching can improve the performance of the service: responses can be faster delivered by the service. For this statement to be true, the average time of storing a value to cache in a caching database plus the average time of querying the caching database and retrieving a service response from the caching database has to be smaller than the average value for retrieving a service response the way it is done without Mid-Tier Service Response Caching. Furthermore, using Mid-Tier Service Response Caching , less resources are needed to be able to process the service requests, because less load is placed on each layer when a service response is cached. However, extra resources and time are needed to give structure to the caching environment.

6.4.2 Mid-Tier Service Response Caching Disadvantages

One large disadvantage of Mid-Tier Service Response Caching is the occurrence of *'false positives'*. Suppose that a service response is cached into the cache database. When a user performs a service request that is the same as the service request of which the service response is cached into the cache database, the service response is acquired from the cache database. The term for such a retrieval from the cache database is called a *'hit'* or *'positive'*. However, as the service response is kept in the cache database, the data that is used to construct the service response may have changed. This can mean that the service response that has been stored in the cache database has become incorrect (‘*stale*’). Now when a hit occurs, the service response that is retrieved from the cache database is incorrect too. This is called a ‘*false positive*’. To decrease the change of data becoming stale, it is possible to set a Time To Live (TTL) value/countdown timer for each service response in cache. When this timer has reached zero and the service response is still in cache, it will be removed from cache.

Furthermore, when using (Mid-Tier Service Response) Caching and deciding what to cache, you have to consider the three following things:

- Static content that is often requested should be cached. As mentioned above, it is a good thing to cache frequently requested data. However, a downside of this is that the cached data can at a certain moment become stale while the TTL value of the object has not yet been reached, resulting in the retrieval of stale data. Clearly, this is not the case with static content: static content will not change over time, thus the risk of retrieving incorrect data from cache is not present. It can even be possible to choose for not giving this static content a TTL value.
- Mind the size of cacheable objects since each cache has a maximum size. As a result, when you decide to cache large object, the cache can contain less objects. Even worse, suppose that the cache is full with small objects that are often requested. Now, when a large object has to be cached, the small object will have to be removed from the cache (a process called garbage collection). These operations cause additional CPU overhead for identifying and removing the ‘garbage objects’ (Zorn, 1991). Unless a large object is very popular (often requested relatively to other object that are often requested), this overhead outweighs the benefit of caching the object. This makes it important to e.g. set a maximum cache object size.
- When a decision is made to cache an object, the system may have to determine which of the objects currently in the cache must be purged to create space for the incoming one. This is called the cache replacement policy (Otoo & Shoshani, 2003) and there are many different types of this. Some examples are: *random (RND,* remove a random object from cache*), least frequently used (LFU,* remove the object from cache that is the least frequently used*), least recently used* (*LRU,* remove the object from cache that has not been requested for the longest time), *maximum inter-arrival time based on last k-backward references (LRU-K*, look at the time the last k times an object was requested and remove the object were this time value is the largest).

6.4.3 Proof of Concept

Being nearly impossible to test Mid-Tier Service Response Caching accurately, the need for proving that it is increasing performance still exists. For this reason, one small, stand-alone service that uses Mid-Tier Service Response Caching was created. This service was created in ASP.NET by using Visual C# .NET. As mentioned in the beginning of this chapter, there are two ways to 'test' Mid-Tier Service Response caching:

- Testing it in a real, large SOA environment that uses a lot of service requests, service responses and data flows.
- Testing it using a single service that caches service responses.

For the service that is made during this research as a proof of concept, the second approach is chosen. A single service is made, that asks the user for a search query . This query is then passed along to Google using a HTTP Request. Google in its turn searches the Internet for query results, and passes the results back to the web service. This service then passes the results to the service consumer. The combination of ASP.NET and Visual C# .NET gives the service developer the possibility to build services that support simple, easy-to-use Mid-Tier Service Response Caching (Microsoft, 2007). This caching in ASP.NET holds the service responses in memory for a certain amount of time (that can be set by the service developer), so when a service request is posted to the web service, it first checks if the service response is in memory. Two versions of the web service were developed, one with service response caching (see Figure 6-3), and one without service response caching (see Figure 6-4), The service with caching had a TTL value of 60 seconds. These services are called over the internal network at Logica, sending the query to Google over a proxy server.

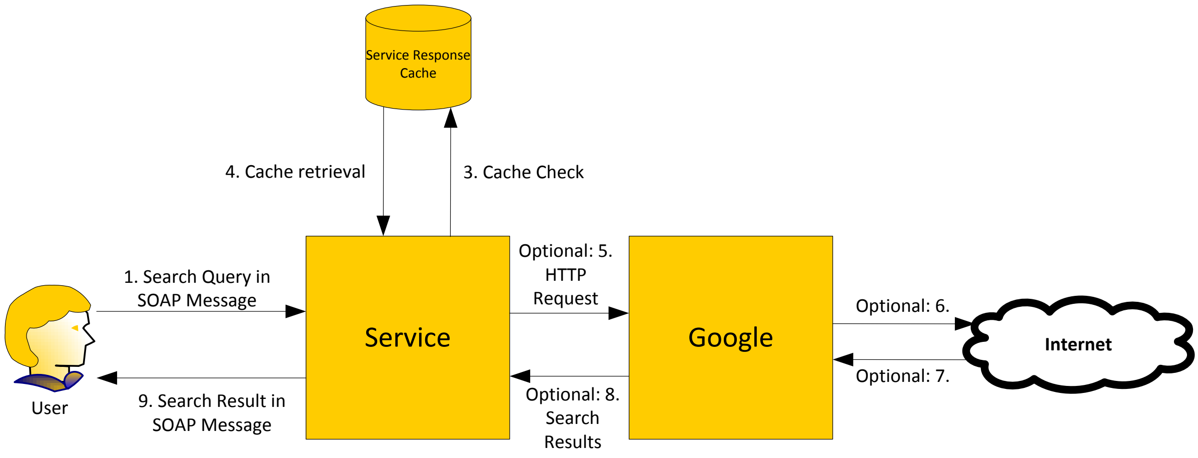

Service Response Cache
4. Cache retrieval
3. Cache Check
1. Search Query in SOAP Message
Service
Optional: 5. HTTP Request
Google
Optional: 6.
Internet
Optional: 7.
User
9. Search Result in SOAP Message
Optional: 8. Search Results

Figure 6-3: By including a service response cache in our proof of concept, a service request is processed more efficient than without a service response cache. It is possible that the service response is held in cache, reducing the response time.

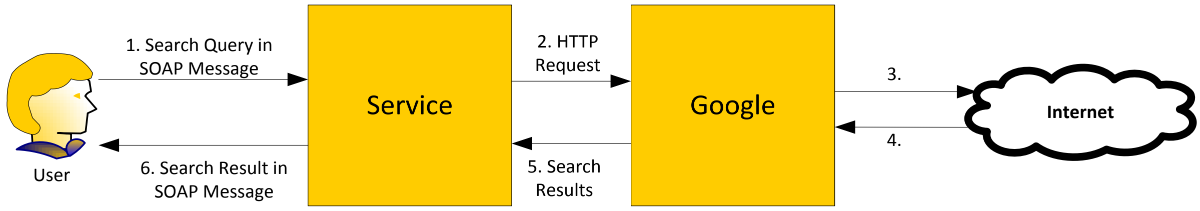

1. Search Query in SOAP Message
Service
2. HTTP Request
Google
3.
Internet
4.
5. Search Results
6. Search Result in SOAP Message
User

Figure 6-4: Without a service response cache in our proof of concept, the request is processed less efficient than with service response caching.

Chart 6-1 shows the differences in the response time between this service with service response caching and this service without service response caching. This figure shows that there is a big difference in response time between the server that uses service response caching and the one that doesn't. For each first search query the response time between the two servers is approximately the same. However, where the service without service response caching shows the same response time when a certain service request is send for the first time, the response time of the service with service response caching has decreased to (almost) zero milliseconds. When a service request is send to the service that provides service response caching that also has been sent before (longer ago than the TTL value), the service shows a similar behaviour as when the service request is send for the first time.

