NTNU
Innovation and Creativity

An Evaluation of messages-based Systems Integration with respect to Performance

A case study

Trond Fallan Smaavik
Nils Torstein Øvstetun

Master of Science in Computer Science
Submission date: June 2007
Supervisor: Reidar Conradi, IDI
Co-supervisor: Jingyue Li, IDI
N N, -

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The assignment contains a litterature study of systems integration strategies and technologies. Two strategies/technologies will be used to implement test applications. The applications will be used to evaluate the technologies with respect to a relevant quality attribute.

Assignment given: 22. January 2007
Supervisor: Reidar Conradi, IDI

Abstract

This report describes a case-study evaluation of two integration strategies with particular focus on performance. The study is motivated by integration challengers within a company we have cooperated with, and our wish to gain insight into systems integration.

The goal of the study has been to evaluate the performance of two message-based system integration strategies. We have evaluated this by implementing several applications which are integrated using either Web services technologies or an integration technology provided by our cooperator. Our research questions have been as follows:
Q1: Which integration solution has best performance in a publish-subscribe scenario?
Q2: Which integration solution has best performance in a request-response scenario?

The results show that the Web service applications has best performance when sending small messages (up to 160kB). For large messages, the applications based on the integration technology from the cooperating company perform better.

The contributions of the study may be split in two. The contribution to the company is the performance evaluation of their technology. Collected data for response time and throughput, and performance models for our test applications are also contributed. For a broader context, we contribute with performance evaluation of Web services technologies. Data is collected for response time and throughput on test applications, and performance models are made. The comparisons of integrations based on Web services and MIP also serves as example of the performance of Web services versus other middleware.

Keywords: Systems integration, Web services, performance evaluation

Preface

This report is written as part of the course TDT4900 Master Thesis at the Department of Computer and Information Science (IDI) at the Norwegian University of Science and Technology (NTNU). It documents the work done by Trond Smaavik and Nils Torstein Øvstetun in the final semester of the study for the degree of Master of Science in computer science. The work has been done in cooperation with a Norwegian software house that do not wish to be mentioned by name.

We would like to thank our supervisor and mentor throughout the project, research scientist Jingyue Li, for good feedback on our work in addition to good discussions. Also a great thank to our software industry partner for letting us get access to their source code and documentation of relevant software products. Discussions with and insight provided by them have been very valuable for us during our work. Finally we would like to thank our teacher of the course, Professor Reidar Conradi, for guidance, feedback and other help.

Trondheim, June 2006.

Nils Torstein Øvstetun

Trond Smaavik

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Our software industry cooperator have created many different applications for business administration [1]. Today there exists no common way of integrating these applications with each other. The company wishes to find a strategy and a software solution to do this. They have looked at several possibilities, and decided to build a new, proprietary infrastructure for message sending between their applications. One of the reasons for this is that performance of integrations is important to the company.

We challenge this solution by proposing that it is possible to achieve the same or better performance by using Web services technologies. To evaluate this, we implement several applications which are integrated either based on the company's technology or Web service technology.

1.2 Motivation

Our work within the field of message-based systems integration is motivated by several factors. The main focus with all systems integration is to make software applications that were not created to communicate with each other do that. This is a challenging task as there exist systems with different structures on different platforms created with different technologies that might need to be integrated. We

[1] The project has cooperated with a Norwegian software house that do not wish to be mentioned by name. The company has provided software for some of the tests that have been performed. Addition to this some employees have served as discussion partners.

wish to gain insight into this complex area to better understand the challenges of it.

Secondly, there is great focus on service-orientation in the software industry today. Many believe that creating integration solutions that are based on services and message passing is a viable way of solving many problems related to systems integration. By looking at different ways to make messages-based integrations we hope to be able to say something about the feasibility of them in different situations.

For our cooperating software industry partner, performance of integration solutions is an important issue. When many applications are integrated, the ability to process messages without causing significant delay is essential. This is a motivation for us to evaluate the performance of different integration approaches to add to the feasibility assessment mentioned.

1.3 Research questions

Our study is based on the GQM approach [31]. Here we briefly present our research questions. The company has identified two typical integration scenarios; we have assigned one research question for each.

Q1: Which integration solution has best performance in a publish-subscribe scenario?
To answer this research question we implemented applications to fit into a publish-subscribe scenario. Two solutions were created as identical as possible, but based on the two different integration technologies. Then we performed performance tests on the applications to be able to evaluate the performance. The important metric for this question was throughput; how many messages the system can process per second.

Q2: Which integration solution has best performance in a request-response scenario?
To be able to evaluate this question, we implemented applications to mimic a request-response scenario. Two solutions were created, as identical as possible, and based on the two different integration technologies. Performance tests were performed to measure the response time; the time that elapses from a message request is sent until a response is received.

1.4 Contributions

The contributions in this study may be split in two.

- First of all we have had a close cooperation with the company, and tried to satisfy some of their concerns in the field. The main motivation to perform performance evaluations is based on the company's requirements for high performance in their integration applications. A contribution to the company is therefore the evaluation of the performance on their product compared with Web services. Data for response time and throughput for applications using their technology is collected. We also provide performance models for application integrations based on MIP, which might serve as a tool for predicting performance of integrations between similar applications.
- Secondly, contributions in a broader context are from the Web service part of the study. We contribute with data collection for throughput and response time for the applications and performance models for Web service based integrations. The comparisons of integrations based on Web services and MIP also serves as example of the performance of Web services versus other middleware.

1.5 Report outline

This report is organized as follows:

- Chapter 2 introduces systems integration and different approaches for this.
- Chapter 3 introduces the problem that this thesis is investigating.
- Chapter 4 describes the integration solution our software industry cooperator has developed.
- Chapter 5 defines our research design using the GQM method.
- Chapter 6 introduces Web services technology that we will be using later.
- Chapter 7 introduces performance engineering methods that we will be using when testing.
- Chapter 8 describes how we have implemented different integration scenarios.
- Chapter 9 gives answers to research question Q1.
- Chapter 10 gives answers to research question Q2.
- Chapter 11 presents our evaluation of the results, the problem and the validity of our findings.
- Chapter 12 presents our conclusions and proposes further work.

Chapter 2

Systems integration

Typically large enterprises consist of hundreds of applications which were never developed to communicate with each other. Applications developed for different platforms with different technologies can be impossible to make work with each other without large modifications. Many old legacy applications have critical roles in enterprise solutions, making companies dependent on these for their day-to-day business. Software integration is about how such application can work together despite the obstacles. Over the last decade, a whole industry sector has emerged, typically known as enterprise application integration (EAI) or systems integration which is about solutions to integrations issues [14]. Today there exist many strategies and tools which provide software infrastructures for software integration. We will in this chapter provide an introduction to system integration, discuss challenges and give an overview of current state-of-the art in integration solutions and technologies.

2.1 Approaches to systems integration

There are different approaches to integrating information systems [22]. Terms like data integration, application integration and business process integration often occur. If we look generally at it, the difference between them lies in the abstraction level. Choosing an integration approach is much about the constraints enforced by the system that are to be integrated, but also about the inherent properties of the different approaches.

2.1.1 Data integration

Data integration is at the lowest level of abstraction we will look at. In this kind of integrations, systems are typically integrated by letting the involved applications get access to each others data. Data integration has the advantage that it is possible to go straight to the data sources (in most cases a database) and extract the needed data. The applications do not need to deal with the application logic of the other system and there is no need to create interfaces between them. This means that it is very easy to establish such integration, and the initial performance is high because no application logic is used. The biggest disadvantage is that if the data model for one of the applications is changed, the other applications using this data model also has to change equivalently. Data warehousing and virtual data warehousing [15] are two examples of data integration methods.

Data warehousing approach integrates disparate data sources by developing a global schema and providing a consistent and unified API to the global schema. Issues with this approach is that data warehouses tends to be very big in size, expensive to develop and currency of data is an issue since the warehouse is refreshed once or a few times per day. Also warehouses typically do not support transactions on the global schema, and for this reason data warehouses is typically used in read-only mode.

Virtual data warehousing also uses a global schema, but it is not populated. Applications specify data access requests as queries on the global schema. This approach tries to alleviate some of the problems with data warehousing - size and currency. This integration method is also limited in its capability to make transactions on the global schema.

2.1.2 Application integration

With application integrations, logic and functionality of the applications is exposed to each other. As long as the interfaces to the applications are unchanged, such integrations allow the internal parts of the systems to be changed without affecting the integration solution. There are many methods/technologies for application integration. Common ways of doing this type of integrations are by using remote procedure calls [15] or message-based integrations [15][16][14].

Remote Procedure Call (RPC) enables an application to invoke a function in another application, which typically runs on different computers. The developer often needs to explicitly address issues related to the differences in the representation of data and network communication protocols. In most cases

the solution is limited to the programming language or platform. There exist several major technology standards for RPC. Object Management Group's standard for application interoperability and integration is Common Object Request Broker Architecture (CORBA). CORBA has heavy foot-print, steep learning curve and lack of a productive development environment, but a good intention of been platform independent. Microsoft's solution to RPC is Component Object Model (COM)/Distributed Component Object Model (DCOM). This technology is complex, and is limited to the Windows operation system. Sun Microsystems introduced Java Remote Method Invocation (RMI) for RPC-based interoperability between applications based on the Java programming language, which is a technology dependent on the Java platform.

Message-based integration solutions have lately become widely used. XML documents have become the preferred format for representation data as messages. The applications using and sending messages must follow pre-defined message schema. The biggest advantages of this are that the involving applications are not dependent on the platform or programming languages it uses. Web services are a commonly used approach to message-based integrations which supports different platforms. Web services can promote the use of more standardized protocol. Ro et al. [23] have identified a list of advantages Web service-based methods have:

- Standard interface support
- Data exchange among heterogeneous systems
- High reliability, extensibility and security
- Easy management of asynchronous handling and transmission fail record
- Management of service categorization system, service register, and search
- Higher productivity and lower cost

Web services will make integration simpler through common transports, data formats and associated services. Web services are transport-independent, provide both synchronous and asynchronous operations, and make it possible through the use of XML documents to create extremely loosely coupled applications.

Software middleware integration solutions

For application integration there also exist several commercial integration middleware applications and tools. There are many kinds of message-oriented middleware (MOM) tools [15], for example extraction, transformation and loading (ETL) tools,

business process management (BPM) tools and integration broker suites (e.g. IBMs WebSphere MQI and Microsoft BizTalk). Implementing message-oriented middleware is often a complex task involving both technological and business challenges which requires appropriate architecture [1]. The tools can be used for either integration of different applications within a single computer or within a locally network or the Internet. The middleware solutions are often very expensive to buy, have high prerequisites and all the applications using it must be modified to meet the requirements of the middleware application.

2.1.3 Business process integration

Business process integration (BPI) is about identifying and orchestrating business processes. Business processes can both be internal (processes within a single organization) or external (including other organizations). Since business processes can be regarded as high-level in this context, they usually are composed of services.

Typically BPI is done by describing the process flow in an XML-language [2]. There are several languages for this purpose, with BPEL (business process execution language) and XLANG among them. Such languages define which services that should be invoked, and whether it should be done in sequence or parallel. Common control structures, such as conditions and loops, also exist. These process descriptions are executed by applications that have support for such, for instance Microsoft BizTalk.

2.2 Service-oriented architecture

Using a service-oriented architecture may also be a way of doing integrations. There is no clear definition of what a service-oriented architecture (SOA) is. Proposed definitions differ in perspective, such as technical or business oriented, granularity and scope. We feel that this statement about SOA describes the phenomenon quite well.

> In short, SOA is about loosely coupled systems, message based communication and business process orchestration. As an abstract architectural model, it acts as an indirection between the business and the technology model. Web Services are the preferred implementation technology for loosely coupled and inter-operable systems. – Beat Schwegler, Microsoft [28]

As we see from above SOA involves both application integration and business process integration.

One of the advantages with service-oriented architectures is that it makes it possible to align business processes with IT systems better than before. Many of today's enterprise IT architectures are application-centric. Several application silos that each are meant for one specific purpose, make up the process support tools for the business processes. With this architecture you get more expensive and less flexible IT systems, as there for instance will be separate data stores for each system, likely with duplicated data. In order to complete business processes, functionality from several applications may be needed. Since the functionality is spread on multiple systems, systems integration is needed [19].

SOA tries to resolve the mentioned problems by adding new levels of abstraction to the enterprise architecture, namely enterprise business services and business processes. The enterprise business services provide high-level functionality needed in the business. These services are put into context as they are composed to and orchestrated by the business processes.

2.3 Integration topologies

There are several topologies used to integrate applications with each other [29]. Integration topologies are different ways of how applications are connected with each other, making it possible to exchange messages/functionalities.

2.3.1 Point-to-point

Point-to-point topologies imply two communicating parties. The sending system must know the location of the receiver to send a message. Often the message contents must first be translated to a format understood by the receiver. When an application needs to communicate with a new application, a new integration solution must be created specifically between the two involving applications. This means when n applications needs to communicate with each other, the number of integrations is given by $\frac{n*(n-1)}{2}$. This is illustrated in figure 2.1 with four parties yielding six integrations. Using this topology may require much programming work, and possibly generate duplicated code in the different applications. In short, point-to-point integrations are easy to implement between few applications, but hard to maintain and scales very poorly.

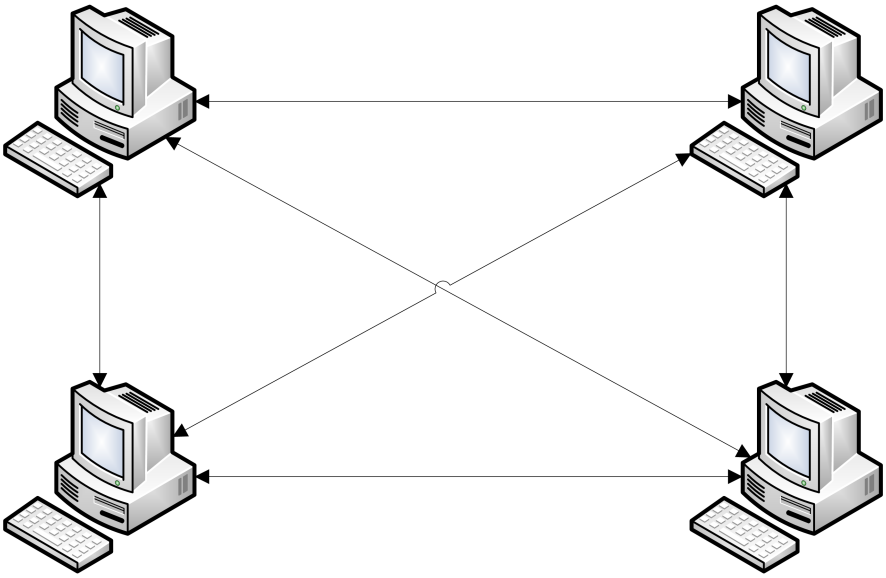

Figure 2.1: Point-to-point topology

2.3.2 Bus

In a bus topology, the integrated applications are connected to a module known as a bus. The applications communicates by sending messages to the bus, which in turn distributes the message to the applications connected to it, as illustrated in figure 2.2. In the simplest type of bus using, a broadcast style communication is used, sending the messages to all the applications connected. It is then up to the applications themselves to determine if the message was meant for them. Common to all message buses is that the connected applications must know the message schema and command messages of the bus in order to use it.

There are more advanced and more common ways of using a bus than the simple broadcast style described over. By using a publish/subscribe pattern, there exist control over which applications that shall receive specific messages. This reduces the traffic on the bus. Two other configurations for the publish/subscribe pattern are list-based and content-based.

List-based publish/subscribe

This pattern is also known as the observer pattern [13]. An identified subject keeps track of subscribers in a list. When an event occurs, all subscribers on the list are notified by calling the update-method on them. The subject has methods for subscribers to attach and detach.

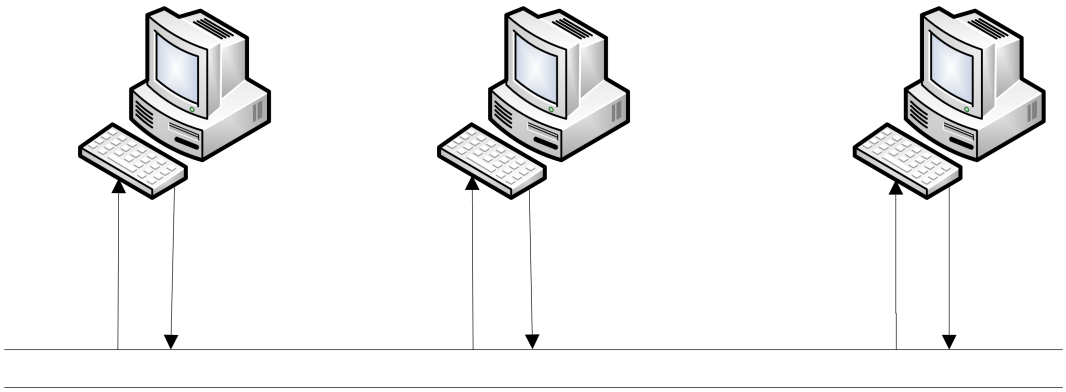

Figure 2.2: Bus topology

Content-based publish/subscribe

Both broadcast and list-based publish/subscribe use communication paths based on predefined categories. In the content-based pattern the messages are examined to determine where to send them. For example, different combinations of information from the message can be used in a decision tree to determine the receivers of the message. This pattern opens for more complex and flexible rules for message handling.

2.3.3 Hub

This topology is based on the broker pattern. The purpose of the broker pattern is to decouple the source and destination systems by taking responsibility for administrating the communication between them. In this process there are three tasks.

Routing involves determining the location of the receiver.

Endpoint registration deals with letting the communicating parties register themselves with the broker so it can contact them.

Transformation is about converting data from one format to another.

There are two main types of brokers, direct and indirect. A direct broker only provides the location of the receiver. The sender is then given this location and does the rest of the communication on its own. An indirect broker is responsible of forwarding all messages from the sender to the receiver. The direct broker typically has better performance, while an indirect broker gives you better control over the communication and the messages sent.

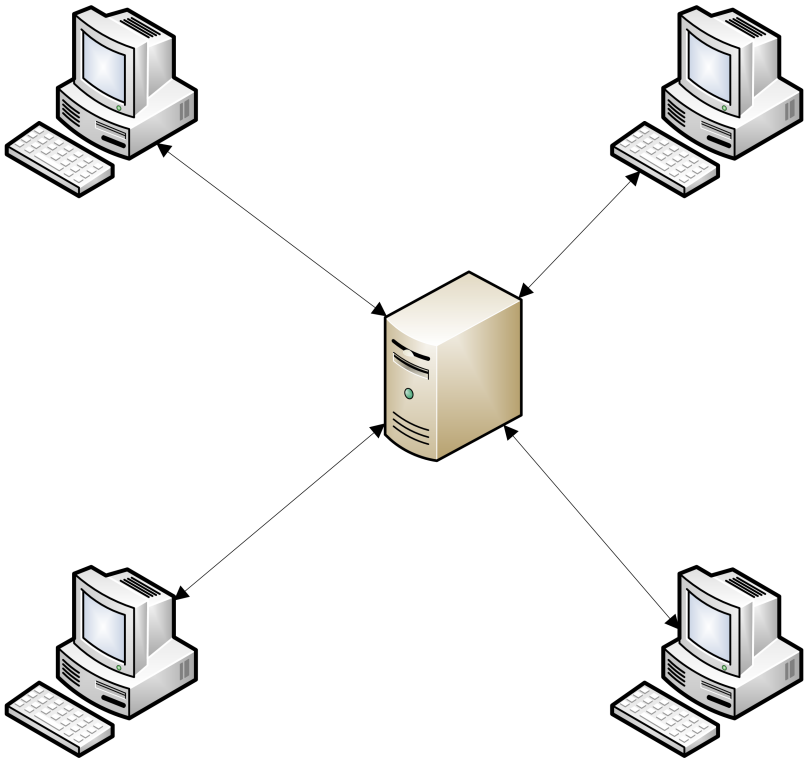

Figure 2.3: Hub topology

2.4 Challenges of System Integration

There are a lot of challenges that remains to be solved within EAI technologies. The challenges involves [14]

Scale. New applications require rapid access to multiple data sources, and the ability to rapidly fuse data from disparate formats. Integration technologies need to have the ability to scale to handle large numbers of data sources with flexible, fast and scalable format transformation engines.

Dynamic configuration. Typically EAI technologies must be statically configured to interact with an existing data source, which requires an adapter component to be acquired, configured to produce and consume messages and deployed. Ideally it should be possible to locate a new data source, automatically deploy or generate an adapter, and make the data immediately available.

Semantics. Some modern programming technologies support dynamic discovery of the syntax of a service or object interface, making it possible for clients to connect to a server, retrieve the operation and parameter names/types, and dynamically construct calls. The problem with this is that do not convey the meaning of the operation and parameter names, so the same name can be used in different interfaces for completely different purposes. This leads to the need for integration technologies to augment reflective interfaces with semantic descriptions of how the interfaces should be used.

Finding relevant data. Most data sources are not accompanied by searchable catalogs and meta-data that can be used to locate data items of interest.

Chapter 3

Problem introduction

In this chapter we will introduce concrete challenges a Norwegian company has with application integration. The company is a leading software house in Norway, with focus on administrative applications. This chapter is based on discussions with developers and project management in the company and internal papers which describe some of the concrete integration challenges the company has. First we describe the challenges the company has with integration. Then we describe two conceptual integration scenarios the company has and discuss the requirements the company have to a integration strategy. Then we discuss integration strategies for the company.

3.1 Integration challenges in the company

Due to a history of acquisition of smaller companies, the company has many applications with different architectures and features. Making these systems work together is a big challenge. There are several ways of solving this challenge, each with its cons and pros. Today the company has various point-to-point integrations between the applications, which create very tight couplings between the applications. Especially when multiple applications needs to cooperate with each other, the need of many point-to-point integrations is an expensive solution. Even between two applications these problems exist when there are multiple versions of each application which needs to be able to talk with each version of the other application. Dealing with different integration technologies further complicates this picture.

Today, the company uses different integration solutions for different applications. Some applications works directly on the target systems database, some applications is based on DCOM while other is based on Web services. Especially where applica-

tions work directly on a shared database, problems occur when the database model needs to modified. This means that every integration made toward it also must be altered according to the new data model. These different point-to-point solutions are also based on different authentication procedures. For a programmer's point of view, knowledge of several integration technologies and several authentication models is needed.