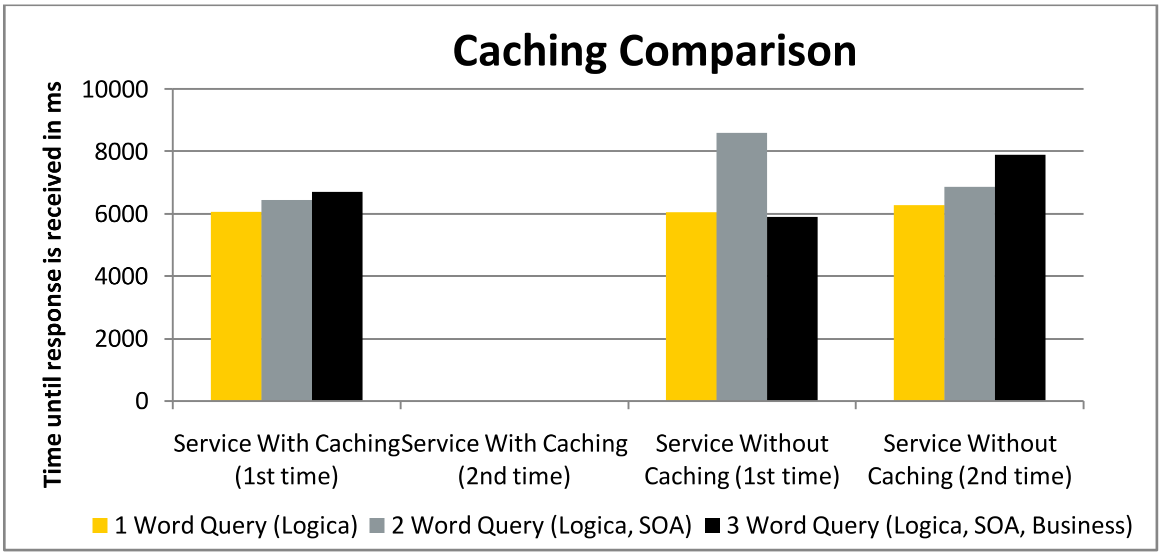


Chart 6-1: A comparison between a service with Service Response caching and one without. The first performs much better as repeated requests can be answered without any delay.

6.4.4 Mid-Tier Service Response Caching Example

To look whether Mid-Tier Service Response Caching is applicable in real projects within Logica, Figure 6-5 gives an example of a possible project, including Mid-Tier Service Response Caching. It shows a system that interacts with and gets data from internal services, such as a dossier and a login service and external services, such as employee administration and customer administration. In order to prevent a lot of service requests between the application and internal or external services, a Mid-Tier Service Response Cache can be placed between the application and the internal and external services. In real projects, it is important to deliberate on what services that use caching and what services do not. For example, in the project that is shown in Figure 6-5, it would be wise to cache the service responses that are generated from the dossier, and not for the login service. Login services have to be secure, so caching service responses are not be a wise choice.

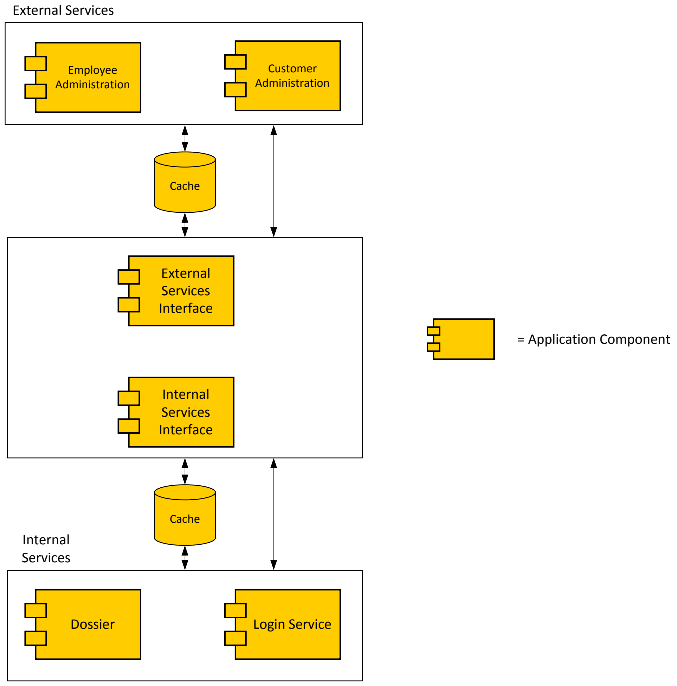


Figure 6-5: An example where Mid-Tier Service Response Caching is applicable within real projects.

6.5 Model of Middleware Solutions

When modelling the aforementioned solutions (Mid-Tier Service Response Caching and caching in combination with a data grid), it will yield a schematic overview of the request message flow as can be found in Figure 6-6. As can be seen, the in-memory data grid with caching is at the back end of the request message flow. If the request message invokes multiple services (A, B and C), each service can put data at the in-memory data grid with caching and this data will be available for each service.

In front of the ESB is the Mid-Tier service that offers the Mid-Tier Service Response caching. The request message flow clearly shows that, by using techniques such as the ones mentioned in this chapter, request messages do not have to go through all the layers all the way down to the resource management/data layer.

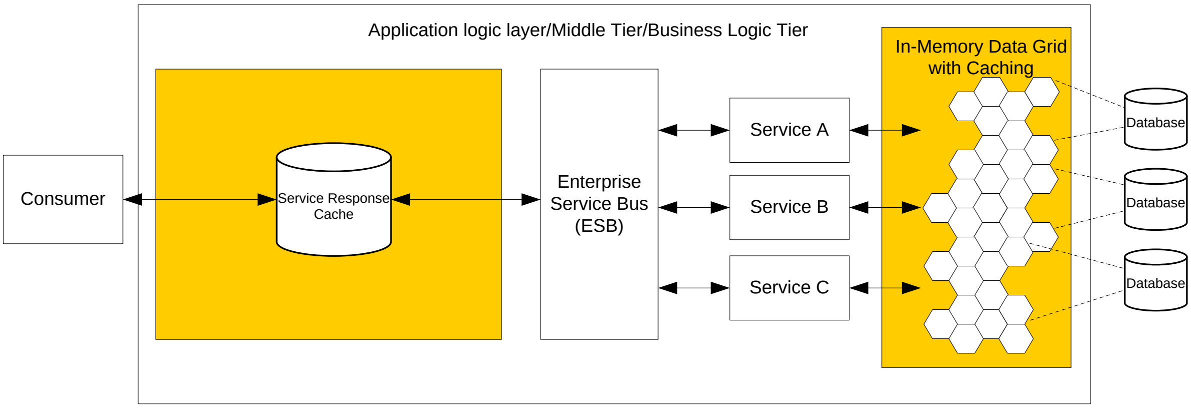

Application logic layer/Middle Tier/Business Logic Tier
Consumer
Service Response Cache
Enterprise Service Bus (ESB)
Service A
Service B
Service C
In-Memory Data Grid with Caching
Database
Database
Database

Figure 6-6: A mid-tier service response cache in combination with an in-memory data grid with caching results in this model.

6.6 Summary

There are various options for improving SOA performance, such as the FastSOA architecture, Mid-Tier Service Response Caching and middleware solutions. Mid-Tier Service Response Caching introduces a cache that can be used to store service requests and service responds. Furthermore, caching can also be applied between the databases and the services, in order to reduce the communication between the second and third layer. It is recommended that such options to increase SOA performance is more often used. All these solutions are general purpose solutions that can improve the performance. However, implementing such solutions can be time and investment consuming. Therefore, it is important to pay attention to important choices, such as the choices mentioned in Chapter 3 (SOAP encodings, parsers, out-of-the-box tools).

7 Conclusions and Advice

In this thesis we tried to get to know what SOA is and how it performs. This chapter starts with explaining what research actions have been taken and showing the results of research by answering the research questions. Next, it gives advice to Logica that can be used during SOA projects. It ends with suggestions for further research on the topic of SOA performance.

7.1 Carried Out Actions

The research goal for this thesis is defined in Chapter 1.3.1 as follows:

> *To determine whether a Service Oriented Architecture environment has to cope with load and performance issues by looking at it from multiple (4) perspectives. Research how the environment can cope with these possible issues. The results can be used to build SOA environments with high performance.*

In order to achieve this goal, the following actions were carried out:

1. Study theory about SOA. Since this research is about SOA, it is important to get to know what it exactly is.
2. Study theory about SOA performance. This theory is used to get to know whether people experience performance problems within SOA environments.
3. Perform a questionnaire with employees at Logica. This questionnaire was necessary to get to know how Logica handles SOA projects and whether Logica experiences problems during these projects.
4. Perform tests based on the results of the questionnaire. These tests are used to validate/reject another research on SOA performance.

7.2 Main Conclusion

Three research questions were defined in this thesis. These research questions are used to achieve the research goal. The research questions and answers can be found below.

What is Service Oriented Architecture?