3.2 Integration scenarios

The company wants to adopt a common integration solution for all of their applications. There exist several applications, and the functionalities of them are very different, so the solution needs fit all the applications. Some applications are real-time applications, with an end user demanding fast responses (real-time). Other applications may need an integration solution to synchronize a big database, working in the background. To elaborate the requirements of an integration solution for the company, we will introduce two conceptual integration scenarios which are typical in the eyes of the company.

	Parties	**Communication pattern**	**Topology**
Scenario 1	Two or more servers	Eventing	One-to-one, one-to-many, many-to-many
Scenario 2	Client and server	Request-response	One-to-one

Table 3.1: Integration scenarios

3.2.1 Scenario 1: Server to server notification

This is normally not a real-time scenario. A change is being done in one server, and is forwarded or propagated to one or more servers. An example is updating of customer data. A client application updates customer data in server application "A", which simultaneously informs server application "B" about the updated customer data.

The integration topology can be one-to-one, one-to-many or many-to-many. Publish-subscribe are an integration pattern which is very useful in this scenario.

3.2.2 Scenario 2: Client to server data request

This is often a real-time scenario, where a user waits for a response. In this scenario, a client application typically requests a service from a server application and waits for the response. An example is if a CRM system asks an ERP system for a data set with customer information.

The integration topology is one-to-one, and request-response is the most useful communication pattern for this scenario.

3.3 Requirements for a common integration solution

The overall goal for the company is to create a standardized software infrastructure for application level integrations between the company's applications. The possibility to create integrations with third party applications is an extra benefit, but the primary target is the company's own applications. The company has some important requirements for the solution:

Loose coupling. This means in practice that one system, module or function can be changed or reused without affecting other systems, modules or functions.

Strong coherence. The company wants each component to have a clear responsibility without mixing different tasks.

Security. It must be possible to ensure data integrity, confidentiality protection and access control in machine-to-machine scenarios.

Performance is very important for the company. Many of its application need to have real-time integrations in client-server communications.

Easy installation and configuration. The solution must be easy to install and use for the company's customers. The aim is a solution that can be deployed without consultant assistance for installing and/or configuration.

3.4 Integration strategy alternatives for the company

We will in this section present different integration strategies the company has evaluated.

3.4.1 Current integration strategy

Here we describe how the company's integration strategy has been the last years. The technology used has not been standardized, so the integration solutions are pure one-to-one integrations based on different technologies. The company has some integrations working directly on the systems database. This is particularly problematic since the application developer must understand the data model of the target system, the integrations are vulnerable to future modifications in the data model and it is hard to achieve security without making the installation burdensome. Other integrations are based on DCOM and other are based on Web services. The different technologies require different forms of authentication, which implies that application developers also must have knowledge of several integration technologies and several authentication models.

The main advantage with this integration strategy is that it is easy to initial make integration between two applications. It is also easy to create integrations with very high performance, for example an application working directly on another applications database will have higher performance than integration based on shared interfaces.

3.4.2 Chosen strategy: Custom made integration infrastructure application

This strategy involves developing a custom made software application/module to serve as an integration infrastructure based on the company's requirements. This gives the company full control over the middleware which will be used, and all the needed features can be implemented. The solution can be developed with particular focus on the most important non-functional requirements, like performance, security, modifiability etc. Also, dependent on the solution developed, the software can support all the required integration topologies like one-to-one, one-to-many and many-to-many. The main disadvantage with this solution is the development cost associated with it.

The company has evaluated this strategy as the best solution to their needs. An integration application based on message bus pattern is developed. Details about this application are given in chapter 4.

3.4.3 Not applicable strategy: Commercial middleware integration product

This strategy is basically to buy a middleware integrations application or use an open source middleware application to achieve the needed integration infrastructure. Products like Microsoft BizTalk, Microsoft Message Queue and ActiveMQ are examples of such products. There exist many such products, so the biggest advantage with this strategy is low or non development cost with the infrastructure. The different products have different features. For example, some solutions provides infrastructure for both one-to-one, one-to-many and many-to-many integrations. Also features like security, reliable communication, publish-subscribe solutions are available in many of the products. The products are in most cases widely used within many enterprise solutions, which means they are widely tested and stable. Also a big advantage with buying a product is if there exist support for the product, and common user problems are often described in detail on the Internet.

The main problem with this alternative is to find a product which fulfills all the requirements the company has. Other requirements like runtime cost, easy installation, small footprint and low deployment prerequisites is important for the company. Another important issue is if the products are platform dependent and how much the involving applications need to be modified to support the integrations middleware product. Another factor is if the customers of the company also need to buy and use the middleware product in addition to the company's products.

The company has evaluated several products, including Microsoft Biztalk, Microsoft Message Queue and ActiveMQ. None of the products fulfilled all the requirements the company have to its integration infrastructure. Especially the requirements regarding usability are an important factor why such products do not fulfill their needs. The biggest disadvantages with the evaluated products was that the they are dependent on particular versions of Windows, difficult to install, require much hardware resources or require the company's customers to buy the product.

3.4.4 Competitive strategy: Web services

This strategy is to use a standard commercial communication integration technology for all applications within the company. It is important that the integrations are loosely-coupled and that the services provided has strong coherence in such a realization. The main advantage with this strategy is that it uses a standardized technology between all applications, so the developers only have one technology they need to learn. Web services is the technology which provides much of the needs the company have. Web services are easy to implement, and since it are commonly used on the Web today it will be easy to integrate external third party applications using Web services with the company's applications.

Web services are a pure point-to-point technology, but there exists more advanced commercial solutions to create integration patterns such as publish-subscribe based on Web services. This opens up the opportunities to create one-to-many topology integrations in addition to one-to-one integrations.

The company has evaluated this strategy to be competitive with the strategy they implemented. We use this strategy later in our study. Details about Web service is given in chapter 6.

3.4.5 Summary of alternatives.

The table below summarizes the described strategy alternatives.

	Benifits	**Shortcommings**	**Company's view**
Current solution	Posible to make fast integrations.	Hard to maintain. Bad scalability. High learning threshold.	Not desirable.
Custom made	Can implement needed functionality. Ownership of code.	Development costs. Proprietary.	Viable.
Commercial products	Mature products. Much functionality. Support.	Expencive for customers. Resource demanding.	Not applicable.
Web services	One single technology. Ease of third-party integration. Extensebility. Interopability.	Security dependent on each implementation.	Posible, but suboptimal.

Table 3.2: Summary of strategy alternatives

Chapter 4

Message bus implementation

In this chapter we will first give a technical introduction to the integration middleware the company has developed. Then we will describe how this program fits into a SoA context.

4.1 Backgound

As described in chapter 3, the company is developing a middleware solution to serve as an infrastructure for their integration with their applications. The solution is developed mainly with the aim of being used with administrative applications, and to fit the requirement the company has with such product. One of the main problem with the previously integrations the company uses, is that it is based on pure point-to-point between the applications. The new product is based on the message bus pattern, and for further reference we call the product "Message bus based Integration Product", MIP.

In the message bus pattern, the publish/subscribe-mechanism is inherent. With this pattern, many applications can subscribe to a service. When an event is created, the event source sends this event to all the subscribers to the service. This is convenient in integration of administrative systems as many of the systems shares entities, for example shared *customer data*. When one application updates information about a customer, this change can easily be shared to the other applications that use information about the same customer.

The program has a lot of functionalities. By including functionalities like reliable messaging and security mechanisms on MIP, the application developers do not need to focus on implement these kind of important functions to the applications.

4.2 Technical details

4.2.1 MIP overview

4.2.2 MIP routers

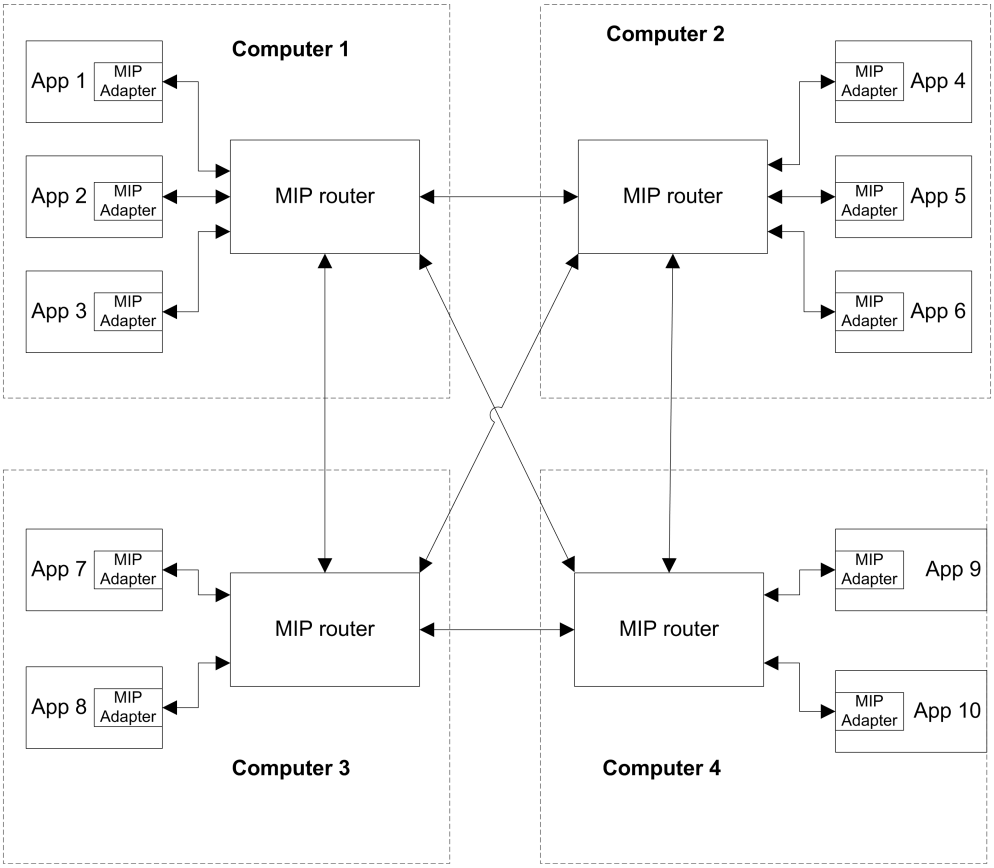

Figure 4.1: MIP routers overview

The primary aim for MIP is: "'to create a standardized software infrastructure for application level integration between the products in the company"'. Although the company s own applications is the primary aim for the MIP, the solution is designed so other external third parties application also can connect with the bus in future version of the program.

Every application which uses the MIP needs to implement a MIP adapter. This adapter connects to the MIP and handles all the communication between the application and the bus, as illustrated in 4.1. A big advantage with this solution is that

the application does not need to know anything about the internal implementation of the service it uses. The only thing the application needs to know is which service it shall use, which channel this service is provided and which schema this channel uses. In this way the application does not even know what server the service is provided from.

From a programmer's point-of-view, a simple, powerful API is offered. Typically functions offered by the API are a function to subscribe for messages and a function for sending a message on a channel. A big benefit with this is that complex operations like reliable messaging and secure communications is hidden for the programmer.

Every computer using MIP needs to have a MIP router process running. The MIP router is written in Microsoft .NET, so the only platform currently supported by MIP is Windows. However, the messages are in general in platform-independent XML formats, so it is possible to use the services provided if a "'platform-adopter"' is created. Every computer running application that are using MIP must have a MIP router installed, which communicates directly with every MIP-enabled applications on that machine. If two applications on a machine communicate with each other, for example if App 1 needs to communicate with App 2 in figure 4.1, the communication is App 1 -> Router -> App 2. If applications on different machines communicate with each other, the communication would be for example App 1 -> Router computer 1 -> Router computer 2 -> App 4.

Routers can run in either server or client mode. A server application can only communicate with server mode routers, while a client application can communicate with both kinds of routers. All MIP routers are connected with all other MIP routers. This means that physical the MIP network is connected in a star topology. This means that there is only one "hop" between the computers. Logically, from applications point-of-view, the network is a bus topology.

4.2.3 MIP Adapter

Applications communicates with MIP via an "'MIP-adapter"', so all applications which communicates with the MIP network needs to implement such an adapter. In this context, the term adapter is all the code developed to integrate the business logic of the application and the MIP API. An adapter thus provides and consumes services on the MIP network.

4.2.4 Channels

MIP uses channels for communication between applications. There are three channel types defined:

Command channels are used for request-response communication. A request is sent to a specific receiver, which sends a response message back to the initial sender.

Event channels are used to send messages to all subscribers of the channel. The publisher of the message does not get any confirmation that the message has been received, and it is not guaranteed that the message is delivered.

Reliable event channels act the same way as event channels, but guaranties that the message reaches the subscribers within a configurable time span. As for the event channels there is not given any response that the message is received.

Each channel has a channel definition which describes what kind of channel it is and the message scheme the messages on the channels must follow. Channel definitions also specify whether the messages on the channel should be encrypted, and if the channel is open or restricted. Restricted channels can only be used by server mode routers.

It is also possible to set channel properties. These are key/value pair that can be used to distinguish channels with the same definition. Such properties can be used for filtering, and could for instance be *company*. Two channels named *Order* belonging to different companies could then be differentiated.

When an application subscribes to a service, it is in MIP a channel it subscribes to. When a new message is sent on the channel, it gets broadcasted to all the MIP adapters which subscribes to that particular channel.

4.2.5 Messages

There are four types of messages used on MIP:

Event messages are the type of messages used event channels for publish-subscribe communications.

Command messages are the request message sent on a command channel from sender to the receiver.

Response messages are the response message returned to the sender on the command channel from a command message.

Fault messages are a response message sent by MIP when a normal response message can not be obtained.

A message consists of two parts; a body and a header. The *body* must be set according to the schema of the channel. The company recommends that the message body follows a XML schema, but binary forms and objects can also be sent on MIP. The message *header* contains information the MIP needs to be able to route the message. Serializing and deserializing of the message header is done independent of the message body. This is because the router needs to deserialize the header before it is sent to the applications. The body is then deserialized at the consuming application.

4.3 MIP functionalities

4.3.1 One-to-one communication

MIP has the functionalities to support traditional one-to-one request-response communication. This is done by using the command channel and command/response messages.

4.3.2 Many-to-many communication

One of the most usefully functionality with MIP is the many-to-many eventing communication. This is handled with using the event channel and event messages.

4.3.3 Reliable many-to-many communication

As mentioned before, MIP has built-in functionalities for guarantee message delivery through the reliable event channels. This means that a service can send events to a subscriber without being up and running simultaneously. There are two scenarios where this function is used.

- A messages is sent from a service, using reliable messaging. The subscribing application is unavailable at the time the message is sent. The message is first sent to the MIP router on the computer with the unavailable application.

When the application is available again, the router sends the message to the receiver.

- A message is sent from a service to a MIP router which is unavailable . The message is stored on the MIP router with the service and sent to the next MIP router when it gets available again.

This function is implemented by having a database implemented on each router. All undelivered events is stored in this database till the messages can be delivered to the next step. In worst case the receiving computer or application will never come online again. In this case the event message is forwarded to a dead letter queue after a configurable timeout period (may be days or weeks).

4.3.4 Automatic discovery

A standard MIP network consists of all MIP routers within a physically local area network. Each MIP-router has build-in mechanism to automatic discover and connect to the existing MIP network. It is also possible to host several physical MIP network within one local area network or several logical MIP networks within a physical local area network. However, to host a non-standard MIP network, manual configurations of the routers are needed.

4.3.5 Security

There are several security functionalities within a MIP network, which by default is activated. This functions guarantees that all communications on the MIP is integrity protected. This means that all messages are authenticated to ensure that the message is from a valid sender and has not been modified in transit. All messages is also controlled to be in conformance with the dynamic configuration of the logical MIP network. It is also possible to configure that all messages on a certain channel are encrypted.

4.3.6 Distributed configuration

All MIP routers share the channel configurations among them. This means that when a new application is connected to a router, this router spreads that application's configuration to the other router so that the new application can be reached. The configuration of a MIP network consists of two types:

Fixed configuration. This consists of the channel definitions that different application types supports for publishing and subscribing. Each application using MIP provides a fixed configuration to the MIP network when installed.

Dynamic configuration. This consists of the configuration of all channels on a logical MIP network. It also defines which application instances that are allowed to publish and subscribe to a channel. The dynamic configuration is changed via a configuration API or a management console. This can only be done from a machine with a MIP server mode router. When the dynamic configuration files on one server mode router is changed, the updated configurations is distributed to all MIP routers on the network.

4.4 Evaluation of MIP in SOA environments

In this section we will look at how well the MIP is suited in a SOA environment. What properties that makes it desirable to use in different situations and what weaknesses it has.

4.4.1 Strong sides

The MIP has several properties that makes is suitable for a service oriented environment.

Messages based. Using XML messages, and not a proprietary format, makes it possible to easily create new message schemes and extend existing ones.

Transport independent. The MIP uses Microsoft's Windows Communication Foundation (WCF) for communication. By doing this the transport mechanism can be changes, and it will be possible to use HTTP transport that is the most common for Web services in a service oriented architecture.

4.4.2 Weaker sides

Some of the properties of MIP is of a type that it does not directly plug into all other service oriented architectures.

Dependent on local adapter. In order to make use of MIP it is required to implement an MIP-adapter in all the applications which uses MIP. This has to be done to implement the publish/subscribe functionality in MIP.

Platform dependent. Since MIP uses both adapters and routers which is dependent on the Microsoft .NET environment, the MIP is dependent of the Windows platform.

Chapter 5

Research design

This chapter presents the research agenda and focus of this project. First we elaborates on the motivation for the research before the research method we are using is explained. Then we present the Goal, Question and Metric part of the research question. Then we describe what outcome we wish to get from the study.

5.1 Research motivation

As explained in chapter, 3 the company we cooperate with develops administrative software applications, and has some challenges with integrating of their applications. They have developed a middleware integration software application (MIP, see chapter 4) which serves as an infrastructure solution to their integration needs. Before the company decided to create MIP, they evaluated alternatives to fulfill their requirements. The most promising alternative for the company was to create their integrations based on Web services technologies. The literature also shows that message-based integration solutions are a suitable strategy for the company's needs, and that Web services is a commonly used approach.

We will in the study compare the two technologies by developing prototype applications based on typical integration scenarios the company has, described in 3.2. For each of the scenarios we will develop two solutions, one based on MIP and one based on Web service technologies.

5.2 Research framework

The research approach used in the study is based on the Goal-Question-Metric (GQM) method. The GQM approach have three levels[31]:

- Conceptual level (Goal). The goal for the study is defined.
- Operational level (Question). Research questions which will help achieve the goal are defined.
- Quantitative level (Metric). After the questions are defined, we associate the questions with appropriate metrics.

5.3 Goal definition

The company considers the performance of the integrations as a very important factor to determine which is best suited for their environment. We will therefore use performance as the quality of focus in the comparison. Thus we will perform a performance evaluation of the prototype applications we develop.

Following the GQM framework, we define the research goal as:

> **Evaluate two integration strategies with particular respect to performance in a service-oriented context by implementing them using an integration middleware product (MIP) and Web services technologies.**

We use the goal definition template [31] to elaborate on the goal:

Object of study. The study objects are two integration technologies.

1. The MIP middleware integration application created by the company.
2. Web service technologies.

Purpose. The purpose of the study is to be able to compare the two technologies.

Quality focus. The quality focus is the performance of the technologies.

Perspective. The results are interpreted from an architectural perspective.

Context. The context will be the prototype applications we create.

5.4 Research questions

In order to accomplish our goal we have defined research questions. Question 1 and 2 are based on the integration scenarios of the company, described in chapter 3.

Q1: Which integration solution has best performance in a publish-subscribe scenario?
To answer this we need to develop applications suited for a publish-subscribe scenario. We will develop as identical applications as possible, which are based on the two different integration strategies. This will mean we need to create four applications; client and server using MIP and client and server using Web services. We will call the MIP application developed to answer this question for MIP_Q1App and the Web service application for WS_Q1App.

Q2: Which integration solution has best performance in a request-response scenario? Just like with Q1, we will develop software applications fitted to use in a request-response scenario based on the two different integration strategies. We need to create four applications, client and server with MIP and client and server with Web services. We will call the applications for MIP_Q2App and WS_Q2App.

5.5 Data collection and measurement metrics

For the performance evaluation in Q1 and Q2 we will measure the following metrics.

Response time. Response time is the time it takes for a system to react to a specific action or request. The unit of response time is seconds. This metric will be used for Q2.

Throughput. The number of operations completed per unit of time is called throughput. We will for Q1 measure this to see how many operations the system can handle per second.

Memory usage. By looking at memory usage we can see how well the systems scale when the workload increases. We measure memory usage in megabytes or percent of total memory available.

CPU utilization. How much processing power is needed to handle different amounts of requests also tells us something about the scalability of the systems. We measure the utilization as a percentage of available CPU power.

5.6 Research outcome

Based on our results from research question Q1 and Q2, in addition to the literature study, we will evaluate the two integration strategies and make a recommendation for the company for which is better. The requirements the company has, presented in section 3.3, will serve as a guideline for our evaluation.

5.7 Summary of the research design

To ease the reading of the thesis we present how the following chapters are related to our research design.

- In chapter 6 we present Web services and related technologies as this will be used for implementation.
- Chapter 7 introduces general performance engineering theory and a framework we will base our testing on.
- We describe the implementations of the different prototype models in chapter 8.
- The performance evaluations of the prototype models and results are presented in chapter 9 and 10 for Q1 and Q2 respectively.

Chapter 6

Web services

In the later years Web services has become a popular technology for software integration, mainly due to properties such as interoperability. We will in this chapter describe the technical parts of Web services.

6.1 Background

In short, Web services are a technology which offers message-based machine-to-machine services available over the Internet. Because of the ubiquity of the Internet and the HTTP protocol it is possible to use the World Wide Web to offer business services through Web services. Web services is a platform independent technology, which makes it especially suitable for this.

Web services are often mentioned alongside service-oriented architectures (SOA). Even though Web services can be used to implement service-oriented architectures, it is not the only possible technology. Neither does use of Web services create a service-oriented architecture. While SOA is a way of aligning business and technology, Web services is only a technology.

The World Wide Web Consortium defines Web services like this [7]

> A Web service is a software system designed to support inter operable machine-to-machine interaction over a network. It has an interface described in a machine processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.

Before we go into detail about the most common technologies that are used to create Web services, we will explain how they relate to each other. The technology protocol stack for Web services include extensible markup language (XML), simple object access protocol (SOAP), Web services description language (WSDL) and universal description, discovery, and integration (UDDI), as shown in figure 6.1. It is possible to query a UDDI service to get a Web service description in the WSDL format. The WSDL contains location and interface information. Because of this is it possible to create a SOAP message that is sent to the Web service using some kind of transport protocol [5]. The most common transport protocol is HTTP.