Service Oriented Architecture is an architectural style to align IT with business processes in an enterprise. It reaches this goal by a set of application-independent services. Typically, each service that an enterprise offers is implemented by (smaller) IT services. An SOA environment can be seen in three levels of abstraction, namely business processes, services and operations. An SOA environment typically comprises mechanism for describing, discovering and messaging with services. The main advantages SOA offers are increase of enterprise flexibility, agility and consolidation and reduced costs.

Does an SOA environment suffer from load and performance issues?

According to an existing research by Frank Cohen, SOA environments have to cope with load and performance issues. He gives three possible causes for bad SOA performance. A literature study in this thesis does not validate Frank Cohen's reasoning , but it does explain that SOA does not always live up to the high expectation and promises. A questionnaire is taken with 50 employees within Logica that are involved in SOA projects or project with much XML messaging. Using this questionnaire, we found out how Logica implements SOA projects and what implementation choices Logica makes in these projects:

- The different SOAP encodings are most of the time not taken into account, if they are, most of the time SOAP RPC Document Literal Encoding is chosen. According to the best practises, this also is the best choice, because of interoperability.
- For almost every project, frameworks and tools are used to implement SOA environments.
- For most projects, the different XML parsers are not taken into account. If they are taken into account, the size of the messages is often the deciding factor for choosing the parser.
- Middleware to specifically increase the performance of SOA environments and services is almost never used.
- For more than half of the projects, there have been evaluation moments afterwards. This could be improved.
- For about the half of the projects, best-practices were discovered.

Furthermore, this questionnaire showed us that for about one third of the SOA projects and projects with much XML messaging performance specific problems are encountered. The main reason for these problems are the amount and size of sent and received XML messages. Results of this questionnaire and the possible causes for bad SOA performance as mentioned by Frank Cohen were used to set up tests. These tests were performed to see whether there is much difference between two types of SOAP encoding styles and three types of XML parsers. The results of the first tests showed us the following:

- There is not much performance difference between the two types of SOAP encoding, using a regular personal computer. However, an existing research, performing the same tests, showed us that there is a lot of performance difference between the two types of encoding, when using a powerful server. Therefore, further tests should be performed. As long as the tests using a powerful server are not performed, we will assume there is not much performance difference between the two types of SOAP encoding.
- There is much difference between the three difference types of XML parsers. Document Object Model (DOM) is easiest to implement, but also the slowest type of parser. Streaming API for XML (StAX) and Simple API for XML (SAX) deliver better performance but also are harder to implement.

How can XML technology be used in SOA environments, leading to larger load tolerance and reaching higher performance?

There are multiple ways to increase the load tolerance of SOA and reach a higher performance SOA. Since the research by Frank Cohen and the questionnaire with employees at Logica showed us that mainly XML can be a bottleneck for performance, we try to increase SOA performance using XML technology. Frank Cohen suggests a complete overhaul of SOA implementation, by discarding the use of toolkits and promoting the use of native XML technology. However, the questionnaire showed that this is not suitable for Logica. For this reason, this thesis showed that by using simple caching of service responses, a lot of performance increase can be gained. Furthermore, other middleware can be used, such as data grids, to improve performance and reliability.

7.3 Advice to Logica

SOA performance is observed from four perspectives in this thesis:

- A literature study, of which the results are shown in this thesis
- An existing research on SOA performance (Cohen, FastSOA, 2007)
- Questionnaire with employees at Logica
- Tests

After aggregating all the results from these perspectives and analyzing them, the following advices can be given to Logica for future SOA projects or other projects that make use of XML and SOAP messaging:

- *Use SOAP Document Literal Encoding when deciding upon the encoding of SOAP messages. Our questionnaire shows that no attention is given to the choice of SOAP encoding styles and the tests show that there is no clear difference in performance between the two styles when using a personal computer. However, another research, using a powerful server, showed big performance difference between the two encoding styles. Therefore, it is best to follow the W3C WS-I Basic Profile 1.0 standard(SOAP Document Literal Encoding) as long as no other tests, using a powerful server, have been performed.*
- *Our questionnaire shows that practical no attention is given to the different XML parsers. SAX, StAX and DOM differ a lot in performance according to our tests. When considering the differences in performance and implementation, using DOM would be the best for small documents. DOM is easiest to use and implement. However, as the size of the XML documents grow, it is recommended to look at other solutions, such as SAX or StAX. When dealing with large to huge XML documents, it is best to use XML Binding techniques.*
- *Use tool kits such as Visual Studio .NET for developing services. A large group of respondents experienced easier and less error-prone development when using such tool kits. However, mind the choices tool kits automatically make, such as XML parser and encoding style. Where possible, change the default choice according to this advice.*
- *Try to use middleware, e.g. data grids, when reliability and speed is a critical aspect.*
- *Evaluate more often on projects. The questionnaires showed that only a bit more than half of the SOA projects are evaluated. Evaluations can lead discovery of best practices.*
- *Make use of caching. Our proof of concept for caching in ASP.NET C# shows that it is to implement caching in Webservices. Pay attention to the different aspects of caching, such as stale data and time-to-live (TTL) values.*

8 Discussion

Looking back at the (practical) results of the research, and placing them into context of SOA in theory, one can see big differences between practice and theory. In the past two years of the Master Computer Science, SOA was depicted as a new paradigm for aligning IT with business, business being the driving factor for SOA. However, during this research, we found out that SOA is mainly an IT term. Most of the decisions that have to be made in an SOA environment, are based on IT. Even more, we have seen in this research that most decisions are not well considered. For these reasons, my expectations from SOA were quite different than my opinion of SOA after this research. At the moment, I'm curious about the effectiveness of the suggested solution for better SOA performance. I found it striking to see that almost none of the respondents were familiar with middleware solutions to speed up SOA, such as data grids. Personally, I think this really can improve SOA performance, when necessary. Another point that is worth mentioning in this discussion, are the difference between the results of Frank Cohen's research on SOAP encodings and the results of my research on SOAP encodings. As mentioned before, most likely to cause this difference is the set-up that is used for the tests. The set-up used by Frank Cohen is much more powerful than the set-up used by my research. It is possible that my set-up is not able to pull of high TPS values for both encoding styles (resulting in approximately 20 TPS) and that the set-up used by Frank Cohen is able to pull of high TPS values for both encoding styles (resulting in much higher TPS values).

9 Future Research

In Chapter 6.4.3 (our service response caching proof of concept) it is explained why a small service developed for this research is used for the proof of concept. This gave a clear view of the possibilities of service response caching. Still, it is recommendable to test service response caching in a larger environment, consisting of multiple servers. This will be more similar to a real SOA environment, and will give more accurate results and impressions. Moreover, it is suggested to tests service response caching in a real, working SOA environment. This way, the service response caching is tested using real data that are not randomly generated or made up.

As mentioned in Chapter 5.2.1 and in Chapter 8, there is a lot of difference between the SOAP encoding tests results we got and the results Frank Cohen got. In this thesis, we explained that this difference may be possible due to the difference in hardware. Frank Cohen performed his tests on Sun Solaris E4500 servers with 6 CPUs and 4 GB of RAM. We did the same tests on personal computers with one CPU and 1 or 2 GB of RAM, being less powerful than the servers Frank Cohen used. Our advice as given before, is based on our own measurements. As a further research, it would be wise to perform the tests again on similar machines like Frank Cohen used. Based on the new results, the advice could possibly have to be altered.

A last suggestion for future research is to look at other aspects of SOA in order to increase performance. By performing a research on e.g. the ESB, it is possible that other causes for low SOA performance or possible improvements for SOA performance can be found.