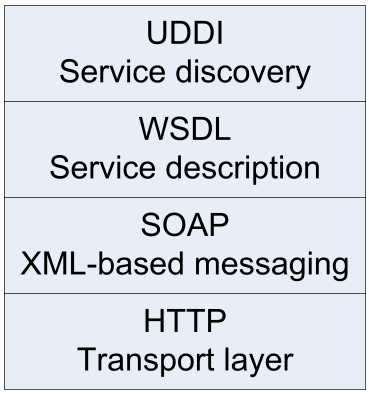

Figure 6.1: The Web service protocols

6.2 Web service technologies

The following section describes the currently most used technologies and standards that are used in Web services [4].

6.2.1 eXtensible Markup Language (XML)

XML is a text based data description format [26]. The format has several uses within Web services, from being the message format to description of available services. One of the big advantages of XML is its flexibility. In to being machine processable it is also easily human readable, even with no prior knowledge.

```
<?xml version="1.0">
<Transportation>
  <Car>
    <Brand make="Mondeo">Ford</Brand>
```

```
    <Color>Blue</Color>
  </Car>
  <Car>
    <Brand>Audi</Brand>
    <Color>Black</Color>
  </Car>
  <Airplane>
    <Brand make="747">Boeing</Brand>
  </Airplane>
</Transportation>
```

Listing 6.1: XML example

The XML language syntax is very similar to the html language. Tags are used to describe an element, "<Element>Element data</Element>". Example 6.1 has a list of different types of transportation. With exception of the first line, which indicates that the document is in XML format, all the tags are defined by the creator of the document. As there are no keywords in the data description, XML documents can not be verified of the correctness of the contents. Document Type Declarations (DTD) and XML Schemas are standards of describing how a document should be written within a context. For our previous example we could create the following schema to describe the rules of how the document should be written.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="Transport" type="TransportType" />
  <xs:complexType name="TransportType">
    <xs:sequence>
      <xs:element name="Car" type="CarType">
      </xs:element>
      <xs:element name="Airplane" type="AirplaneType">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="CarType">
    <xs:sequence>
      <xs:element name="Brand" type="BrandType">
      </xs:element>
      <xs:element name="Color" type="xs:string">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="AirplaneType">
    <xs:sequence>
      <xs:element name="Brand" type="BrandType">
```

```
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="BrandType" mixed="true">
    <xs:sequence>
      <xs:element name="Brand" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="make" type="xs:string" use="required" />
  </xs:complexType>
</xs:schema>
```

Listing 6.2: XML Schema example

By validating XML documents against such schemas, it is possible to be sure that a message contains only valid information. For Web services this means that the messages that are exchanged can be interpreted correctly by consumers of the service. At the top of the schema we find the statement *xmlns:xs="http://www.w3.org/2001/XMLSchema"*. This tells that the schema uses the *http://www.w3.org/2001/XMLSchema* namespace. Locally this is referred to by putting *xs:* in front of the tags of that namespace. Namespaces are used to keep elements with the same name, but belonging to different context from each other. For example, postOffice:Address and memory:Address.

6.2.2 Simple Object Access Protocol (SOAP)

SOAP is an XML based protocol used by Web services as an envelope for the XML data being transferred [5]. The protocol is stateless and one-way. One-way means that a client can make a request to a service(server), but the service can contact the client. More complex interaction between client and server can be achieved by using mechanisms in other protocols or program specific information.

Since the SOAP protocol is XML based it is independent of implementation language and platform. This means that SOAP for example can transport messages from a service implemented in Microsoft .Net to a system based on CORBA running on UNIX. The most common transport protocol for SOAP is HTTP as most firewalls allow HTTP traffic, but the protocol has support for other underlying protocols as well. Examples of uncommon protocol used with SOAP is SMTP[24], SNMP[20] and SIP[9].

In a SOAP document, there are three main elements [5]: envelope, header and the body.

<Envelope> is the root element of the SOAP message, containing an optional

<Header> element and a mandatory Body element. It may include a namespace declaration and an encoding style.

<Header> is optional and can be used to send information that is not considered as application data. This could be information on control flow, how to process the message body or authentication entries. The header information can also target at intermediary nodes. Child elements of header is called header block, and SOAP defines well-known attributes which indicates who should deal with a header block, and whether processing it is optional or mandatory. The details of the header element are open-ended, providing maximum flexibility for application providers. There are two specified header attributes, "'actor"' and "'mustunderstand"'. By specifying the "'actor"' attribute, the client can specify the recipient of the SOAP header. "'Mustunderstand"' is an attribute which indicates if a header element is mandatory or optional. If this is set to true, the recipient must understand and process the attribute.

<Body> is the last and mandatory child element of Envelope, and is a block with the payload to the ultimate receiver. Typical uses of body elements are RPC request and responses.

SOAP is transmitted one-way from the sender to the receiver. Listing 6.3 shows an example of a SOAP document.

```
<?xml version='1.0' encoding='UTF-8'>
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <SOAP-ENV:Body>
    <ns1:getTemp
    xmlns:ns1="urn:xmethods-Temperature"
    SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <zipcode xsi:type="xsd:string">10016</zipcode>
    </ns1:getTemp>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Listing 6.3: SOAP example

6.2.3 Web Services Description Language (WSDL)

WSDL is the first widely adopted mechanism for describing the basic characteristics of a Web services in common XML grammar [5]. WSDL describes interface information for all publicly available functions, data type information for message

request/responses, binding information about the protocol to be used and address information for locating the specific service. The WSDL specification is divided in six major elements:

<definitions> is the root element of all WSDL documents. This element defines the name of the service, declares multiple namespaces used throughout the remainder of the document, and contains the service elements.

<types> describes the data types used between server and client. By default, WSDL uses the W3C XML Schema specification for typing system, which includes types such as strings, integers, Boolean etc.

<messsage> This element describes a one-way message request or response. The name of the message is defined, containing zero or more message part elements. This message part element can be referred to message parameters or message return values.

<portType> combines multiple message elements to form a complete one-way or round-trip operation. A portType can define multiple operations.

<binding> describes specifics of how the service will be implemented on the wire.

<service> defines the address for invoking the specified service, typical a URL.

In addition, there is two utility elements. *Documentation* provides human-readable documentation which can be included inside any other WSDL document. *Import* is used to import other WSDL documents or XML schema, to enable more modular WSDL documents. Listing 6.4 shows an example of a WSDL document.

```
<?xml version="1.0" encoding="UTF-8">
<definitions name="HelloService">
  targetNamespace="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas/xmlsoap.org/wsdl/soap"
  xmlns:tns="http://www.ecerami.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchma">

  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>

  <portType name="Hello:PortType">
    <operation name="SayHello">
```

```
      <input message="tns:SayHelloRequest"/>
      <output messsage="tns:SayHelloResponse"/>
    </operation>
  </portType>

  <binding name="Hello_Binding" type="tns:Hello_PortType">
    <soap:binding style="rpc"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="sayHello">
      <soap:operation soapAction="sayHello"/>
      <input>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examlpes:helloservice"
          use="encoded"/>
      </input>
      <output>
        <soap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="urn:examples:helloservice"
          use="encoded"/>
      </output>
    </operation>
  </binding>

  <service name="Hello_Service">
    <documentation>WSDL File for HelloService</documentation>
    <port binding="tns:Hello_Binding" name="Hello_Port">
      <soap:address
        location="http://localhost:8080/soap/servlet/rpcrouter"/>
    </port>
  </service>
</definitions>
```

Listing 6.4: WSDL example

6.2.4 Universal Description, Discovery, and Integration (UDDI)

UDDI is a technical specification for describing, discover and integrating Web services [5]. UDDI can be described as the part of the web service protocol stack where companies can publish and find web services. There are two main parts of UDDI. *The technical specification* of UDDI is for building distributed directory of businesses and web services. Specific XML formats are used for storing data, and the specification includes API details for searching existing data and publishing new data. *The UDDI Business registry* is a fully operational implementation of the UDDI specification, launched by Microsoft and IBM in 2001. The UDDI registry makes it possible

for anyone to search existing UDDI data, and enables companies to register itself and its services. The register is divided in three categories. *The white pages* describe general information about a specific company, ie name, description, contact information and business identifiers. *The yellow pages* consist of general classification data for the company or the service, ie industry, product or geographic codes. *The green pages* provide technical information about Web services, ie pointer to external specification and an address for invoking the web service.

6.3 Web services challenges

Web services is a relatively new technology, thus there are still some issues to be solved before it can be called a mature technology. New standards are being proposed and developed all the time in order to do something about the weaknesses. Especially issues related to quality of service have been worked on, and there are now proposed solutions for reliability and many security related problems. Still Web services have some challenges when it comes to performance compared to other distributed communication technologies [17].

One challenge in the vision about a fully automated Web service integration between applications, is that even though it is possible to query a UDDI service a machine cannot assess the suitability of the service. There might be many Web services that match given criteria, but there is no way to know anything about the semantics of the services. This judgment requires human interaction [5].

6.3.1 Web service versioning

A problem that often occurs when using distributed applications is the *API versioning issue* [3]. Web services are no exception. In addition there is the issue of changing message formats, in other words changing XML Schemas. Versioning has not been included in the Web service architecture. How can Web services be created in such a way that a change in it does not cause service consumers to be unable to communicate with it?

When making or maintaining a Web service there are different types of changes that should be accommodated. [12].

Forward compatibility means anticipating changes. A new service can be deployed without breaking the existing consumers. A service must be able to interact correctly with unknown features.

Backward compatibility is easier to accomplish than forward compatibility. It means that consumers of old services still can function when interacting with the new service. This is done by keeping existing functionality when expanding the service.

Message versioning

Message versioning builds on techniques for XML extensibility and versioning. Possible solutions are use of namespaces, extension elements or custom version attributes.

Namespaces. When using namespaces, different namespace can be used for each version. This should only be used after major changes, as it will mean a new XML Schema. XML Schema does also offer an optional version attribute. Setting this will leave the namespace untouched and it is easy to implement. The problem with this approach is that parsers are not required to validate on this attribute, with means you have to add custom validation code.

Extension elements. By adding a special element where extensions can be places you can keep the same namespace as before. The problem with this way of solving versioning is that it adds complexity to your schema and you can not group new tags with related ones.

Custom version attributes can be added to you schema, as the standard one is not validated. The problem can be solved by adding a required attribute with a fixed value for the version number. When you force a version number like this it will not be possible to operate with parallel versions of the message.

Interface versioning

When operations of a Web service are changed, it might be necessary to change the WSDL which describes the service as well. As for messages, not all changes breaks communication with consumers. Such as adding new operations or additional interfaces to existing operations work fine, as it supports backward compatibility. Also new data structures and data types can be added as long as they are added at the end of the request and made optional.

Changes the interface of a Web service may have impact on consumers in several ways.

Keeping the old service running will allow users to not relate to the new one. This means more maintenance work and requires more resources.

An exception could be sent back to the consumers to inform that the service is updated and where a description of the new service could be found.

By using an intermediary such as a broker requests to discontinued services could be routed to the new version.

The W3C recommends using namespaces with date stamps for versioning XML documents, and thus also WSDL documents, an example of this is shown in 6.5

```
http://www.example.com/2007/02/description.xsd
```

Listing 6.5: W3C XML namespace versioning

It is important to remember that even though the interface of the service does not change, some changes need a new version. One example can be if you change the semantics of a Web service, but there is no need to change the interface. Communication would still be running as before, but the data would be used in a different way then intended by consumers.

Chapter 7

Performance engineering

The main issues of performance engineering are how to guarantee that a system will meet its performance goals. The metrics the end-users of a system are mainly interested in are related to the Quality of Service (QoS) of a system. Weyuker and Vokolos claim [30] that usually, the primary problems that projects report after first release are not system crashes or incorrect systems responses, but rather system performance degradation or problems handling required system throughput. They also say that if queried, the fact is often that although the software system has gone through extensive functionality testing, it was never really tested to assess its expected performance.

To understand and predict performance issues of a system, there are two central models that can be used. Workload models are used to mimic real world workload in study, or used as input data for an analytical model of a system. Performance models are used to understand the problems in an IT system. These models are central in the quantitative analysis cycle which serves as a guideline for how to evaluate the performance of a system under different scenarios.

7.1 Quality of Service

Many computer systems have very high requirements for performance related attributes. For example, some system have high demands for very short response time, other business area may loose customers to other companies if their online shop is not open 24h/day, while other system may have very high requirements for safety. Menasce et al. [21] have discussed some important (QoS) attributes related to performance of a system:

Response time. The time it takes for a system to react to a human request, most often measured in seconds. The response time can in most cases be broken into several components. An example of response time is the time it takes for a web browser to display the content of a page after the user has requested it. In this case the response time can be divided in the time the browser use to send the request, the time it takes for the messages to be sent over the network and the time the web server uses to handle the request and send the response message.

Throughput. The rate at which requests are completed from a computer system, measured as operations per unit of time. Metrics for throughput vary for system to system, for example transactions per second for OLTP systems, HTTP request/second for web servers, packets per second for router and millions of instructions per second for a CPU. The throughput is a function of the load offered to a system and of the maximum capacity of a system to process the work. For some systems, the throughput can decrease if the overall load is higher than a certain level. This phenomenon is caused because the system uses more resources than it can handle, and therefore must use resources to handle faults instead of productive work. Typical this can happen if the system uses more memory than it has physically installed, and have to use the disk for virtual memory. Because the access time for a hard disk is much slower than the memory the overall system performance will be much slower. This phenomenon is called thrashing.

Availability. The fraction of time a system is up and available for its customers is the definition of availability. This is measured in percentage. For example a system which is available in 99,99% of a time period of thirty day is unavailable 4,32 minutes in that time period. If the given availability is good or bad depends of the context the system is in, for example a internet shop which has 99,99% availability is very good, but can be fatal for an emergency or defense system. The two main reasons why a system can become unavailable is failure and overload.

Reliability. This attribute is defined as the probability the system functions properly and continuously over a fixed period of time. This attribute is closely related to availability.

Security. This attribute can be split in three parts: *Confidentiality* is about that only authenticated users get access to the information. *Data integrity* is about data can not be modified by unauthorized users. *Non-repudiation* is about senders of a message can not deny having it sent.

Scalability. This attribute is about if the performance of the system is not affected significantly as the load to the system is increased.

Extensibility. This is the property of the system to easily evolve to cope with new functional and performance requirements.

Not all of these QoSs are directly performance measures, but they are affecting the overall performance of the system. For example, if a system have high demands of security, it is likely that this will affect the throughput and response time the system negatively.

7.2 Method for analyzing performance of computer system

Performance engineering is used to be able to evaluate and predict performance of a system in a systematic way. Menasce et al. [21] present a method to analyze the performance of a system, as shown in figure 7.1. There are some basic steps to follow in the method. First, the system performance objectives should be specified as a part of the systems requirements. After the system performance objectives are defined, *the quantitative analysis cycle* can be followed. Steps 1-3 are used to evaluate the performance of the system, while steps 4-8 are used to be able to predict performance.

1. Understanding the system. This step involves getting an in-depth understanding of the systems architecture with focus of performance issues. A systematic description of the architecture, its components and goal are typically needed in this step. This is a good opportunity to review the performance issues related to the proposed architecture.

2. Characterize the workload. The goal of this step is to identify the basic components which compose the workload of the system. The workload of a system is the set of all inputs that the system receives from its environment during any given period of time.

3. Measure the system and obtain workload parameters. This step involves measuring the performance of the system and obtain values for the parameters of the workload model.

4. Develop performance model. Quantitative techniques and analytical models are used to develop performance models of the system. These models can be

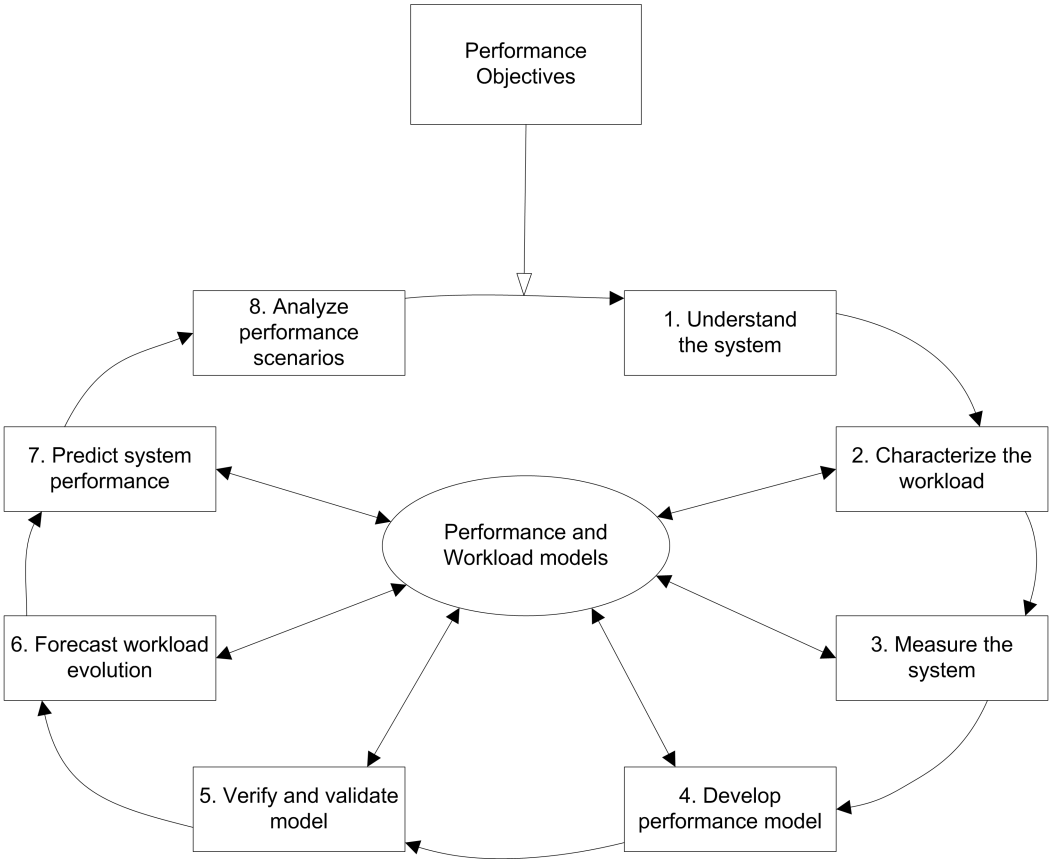

Figure 7.1: The quantitative analysis cycle

used to predict performance when any aspect of the workload or the system architecture is changed.

5. Verify and validate model. In this step, the model specifications and the model's results are validated. A performance model is validated if the performance metrics calculated by the model math the measurements of the actual system within a certain acceptable margin of error. This step answers the question: "Does the model capture the behavior of the critical components of the system".

6. Forecast workload evolution. This step forecasts the expected workload of the system. Demands grow or shrink over time, dependent of factors such as the functionalities offered to users, number of users, hardware upgrades etc.

7. Predict system performance. In this step, performance models are used to predict the performance of a system under different scenarios.

8. Analyze performance scenarios. In the last step, the performance of the system is predicted under different scenarios based on validated performance and workload models.

7.3 Workload Model

Step 2 in the quantitative analysis cycle described in section 7.2 is to characterize the workload. A workload model is a representation that mimics the real workload under study. This can be a program(s) written with the goal to artificially test a system in a controlled environment or to serve as input data for analytical models. The models should be compact and representative for the actual workload of the system. There are to categories of workload models.

Natural models are based on basic components of the real workload of the system or execution traces of the real workload. Programs extracted from the real workload of the system can be used as natural benchmark, representing the overall system load in given periods of time. Workload trace consists of a chronological sequence of data representing specific events that occurred in the system during a measurement session. Traces often consist of huge amounts of data, making it complex to use it in modeling.

Artificial models are based on special-purpose programs and descriptive parameters. These models can be grouped into executable artificial models and non-executable artificial models. Executable artificial models consist of a suite of

programs written to experiment with particular aspects of a computer system. This includes workloads such as instruction mixes, kernels, synthetic programs, artificial benchmarks and drivers. Non-executable workload models are described by a set of average parameter values that reproduce the same resource usage as the real workload, where the parameters denotes an aspect of the execution behavior of the basic component on the system under study. Examples of such parameters are component interarrival times, service demands, component sizes and execution mixes.

7.4 Performance models

A central component in the quantitative analysis cycle presented in section 7.2 are the performance models. Performance models include both visual and quantitative representations of a system. A model is an abstraction or generalized overview of a real system, and system models are central part in performance engineering [21]. There are many advantages of abstracting real system into a modeled representation:

- Several properties of a system can be elicited in the process of building a model.
- A model is a useful guide on what type of measurements to take and what kind of data to collect.
- A number of interesting metrics can be readily computed from the input parameters even before model is solved.
- A model can be used to answer what-if questions about a real system, avoiding costly and time-consuming experiments.

In [21] three different types of models are presented. Common for all three is that the model first is constructed, and then it gets solved. How a model is solved depends of the type of model.

Prototype model is a physical construction of a scaled version of the actual system it represent, and executes a typical workload representative for the system. The main advantage by this model is its accuracy, but it is expensive to develop. It is solved by running an experiment or tests while monitoring its performance.

Simulation model involves software programs which emulates the performance of the system. For example, a script from a typical workload can mimic the behavior of the actual system. Simulation models are less costly than prototype

model, but it tends to be less accurate. These models are solved by running a software package (simulation) and record the emulated performance results.

Analytical model involves capturing the key relationships between the architecture and the workload components in mathematical expressions. The advantage is that it captures and provides insight into the interdependencies between the various system components, is flexible, inexpensive and easily changed. The disadvantage is lack of detail and they tend to be difficult to validate. They are solved by solving a set of mathematical equations and interpreting the performance expressions correctly.

7.5 Visual representation of models

7.5.1 Conceptual Modeling

Conceptual modeling gives a good visual representation of a computer system. Robinson [25] uses the following definition of a conceptual model:

> The conceptual model is a non-software specific description of the model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplification of the model.

A conceptual model is in effect a functional specification of the computer software. The definition outlines the key elements of the model, which is:

Objectives. The purpose of the model and the modeling project.

Inputs. The elements of the model that can be altered to effect an improvement in, or better understanding of, the real world (experimental factors).

Outputs. The results from the model.

Content. The component that are represented in the model and their interconnections.

Assumptions. Either when there are uncertainties or beliefs about the real world being modeled.