10 Bibliography

- Allcock, B., Bester, J., & Bresnahan, J. (2001). *Data Management and Transfer in High-Performance Computational Grid Environments.pdf.*
- Alonso, G., & Casati, F. (2004). *Web Services.* Springer.
- Arsanjani, A., Zhang, L.-J., & Ellis, M. (2007, March 28). *Design an SOA solution using a reference architecture*. Retrieved March 13, 2008, from IBM Developerworks: http://www.ibm.com/developerworks/library/ar-archtemp/
- Berg, M. v., Hompes, J., & Truijens, J. (2007). SOA in Nederland - een digitale aandoening? *Landelijk Architectuur Congres 2007* (pp. 17-52). Sdu Uitgevers.
- Cohen, F. (2003, March 1). *Discover SOAP encoding's impact on Web service performance.* Retrieved July 28, 2008, from IBM Developerworks: http://www-128.ibm.com/developerworks/webservices/library/ws-soapenc
- Cohen, F. (2007). *FastSOA.* Morgan Kauffman.
- CWI. (2003, June 26). *xmlgen - The Benchmark Data Generator*. Retrieved June 3, 2008, from XMark - An XML Benchmark Project: http://monetdb.cwi.nl/xml/
- Devlin, B., Gray, J., Laing, B., & Spix, G. (1999). *Scalability Terminology: Farms, Clones, Partitions, and Packs: RACS and RAPS.* Redmond: Microsoft Research.
- Dodani, M. H. (2004). From Objects to Services: A Journey in Search of Component Reuse Nirvana. *Journal of Object Technology* , 49-54.
- Evdemon, J. (2005, May 19). *The Four Tenets of Service Orientation*. Retrieved March 13, 2008, from SOAInstitute.org: http://www.soainstitute.org/articles/article/article/the-four-tenets-of-service-orientation.html
- Google. (2008). *Google*. Retrieved July 21, 2008, from Google: http://www.google.com
- Gray, J., & Prashant, S. (2000). Rules of Thumb in Data Engineering. *16th International Conference on Data Engineering*, (pp. 3-12). Redmond.
- Hao, H. (2003, September 30). *What Is Service-Oriented Architecture*. Retrieved March 3, 2008, from O'Reilly webservices.xml.com: http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html
- IBM. (2008, July 22). *developerWorks: IBM's resource for developers*. Retrieved July 25, 2008, from IBM: http://www.ibm.com/developerworks/
- Intel Software Network. (2007, February 15). *Determine the Correct XML Parser Type for a Java Application*. Retrieved April 7, 2008, from Intel Software Network: http://softwarecommunity.intel.com/articles/eng/3151.htm
- Josuttis, N. M. (2007). *SOA in Practice.* O'Reilly.

- Jung, J., Berger, A., & Balakrishnan, H. (2003). Modeling TTL-based Internet Caches. *Infocom: Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, (pp. 417-426).
- Karczewski, K., & Kuczynski, L. (2007). *Clusterix Data Management System (CDMS) - Architecture and Use Cases.*
- Logica. (2008, February 27). *Logica*. Retrieved June 6, 2008, from Logica: http://www.logica.com
- McNeill, P. (1990). *Research Methods (Society Now).*
- Microsoft. (2003, April). *Caching Architecture Guide for .NET Framework Applications*. Retrieved April 23, 2008, from Microsoft patterns & practices Development Center: http://msdn2.microsoft.com/en-us/library/ms978499.aspx
- Microsoft. (2007, March 27). *How to cache in ASP.NET by using Visual C# .NET.* Retrieved July 23, 2008, from Microsoft Hulp en ondersteuning: http://support.microsoft.com/kb/318299
- OASIS. (2007, April 12). Members Approve Web Services Business Process Executiuon Language (WS-BPEL) as OASIS Standard. Boston.
- OASIS. (2006, August 2). Reference Model for Service Oriented Architecture 1.0.
- Oracle. (2007, May). Achieving the Impossible: Unlimited Application Scalability.
- Oracle. (2008, February). Scalable, Grid-Enabled Service-Oriented Architecture Using Oracle Coherence.
- Ort, E., & Mehta, B. (2003, March). *Java Architecture for XML Binding (JAXB).* Retrieved June 27, 2008, from Sun Developer Network (SDN): http://java.sun.com/developer/technicalArticles/WebServices/jaxb/
- Otoo, E., & Shoshani, A. (2003). Accurate Modeling of Cache Replacement Policies in a Data Grid. *20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies*, (pp. 10-19).
- Papazoglou, M. P. (2003, December). Service Oriented Computing: Concepts, Characteristics and Directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering* , pp. 3-12.
- Papazoglou, Mike P.; Heuvel, Jan-Willem van den. (2007). Service Oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal 16* , 389-415.
- Purdy, C. (2008, January 10). *Defining a Data Grid*. Retrieved February 2, 2008, from /dev/null: http://www.jroller.com/cpurdy/
- *PushToTest*. (2008, March 1). Retrieved May 1, 2008, from PushToTest - The Open-Source Test Automation Company: www.pushtotest.com
- SAP. (2006). Flexibility with Enterprise SOA.

- SAX. (2001). *Events vs. Trees*. Retrieved May 13, 2008, from SAX: http://www.saxproject.org/event.html
- Sprott, D., & Wilkes, L. (2004, January). Understanding Service-Oriented Architecture. *Microsoft Architecture Journal* .
- Sun Microsystems. (2008). *Enterprise JavaBeans Technology*. Retrieved April 3, 2008, from Sun Developer Network: http://java.sun.com/products/ejb/
- Sun Microsystems. (2005, June 14). *Why StAX?* Retrieved May 21, 2008, from The Java Web Services Tutorial: https://java.sun.com/webservices/docs/1.6/tutorial/doc/SJSXP2.html
- The National Science Foundation. (2002). *The 2002 User Friendly Handbook for Project Evaluation.*
- The Open Group. (2008, January 31). The SOA Working Group.
- van der Lans, R. (2006, October 13). *Is soa eigenlijk eai met een dun laagje soap?* Retrieved February 13, 2008, from Computable: http://www.computable.nl/artikel.jsp?rubriek=1272857&id=1823705
- Verschuren, P., & Doorewaard, H. (2000). *Het ontwerpen van een onderzoek.*
- W3C. (2008, April 3). *W3C XML Query (XQuery)*. Retrieved April 24, 2008, from World Wide Web Consortium: http://www.w3.org/XML/Query/
- World Wide Web Consortium. (2007, June 26). *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Retrieved March 13, 2008, from World Wide Web Consortium: http://www.w3.org/TR/wsdl20/
- *XML Path Language (XPath) Version 1.0*. (1999, November 16). Retrieved July 10, 2008, from W3C Recommendation: http://www.w3.org/TR/xpath
- Zachman International. (2008). *Zachman International*. Retrieved March 13, 2008, from Zachman International: http://www.zachmaninternational.com
- Zimmerman, O., Krogdahl, P., & Gee, C. (2004, June 2). *Elements of Service-Oriented Analysis and Design*. Retrieved March 13, 2008, from IBM Developerworks: http://www-128.ibm.com/developerworks/library/ws-soad1/?ca=dnt-522
- Zorn, B. (1991, May). The Effect of Garbage Collection on Cache Performance.

Appendix A. Questionnaire Structure

1. Introduction of the researcher
2. Subject and goal of the questionnaire
3. State that filling in the questionnaire will take approximately 15 minutes.
4. Questionnaire
    a. Are you or have you been involved in projects in which Web Services and/or XML SOAP messaging played a large role?
    b. During this project, were SOAP encoding styles taken into account?
        i. For what reasons?
    c. During the project, did you make use of any out-of-the-box tools to ease the development of the services? For example to build SOAP bindings (proxies)?
        i. For what reason did you/didn't you?
    d. What considerations were made for choosing an XML parser?
        i. Were different parsers tested and what were the results of these tests?
    e. Did you make use of any middleware to improve performance during the project?
        i. If so, what were the considerations to use such middleware and what were the experiences?
    f. Did any problems arise during the project in terms of implementation technical issues (performance and scalability)?
        i. What caused these problems?
    g. Have there been any evaluation moments after the project?
    h. Did any best-practices come forward out of this project?
5. Final words

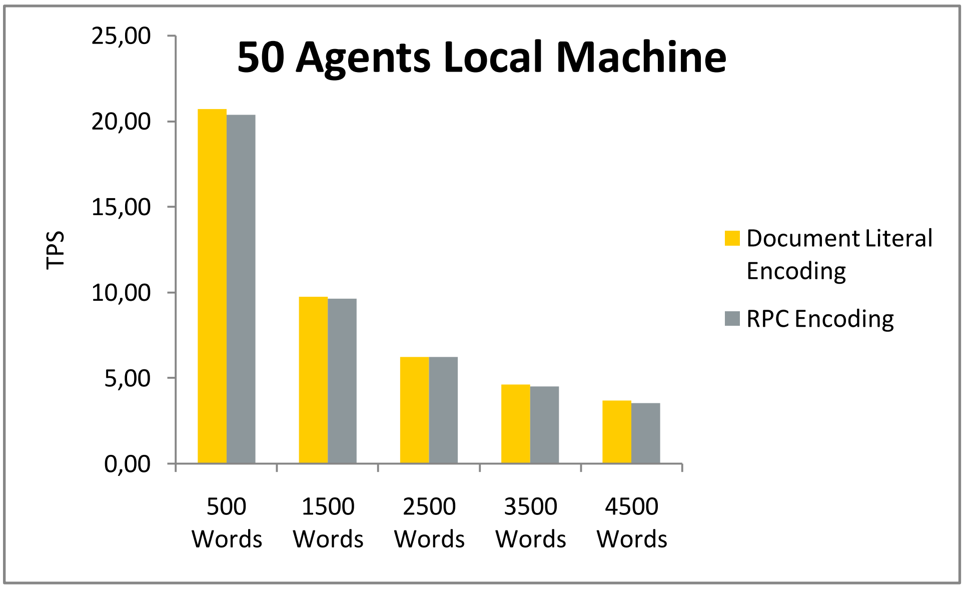
Appendix B. SOAP encodings Performance Test Results

This appendix shows the of the SOAP encodings performance tests that were explained in Chapter 5.2. There is a table and chart for each test performed. As can be seen, some numbers in the tables are coloured green and others are coloured red. The colour green means that the encoding was faster than the other encoding, the colour red vice versa.