Simplifications. By making the model simple, it is more rapid to develop and use.

It is useful to have requirements in mind when designing the model. Typically there are four important requirements to a conceptual model. *Validity* is, from the modeler's perspective, the question if the model is right, if the model represent the actually real world thing it is supposed to do. *Credibility* is the same as validity but in the perspective of the clients rather than the modeler. *Utility* is about the usefulness of the model. The conceptual model shall lead to a computer model that is useful as an aid to decision-making within the specified context. *Feasibility* is that the model can be developed into a computer model.

To be able to meet all the above requirements, the model should be kept as simple as possible to meet the objectives of the model. The advantages of simple models is that they are fast to develop, flexible, require little data, run fast and is easy to interpret the results since the structure of the model is easy to understand.

To represent the content of the conceptual models, there is four main methods in common use. By using more than one of these methods, different views of the same conceptual model is given.

Component list

This representing gives a list of the components of the system with a description of the detail of each component. This is a simple approach, which does not give a good visual representation and it is difficult to capture complex logic and the process flow of the system based on this. Table 7.1 is an example of a component list.

Component	Detail
Customers	Time between arrivals (distribution)
Queue	Capacity
Service desk	Service time (distribution)

Table 7.1: Example: Component list

Process flow diagram

This is an approach where the visual representation is useful for showing the process flow of the system. The model is represented as a process flow or process map which shows each component of the system in a sequence, including description of the model detail. Typically a process is shown as a box and a queue is shown as a circle. An example of a process flow diagram is shown in figure 7.2

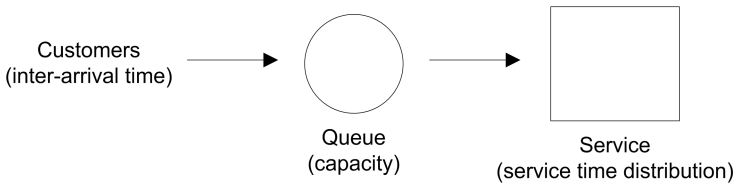

Figure 7.2: Example of Process Flow Diagram

Logic flow diagram

This model uses standard flow diagram symbols to represent the logic of the model. The disadvantage of this diagram is that the process is not always obvious and becomes quickly very large and complex. An example of a logic flow diagram is shown in figure 7.3.

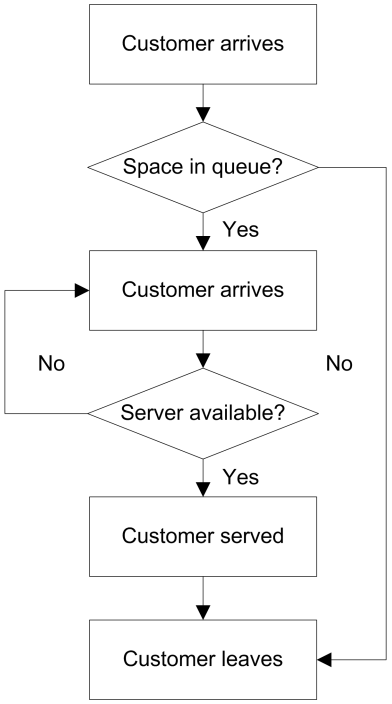

Figure 7.3: Example of Logic Flow Diagram

Activity cycle diagram

This diagram represents the logic of a model and gives a good visual representation. It provides a convenient means for identifying the events of the system, so the main

use is to act as a basis for programming models. This model quickly becomes very complex for big systems. In figure 7.4, active states are represented as rectangles and the circles represent dead states (the system is waiting for something to happen).

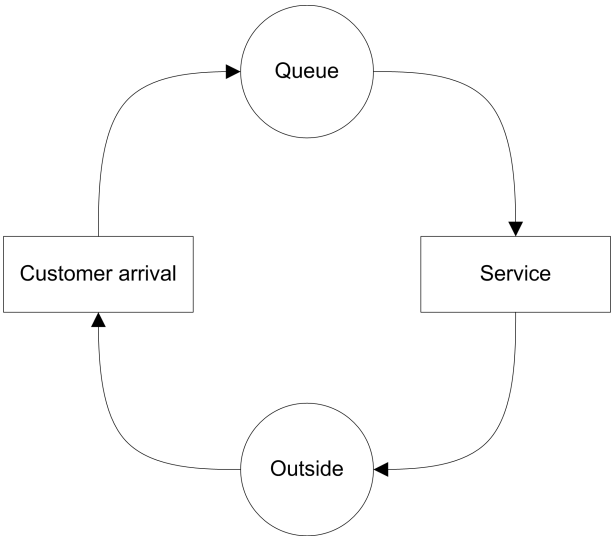

Figure 7.4: Example of Activity Cycle Diagram

7.5.2 Queuing Network models

Menasce et al. [21] use a framework called *queuing models of computer systems* for describing performance issues in computer systems. This framework is based on the observation that computer systems are composed of a collection of resources (processors, disk, communication links, database locks etc) which are shared by various requests (transactions, Web request, batch process etc). There are usually several requests made concurrently, wanting to access the same resource at the same time. Since the resources only can handle one request at a time, waiting lines can be build up in front of these resources. Basic queuing network models (QN) have been established as tools to evaluate and predict system performance. A formal definition is given:

"'A QN is a collection of K interconnected queues. A queue includes the waiting line and the resource that provides service to customers. Customers move from one queue to the other until they complete their execution and may be grouped into one or more customer classes. In some cases, customers may be allowed to switch to other classes when they move between queues. A QN can be open, closed, or mixed depending on its customer classes: open if all classes are open, closed if all classes

are closed, and mixed if some classes are open and some are closed."'

An open model has an external input and external destination. In a closed model the customers circulates without leaving the network. Figure 7.5 shows an example of a mixed model for a database server.

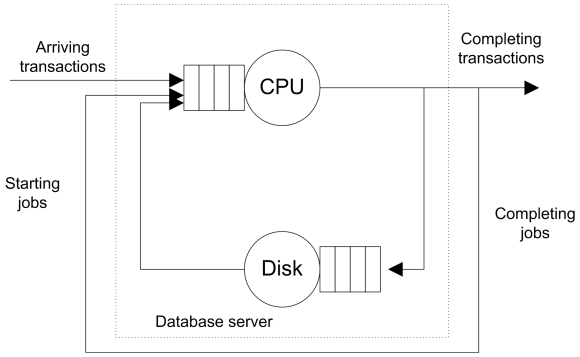

Figure 7.5: Mixed QN model.

7.6 Quantifying Performance Models

7.6.1 Performance evaluation metrics

These metrics are used when measuring a system, and can later be used to evaluate prototype and simulation models.

7.6.2 Operation laws

The following subsections discuss relationships (operational laws) between operational variables. This laws are useful because they are very simple, based on readily available measurement data and can be used to obtain helpful performance metrics [21].

Utilization Law

The utilization of a resource is defined as:

$U_i = \frac{B_i}{T} = \frac{B_i/C_i}{T/C_i}$

B_i/C_i is the average time the resource was busy for each completion for resource i. T/C_i is the inverse of the resource throughput X_i. This relation can be written as:

$U_i = S_i \times X_i$.

Service Demand Law

The service demand is the total average time spent by a typical request of a given type obtaining service from resource i, and is associated both with a resource and a set of requests using the resource. The relation is given by:

$D_i = \frac{U_i \times T}{C_0} = \frac{U_i}{C_0/T} = \frac{U_i}{X_0}$

Where U_i is the utilization of the resource, T is the length of time in the observation period, C_0 is total number of requests completed by the system in the observation period and X_0 is the system throughput.

The Forced Flow Law

The forced flow law relates the throughput to a resource, X_i, to the throughput to the system, X_0. The throughput of a resource (X_i) is equal to the average number of visits (V_i) made by a request to that resource multiplied by the system throughput (X_0):

$X_i = V_i \times X_0$

Little's Law

Little's law is:

The average number of customers in a black box equals departure rate from the box multiplied by the average time spent in the box. $N = X \times R$

The black box may contain a single resource like a CPU, or complex system like the Internet. As long as the customers are not created or destroyed the law holds.

Interactive response time law

Interactive response time law is a formula for the average response time in a system with multiple customers. This is used in for example systems with multiple clients accessing a shared server, where the clients is alternating between thinking and waiting for server response. The formula is given by:
$R = \frac{M}{X_0} - Z$

Where R is average response time, M is number of clients, X_0 is the system throughput and Z is the average think time.

7.6.3 Bounds on performance

The performance of a system is bound by its *bottleneck* resource. This is the resource in the system with highest utilization, or largest service demand. By considering the service demands only, the upper bound on throughput and the lower bounds on response time can be obtained without solving underlying models. Menasce et al. [21] present two relationships to identify bottlenecks. The *upper asymptotic bound on throughput under heavy load conditions* is given by:

$X_0 \leq \frac{1}{max\{D_i\}}$

The *upper asymptotic bound on throughput under light load conditions* is:

$X_0 = \leq \frac{N}{\sum_{i=1}^{K} D_i}$

By using these relationships, it is possible to predict how the new performance of the system will be after the bottleneck is upgraded. Upgrading the old bottleneck component will make another component of the system the new bottleneck, which then the systems performance is dependent on.

7.6.4 Mean Value Analysis

Typically QN models are build upon state space diagrams, and the steps involved in creating it are:

1. Construct the state diagram by identifying all possible states that the modeled system may find itself.

2. Identify the state connections (transitions).

3. Parameterize the model by specify the length of time spent in each state once it is entered.

Solving this model involves abstracting a set of linear balance equations from the state diagram and solving them for long term steady state probabilities of being in each system state. However, this can lead to having to solve a large number of linear equations. A technique called Mean Value Analysis (MVA) can be used instead of solving linear equations. MVA calculates the performance metrics directly for a given number of customers, knowing only the performance metrics when the number of customers is reduced by one. The algorithm for MVA is given by:

```
Initialize the average number of customers at each device i:
                $\overline{n}_i(0) = 0$
For each customer population n = 1,2,...N,
        calculate the average residence time for each device i:
                $R'_i(n) = V_i S_i[1 + \overline{n}_i(n+1)] = D_i[1 + \overline{n}_i(n-1)]$
        calculate the average overall system response time:
                $R_0(n) = \sum[V_i \times R_i(n)] = \sum R'_i(n)$
        calculate the overall system throughput:
                $X_0(n) = \frac{n}{R_0(n)}$
        calculate the throughput for each device i:
                $X_i(n) = V_i \times X_0(n)$
        calculate the utilization for each device i:
                $U_i(n) = S_i \times X_i(n)$
        calculate the average number of customers at each device i:
                $\overline{(n)}_i(n) = X_0(n) \times R'_i(n)$
```

The main advantage of MVA is that it scales to a high number of devices and a high number of customers, and the amount of computations required is negligible. MVA's impact on the field of analytical performance evaluation has been huge, and the majority of commercially viable performance evaluation packages owe their success to MVA [21].

7.7 Examples of performance studies of systems using Web services

In this section we will present studies related to performance of Web services. These papers show how case-studies of performance evaluations can be performed. We will mainly focus on the methods used in these studies, and see how the performance analysis cycle from section 7.2 has been used.