50 Agents Local Machine

		Document Literal Encoding			
TPS	**500 Words**	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**
Testrun 1	20,43	9,32	5,89	4,64	3,62
Testrun 2	20,89	9,78	6,63	4,65	3,65
Testrun 3	20,98	9,88	6,25	4,63	3,58
Testrun 4	20,38	9,82	5,90	4,66	3,80
Testrun 5	20,88	9,93	6,52	4,68	3,78
Average	20,71	9,75	6,24	4,65	3,69

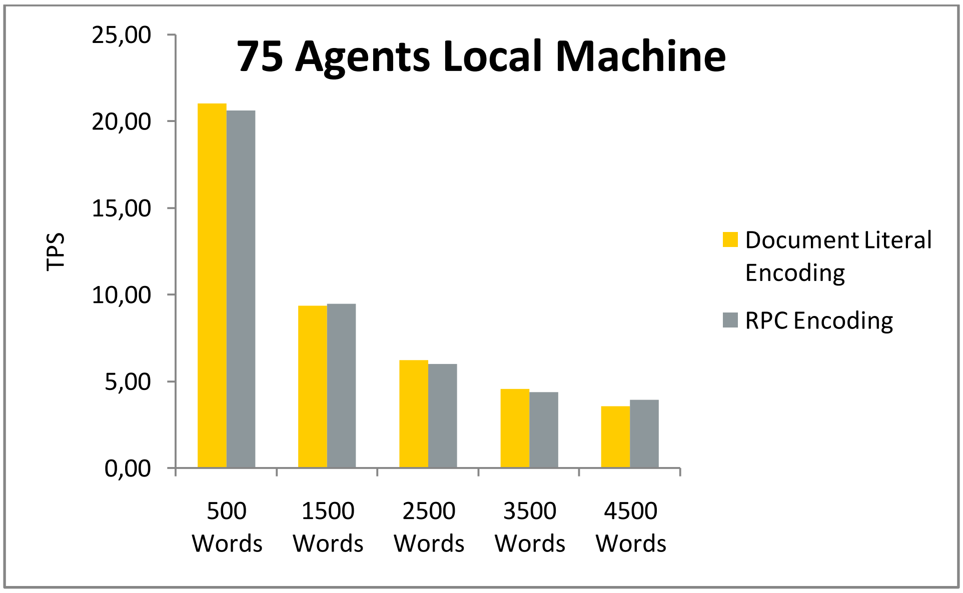
		RPC Encoding			
TPS	**500 Words**	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**
Testrun 1	20,35	9,93	6,10	4,62	3,47
Testrun 2	20,20	9,55	6,05	4,42	3,47
Testrun 3	20,85	9,75	5,95	4,57	3,53
Testrun 4	19,80	9,17	6,62	4,61	3,69
Testrun 5	20,55	9,85	6,41	4,30	3,55
Average	20,35	9,65	6,23	4,50	3,54



75 Agents Local Machine

		Document Literal Encoding			
TPS	500 Words	1500 Words	2500 Words	3500 Words	4500 Words
Testrun 1	20,83	10,02	6,24	4,85	3,58
Testrun 2	21,18	8,47	6,05	4,33	3,53
Testrun 3	21,10	9,12	6,38	4,51	3,33
Testrun 4	21,07	9,62	6,27	4,53	3,62
Testrun 5	20,83	9,54	6,07	4,43	3,64
Average	21,00	9,35	6,20	4,53	3,54

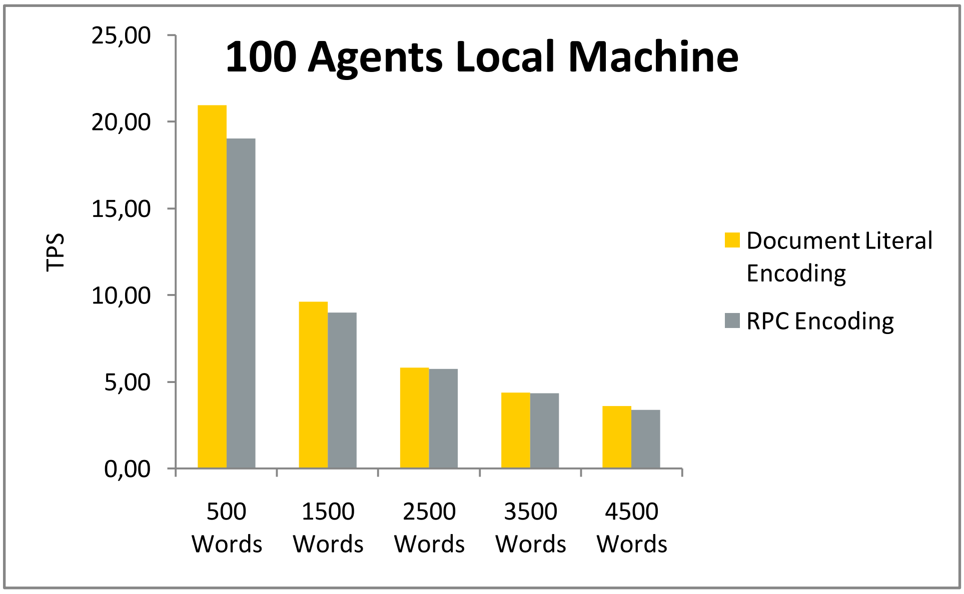
		RPC Encoding			
TPS	500 Words	1500 Words	2500 Words	3500 Words	4500 Words
Testrun 1	19,78	9,34	5,64	4,48	5,64
Testrun 2	20,63	9,30	6,18	4,27	3,37
Testrun 3	21,40	9,90	6,35	4,70	3,25
Testrun 4	20,15	9,45	6,08	3,88	3,54
Testrun 5	21,08	9,30	5,75	4,45	3,72
Average	20,61	9,46	6,00	4,36	3,90



100 Agents Local Machine

		Document Literal Encoding			
	500 Words	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**
Testrun 1	20,73	9,55	5,82	4,41	3,68
Testrun 2	21,19	9,98	6,15	4,50	3,40
Testrun 3	20,15	9,27	6,00	4,35	3,72
Testrun 4	21,25	9,60	6,08	4,40	3,80
Testrun 5	21,45	9,53	4,92	4,10	3,25
Average	20,95	9,59	5,79	4,35	3,57

		RPC Encoding			
TPS	**500 Words**	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**
Testrun 1	19,28	9,58	5,53	4,34	3,53
Testrun 2	18,73	8,69	5,71	4,19	3,25
Testrun 3	20,47	9,22	5,90	4,46	3,63
Testrun 4	16,07	8,03	5,45	4,20	3,05
Testrun 5	20,45	9,37	6,05	4,45	3,28
Average	19,00	8,98	5,73	4,33	3,35

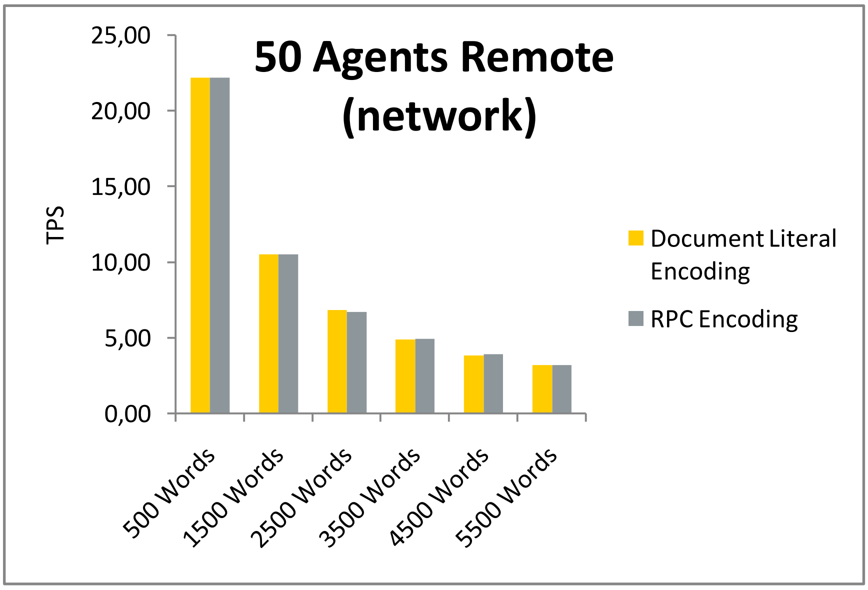


50 Agents Remote

(network)

		Document Literal Encoding				
TPS	**500 Words**	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**	**5500 Words**
Testrun 1	21,92	10,53	6,86	4,93	3,76	3,20
Testrun 2	21,90	10,75	6,83	4,76	3,80	3,22
Testrun 3	22,12	10,27	6,80	5,02	3,81	3,10
Testrun 4	22,53	10,39	6,78	4,86	3,77	3,27
Testrun 5	22,46	10,71	6,85	5,02	4,00	3,19
Average	22,19	10,53	6,82	4,92	3,83	3,20

		RPC Encoding				
TPS	**500 Words**	**1500 Words**	**2500 Words**	**3500 Words**	**4500 Words**	**5500 Words**
Testrun 1	21,67	10,61	6,64	5,00	3,91	3,32
Testrun 2	22,29	10,12	6,72	4,91	3,92	2,98
Testrun 3	22,31	10,67	6,76	4,83	4,00	3,20
Testrun 4	22,30	10,69	6,70	5,00	3,80	3,29
Testrun 5	22,32	10,48	6,81	4,92	4,05	3,20
Average	22,18	10,51	6,73	4,93	3,94	3,20

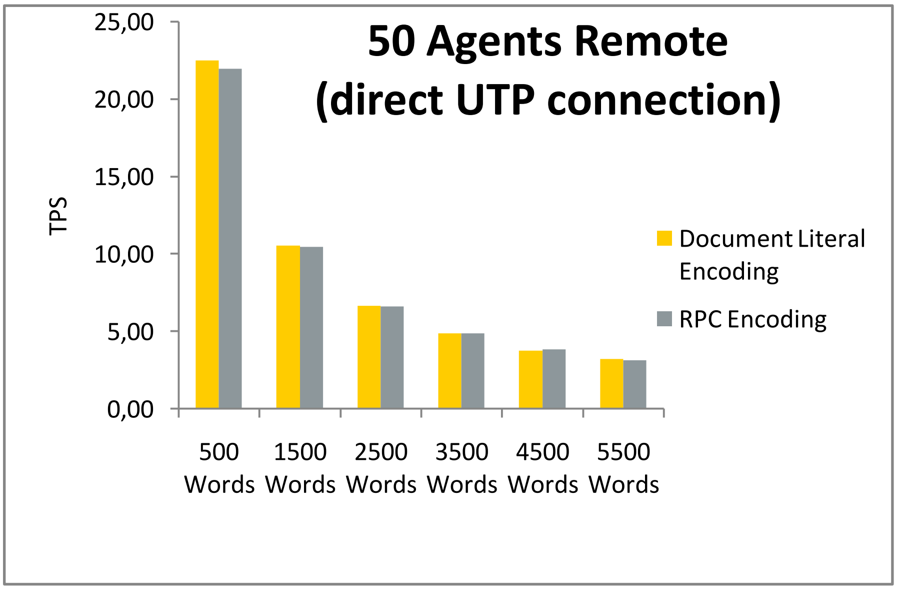


50 Agents

(direct UTP Connection)

		Document Literal Encoding				
TPS	500 Words	1500 Words	2500 Words	3500 Words	4500 Words	5500 Words
Testrun 1	22,40	10,51	6,71	4,97	3,83	3,10
Testrun 2	22,23	10,58	6,72	4,80	3,85	3,25
Testrun 3	22,61	10,66	6,64	4,93	3,64	3,19
Testrun 4	22,64	10,48	6,54	4,93	3,78	3,31
Testrun 5	22,54	10,52	6,57	4,72	3,63	3,19
Average	22,48	10,55	6,64	4,87	3,75	3,21

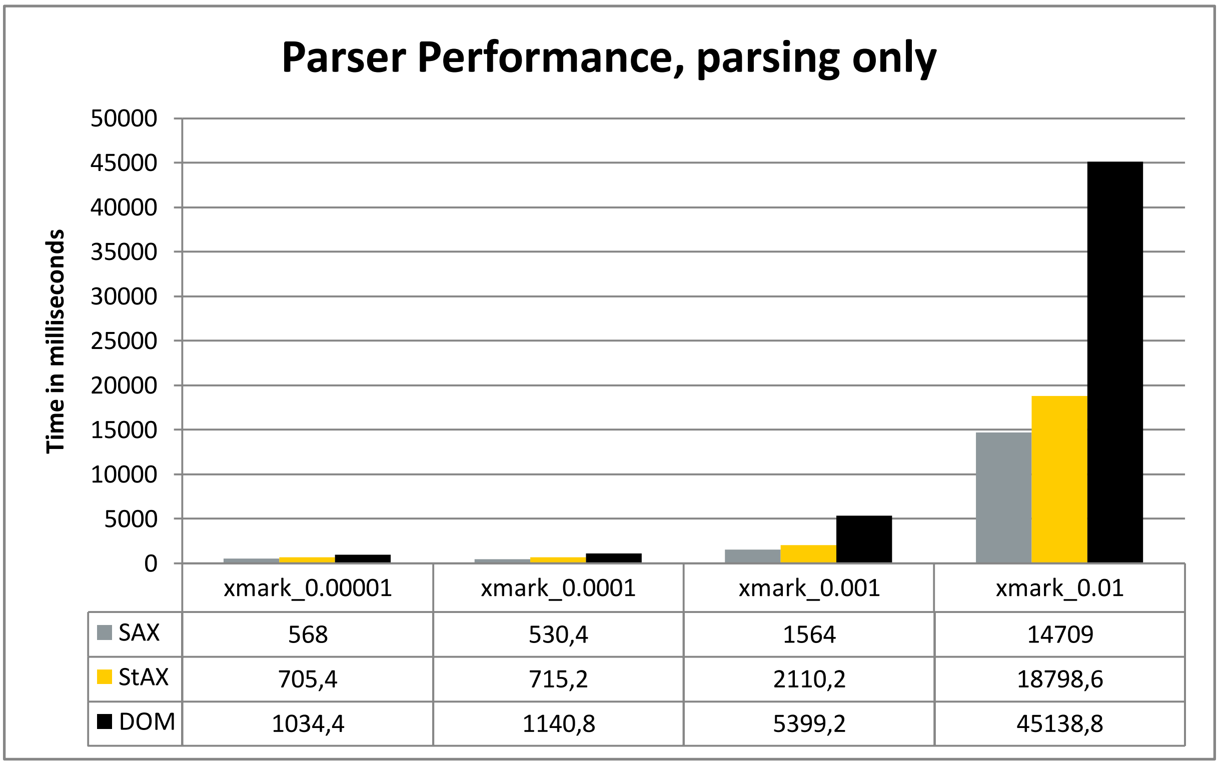
		RPC Encoding				
TPS	500 Words	1500 Words	2500 Words	3500 Words	4500 Words	5500 Words
Testrun 1	21,45	10,33	6,63	4,77	3,97	3,14
Testrun 2	22,42	10,58	6,67	4,98	3,79	3,07
Testrun 3	21,48	10,31	6,75	4,90	3,72	3,15
Testrun 4	22,25	10,51	6,33	4,81	3,88	3,15
Testrun 5	22,10	10,45	6,64	4,92	3,81	3,19
Average	21,94	10,44	6,60	4,88	3,83	3,14



Appendix C. XML Parser Performance Test Results

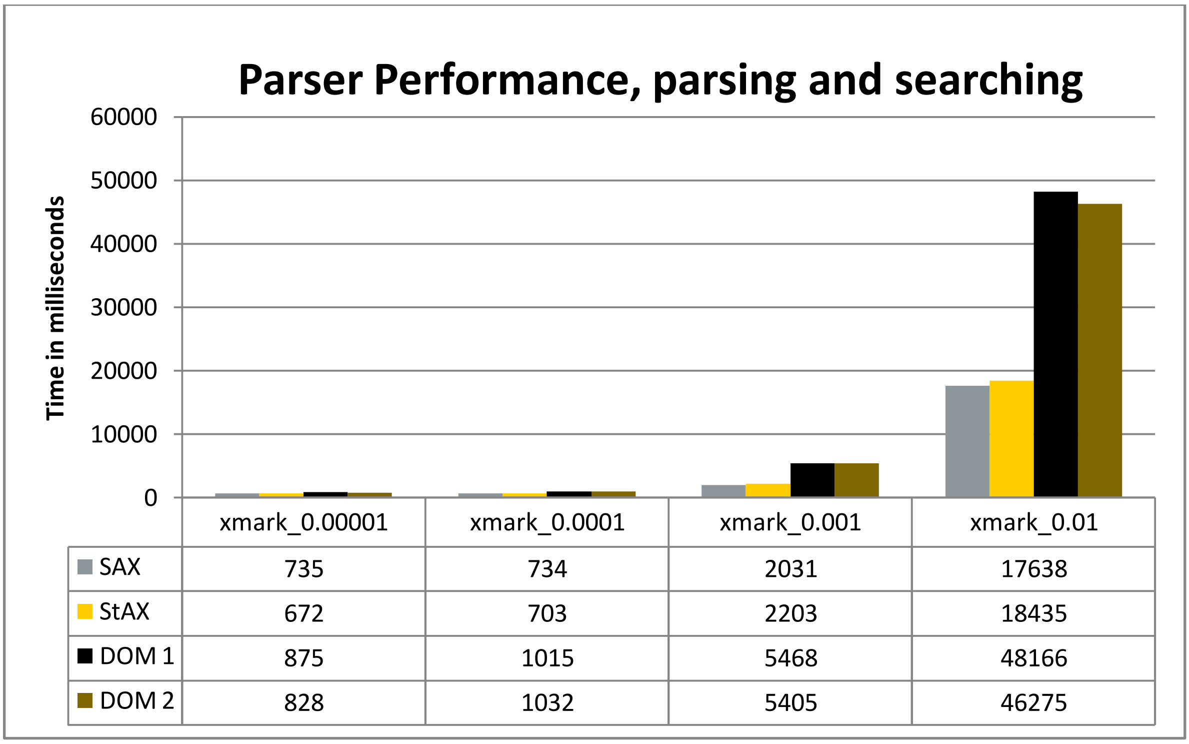
I. Testrun 1, parsing only

parse time in ms	Average SAX	Average StAX	Average DOM
xmark_0.00001	568	705,4	1034,4
xmark_0.0001	530,4	715,2	1140,8
xmark_0.001	1564	2110,2	5399,2
xmark_0.01	14709	18798,6	45138,8



II. Testrun 1, parsing and searching

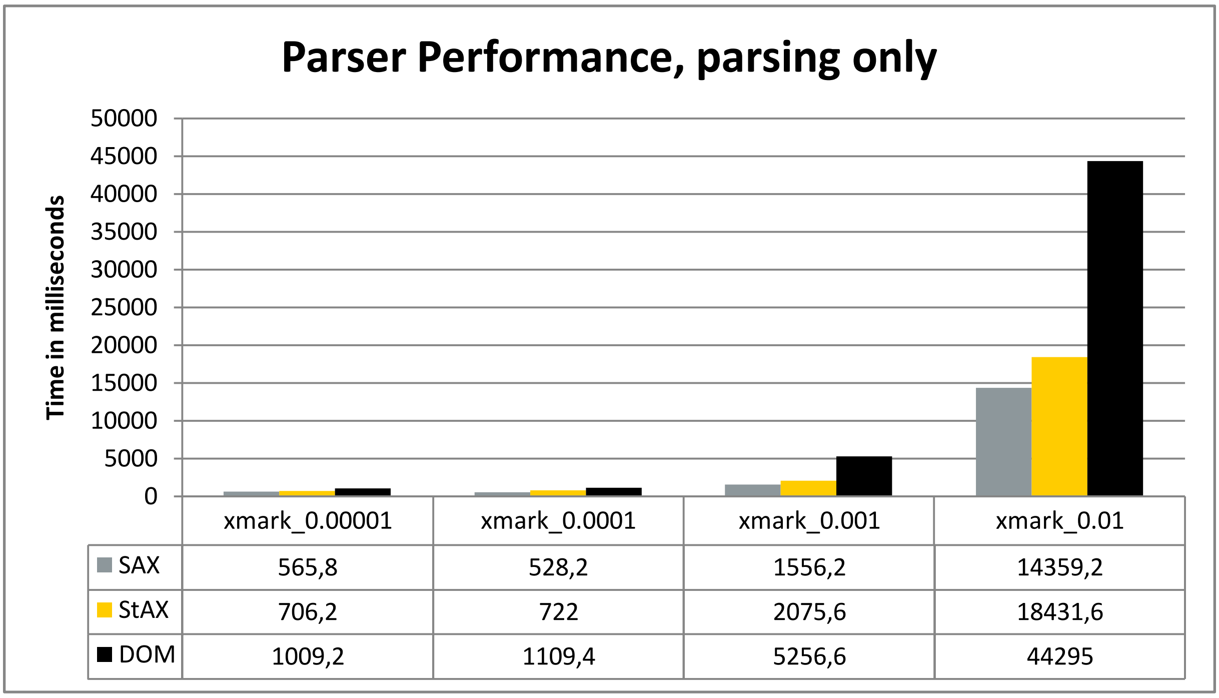
parse time in ms	Average SAX	Average StAX	Average DOM 1	Average DOM 2
xmark_0.00001.xml	735	672	875	828
xmark_0.0001.xml	734	703	1015	1032
xmark_0.001.xml	2031	2203	5468	5405
xmark_0.01.xml	17638	18435	48166	46275



	xmark_0.00001	xmark_0.0001	xmark_0.001	xmark_0.01
■ SAX	735	734	2031	17638
■ StAX	672	703	2203	18435
■ DOM 1	875	1015	5468	48166
■ DOM 2	828	1032	5405	46275

III. Testrun 2, parsing only

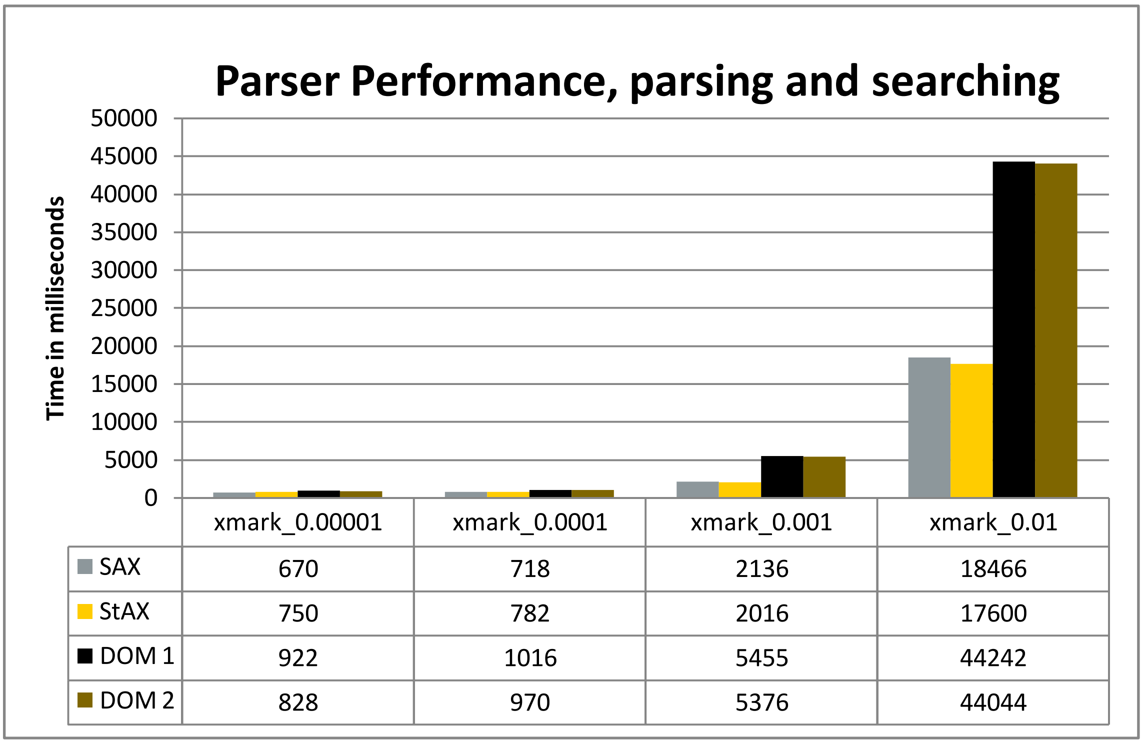
parse time in ms	Average SAX	Average StAX	Average DOM
xmark_0.00001	565,8	706,2	1009,2
xmark_0.0001	528,2	722	1109,4
xmark_0.001	1556,2	2075,6	5256,6
xmark_0.01	14359,2	18431,6	44295



	xmark_0.00001	xmark_0.0001	xmark_0.001	xmark_0.01
▨ SAX	565,8	528,2	1556,2	14359,2
🟨 StAX	706,2	722	2075,6	18431,6
■ DOM	1009,2	1109,4	5256,6	44295

IV. Testrun 2, parsing and searching

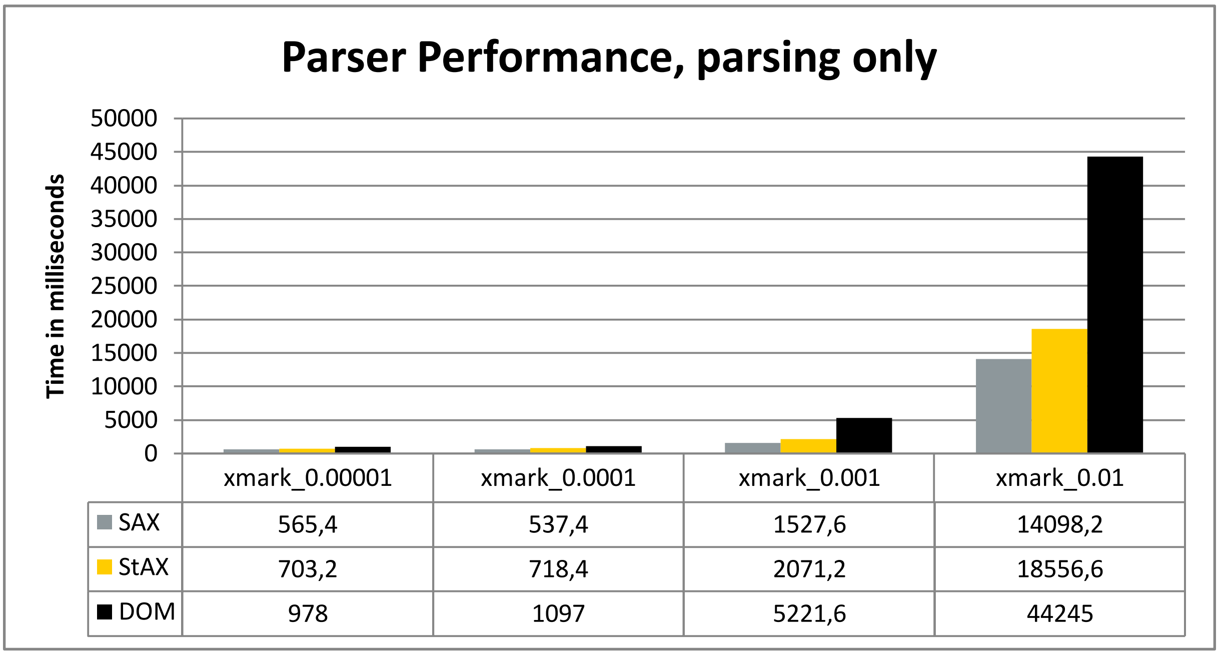
parse time in ms	Average SAX	Average StAX	Average DOM 1	Average DOM 2
xmark_0.00001.xml	750	670	922	828
xmark_0.0001.xml	782	718	1016	970
xmark_0.001.xml	2016	2136	5455	5376
xmark_0.01.xml	17600	18466	44242	44044



	xmark_0.00001	xmark_0.0001	xmark_0.001	xmark_0.01
■ SAX	670	718	2136	18466
■ StAX	750	782	2016	17600
■ DOM 1	922	1016	5455	44242
■ DOM 2	828	970	5376	44044

V. Testrun 3, parsing only

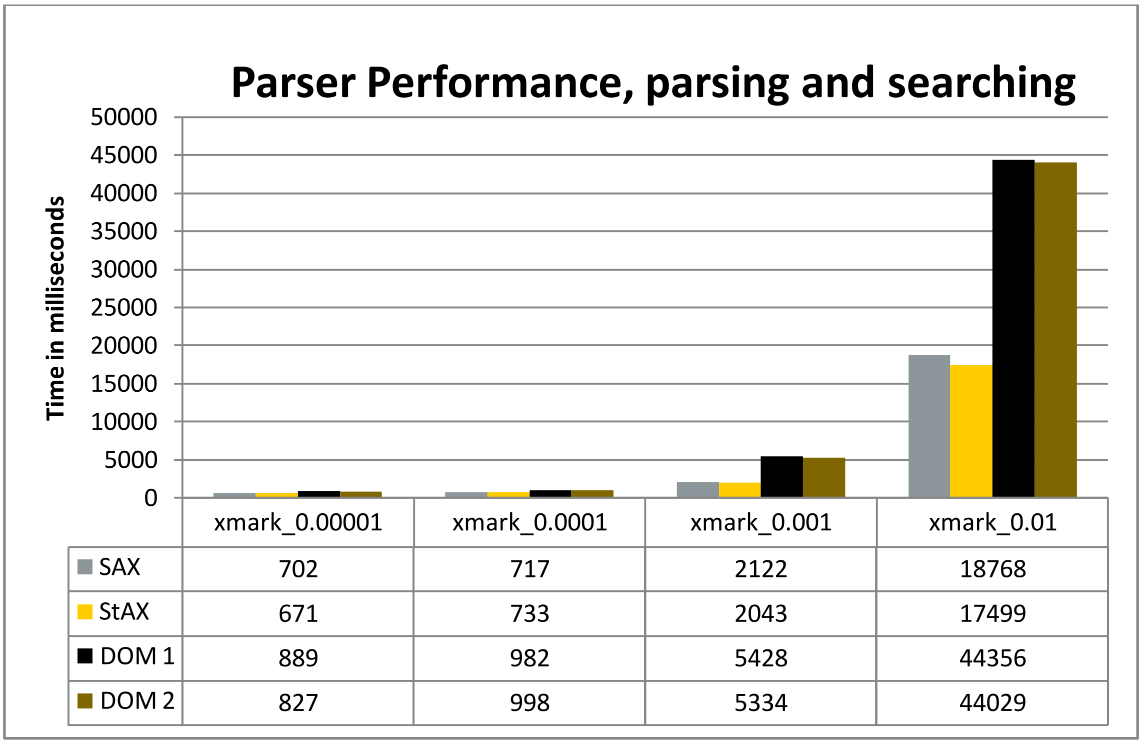
parse time in ms	Average SAX	Average StAX	Average DOM
xmark_0.00001	565,4	703,2	978
xmark_0.0001	537,4	718,4	1097
xmark_0.001	1527,6	2071,2	5221,6
xmark_0.01	14098,2	18556,6	44245



	xmark_0.00001	xmark_0.0001	xmark_0.001	xmark_0.01
▨ SAX	565,4	537,4	1527,6	14098,2
🟨 StAX	703,2	718,4	2071,2	18556,6
■ DOM	978	1097	5221,6	44245

VI. Testrun 3, parsing and searching

parse time in ms	Average SAX	Average StAX	Average DOM 1	Average DOM 2
xmark_0.00001.xml	671	702	889	827
xmark_0.0001.xml	733	717	982	998
xmark_0.001.xml	2043	2122	5428	5334
xmark_0.01.xml	17499	18768	44356	44029



Appendix D. List of Figures, Charts and Tables

Michel Kruiskamp

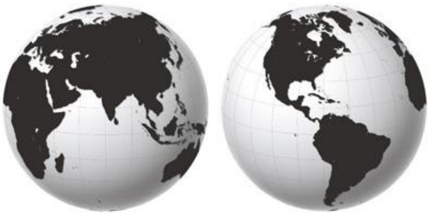
Trainee Working Tomorrow Arnhem

Logica - Releasing your potential

T:	+31 (0) 26 376 5497

M:	+31 (0) 6 45 55 00 54

E:	michel.kruiskamp@logica.com



We're helping the world leader in mobile communications to get to market faster.

We're helping Swedish hospital pharmacies to handle 60,000 prescriptions every day.

We're supporting the missions of a third of the world's satellites.

We're helping utility companies around the world to generate more business.