7.7.1 Measurement-based Performance Analysis

Datla and oseva-Popstojanova [8] focuse on measurement-based performance analysis of an e-commerce application which uses Web services components to execute business operations in the business logic layer. The study is focused on measuring and analyzing the server-side performance at two different levels; the software architectural level and the hardware resource level. The workload is generated with a session based workload generation tool. We link the method used in the study to the quantitative analysis (7.2) cycle, and divide the method as follow:

1. A description of the object is given. The architecture of the system and the implementation and deployment details is described.
2. The workload is described. In the study this is generated by a tool, representing workload which is close to actually user workload in the system. Two different workload profiles were made, ordering and browsing.
3. Two metrics was measured: The response time of software components and hardware resource usage. To measure response times of the components, J2EE application servers application log events was used. To get measures of hardware resource usage, Windows 2000 performance monitoring tools was used.
4. The measures were used to make a performance model. This is used to find bottlenecks in the system. Especially in heavy load conditions Web services components tends to be the bottlenecks in the system due to the overhead by processing the messages. The proposed solution to this is developing more effective XML parsers and better mechanism for encoding and decoding SOAP messages.

In summary, analyzing the performance of e-commerce applications at different levels provides insight about potential bottlenecks and enables system designers to improve performance in a cost effective manner. The adoption of new technologies will depend on the capability to assess and even more to provide guarantees for their QoS.

7.7.2 Evaluation and modeling of Web services performance

Chen et al. [6] present a study of Web services performance by evaluating the current implementations of Web services and comparing them with a number of alternative technologies. The paper introduces three research problems: a) the range of performance the current SOAP implementations can provide, b) the differences between the implementations and c) the impact of SOAP document and encoding styles.

A theory was made: "Alternative technologies to Web services may offer better performance because of reduced interoperability overheads". SOAP was therefore compared with MS DCOM, MS .NET Remoting and MS Message Queue. We link the method used to the quantitative analysis cycle in section 7.2.

Steps 1-3 - Understand the system and workload, and measure the system.
A test benchmark, designed tominimize the business effect on server side, was made. Latency was measured by calculating the average of latencies of running a RPC call over 15 minutes. Three different test scenarios were performed:

1. Benchmarking SOAP
This test compared four different SOAP implementations. The test gave two results:
-Not all SOAP implementations have the same performance, and the performance gaps between SOAP implementations tend to become larger as SOAP message sized increase.
-SOAP document/encoding style has a significant impact on performance. Document/Literal achieves the best performance for all SOAP.

2. SOAP Core Performance
This test compared SOAP code performance with non-SOAP technologies. The results:
-SOAP is a compelling protocol to its binary alternatives for small messages.
-HTTP 100 synchronization degrades SOAP performance for small messages significantly (nearly 100%), which can be improved by turning off HTTP wait or binding SOAP to TCP/IP.

3. SOAP scalability over WAN
This test examined SOAP scalability along message sizes over WAN.
-SOAP does not scale as well as its binary alternatives for message sizes, due to increased amounts of data transferred and synchronizations.
-SOAP document and encoding styles have an impact on SOAP scalability. Document/Literal presents notable better scalability than RPC/Encoding.
-SOAP implementations also present different scalability for message sizes.

Step 4 - Develop performance model.
Step 5-8 - Verify, predict and analyze.
After this test was done, a model of SOAP performance was created. This was based on the OSI reference model. The latency for request-response call over network is:

$$\text{Latency} = T_{msgProc} + T_{msgTran} + T_{sync} + T_{app}$$

Where:
$T_{msgProc}$ is the cost of processing the messages
$T_{msgTran}$ is the cost to transfer a specific amount of messages over network
T_{sync} represents the overhead of extra synchronization required by protocols
T_{app} is the time spent in business logic at application level.

Based on this initial model, a more complex model was made which later was simplified to:

$$Latency = \sum_{i=i}^{s}[n\tau + \frac{D}{L}] + \sum_{i=1}^{w}[n\tau + n\frac{M_0 W_i}{N} + \frac{D}{L}] + \sum_{i=1}^{w} 2(\alpha + \beta M_i)$$

After this model was made, it was validated with a set of separate tests with different technologies like socket ping and Utilization Law. The model was successfully validated by predicting performance for different technologies and different environments.

Chapter 8

Implementation details

To test the performance of Web services and the MIP as strategies for message based systems integration, we implemented client and server applications which use these different communication technologies. The applications were developed to be used in a test to answer the research questions presented in chapter 5. We named the applications to reflect which technology was used and which research question it was created for. The names are therefore like: "WS_Q1App", which indicates the applications using Web service technology, and is created for research question 1.

Simplicity and making the applications as equal as possible was an important design criteria. This was important to be able to get test results based on the performance of Web service and MIP rather than because of differences in the application implementations. The applications were also designed so it shold be easy to change the workload which was sent between them.

All the test applications send messages based on the same message scheme. We do this to contribute to the neutrality of our comparison. The messages are custom made for the purpose of testing. This means that they contain fields for information we might need, such as time stamps, in addition to a field for payload.

During implementation we have been in close dialog with the developers of MIP. Our use of the integration product has revealed several errors in their source code, and we have been supplied with new versions of MIP several times. The MIP applications is still in a beta-phase, but the interfaces for it are set. This means that changes made in the MIP most likely will not influence on the way we have used it.

	MIP_Q1App	WS_Q1App	MIP_Q2App	WS_Q2App
Research question	Q1	Q1	Q2	Q2
Publish-subscribe	X	X		
Request-response			X	X
One-to-one			X	X
One-to-many	X	X		

Table 8.1: An overview of the implemented applications.

8.1 Application implementation for Q1 - Server to server publish-subscribe

The scenario for these applications can be described as: "a change being done to a server is forwarded and propagated to one or more servers". In our implementation, we created two servers which use the publish-subscribe pattern to integrate. MIP_Q1App uses the event channels in MIP. WS_Q1App is based on the WS-Eventing protocol [10].

8.1.1 MIP_Q1App implementation

In this implementation we used MIP as the integration technology. As described in chapter 4, MIP has a mechanism for sending events through its event channels. For this solution we created two server applications and one client application to invoke one of the servers.

Our MIP integration uses one MIP router for each endpoint in the communication. This means that there are two routers in our setup, both configured as server routers. Before a router is started the applications and channels that are directly connected to a router must be set up. This involves specifying that a channel, for instance, should be an event channel and whether it is a publish or subscribe channel. When the routers are started they find other routers and exchange channel information with them on their own.

1: SendMessage(Message) In a client application a message is created and sent to the local server.

2: Send(Message) The server recognizes that the message should be passed on and sent on an event channel. In the MIP adapter the send method is called and the messages are sent to the local router. After the local router has distributed the message to the other routers that subscribe to that kind of

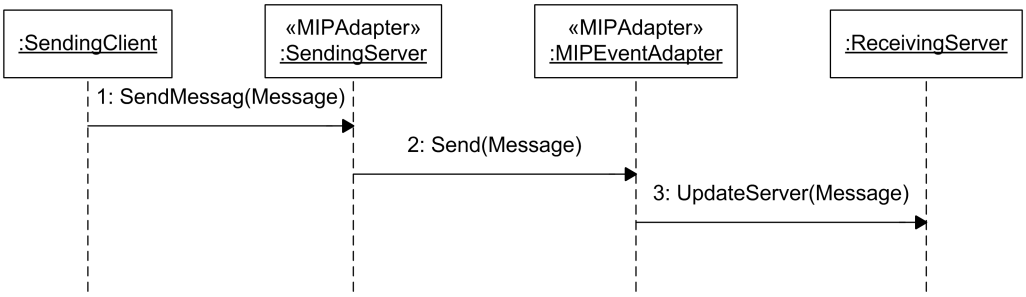

Figure 8.1: Scenario 1 - MIP sequence diagram

messages, in our case this is just one, the second router sends that message to the subscribing applications that are connected.

3: UpdateServer(Message) In our test implementation there is only one application listening for messages. The MIPEventAdapter is responsible for handling the incoming messages. These are in turn sent to the receiving server for final processing.

The routers and message sending are totally transparent for the classes that implement MIP adapters. All they interact with are send-methods on the channels and ordinary event mechanisms to receive messages.

8.1.2 WS_Q1App implementation

This implementation is based on the WS-Eventing for WCF project [18] by Roman Kiss. Design, implementation and usage of the WS-Eventing for distributed applications driven by Microsoft's Windows Communication Foundation is described in the article. The main components of the WS-Eventing implementation are the subscription manager and the notification manager. The first handles the operations defined by the WS-Eventing specification for runtime administration of the subscriptions, and keeps track of the active subscriptions. The latter is responsible for notifying subscribers when a new event is produced.

WS-Eventing specification

WS-Eventing is a W3C member submission from a group of members in Microsoft, IBM and TIBCO Software among others. A member submission is not a standard nor something the W3C is working on, it only describes a proposed technology submitted by W3C members. The WS-Eventing specification describes a protocol that

allows a Web service (subscriber) to register interest (subscription) with another Web service (event source) in receiving notifications (event messages) about events. The event source may also let another Web service handle subscriptions, a subscription manager.

The specification defines five operations necessary to enable eventing for Web services.

Subscribe sends a message containing information about how to reach the receiver of event messages, what kind of messages it is interested in and the duration of the subscription.

Renew sends a message requesting an extension of the duration of the subscription.

GetStatus gets the time of expiry for a subscription.

Unsubscribe sends a message from the subscriber that cancels the subscription.

Subscribtion end is sent from the event source to the subscription manager if the source terminates.

For us the subscribe operation is the most important as it is the only one necessary to be able to send a message. In listing 8.1 [10] a complete subscription message in SOAP format is shown. What is specific to WS-Eventing is found in the body part of the message, and prefixed *wse:*.

EndTo specifies that address of the Web service that handles messages that informs that the subscription has been terminated by the event source.

Delivery holds information about how to contact the Web service that wants to receive event messages.

Expires holds the time when the subscription should expire.

Filter lets you specify what topic the messages the subscriber is interested in should have.

```
<s12:Envelope
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
  xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
  xmlns:ew="http://www.example.com/warnings" >

<s12:Header>
  <wsa:Action>
```

```
    http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe
  </wsa:Action>
  <wsa:MessageID>
    uuid:e1886c5c-5e86-48d1-8c77-fc1c28d47180
  </wsa:MessageID>
  <wsa:ReplyTo>
    <wsa:Address>http://www.example.com/MyEvEntsink</wsa:Address>
    <wsa:ReferenceProperties>
      <ew:MySubscription>2597</ew:MySubscription>
    </wsa:ReferenceProperties>
  </wsa:ReplyTo>
  <wsa:To>http://www.example.org/oceanwatch/EventSource</wsa:To>
</s12:Header>

<s12:Body>
  <wse:Subscribe>
    <wse:EndTo>
      <wsa:Address>
        http://www.example.com/MyEventSink
    </wsa:Address>
    <wsa:ReferenceProperties>
      <ew:MySubscription>2597</ew:MySubscription>
    </wsa:ReferenceProperties>
    </wse:EndTo>
    <wse:Delivery>
      <wse:NotifyTo>
        <wsa:Address>
          http://www.other.example.com/OnStormWarning
        </wsa:Address>
        <wsa:ReferenceProperties>
          <ew:MySubscription>2597</ew:MySubscription>
        </wsa:ReferenceProperties>
      </wse:NotifyTo>
    </wse:Delivery>
    <wse:Expires>2004-06-26T21:07:00.000-08:00</wse:Expires>
    <wse:Filter xmlns:ow="http://www.example.org/oceanwatch"
     Dialect="http://www.example.org/topicFilter" >
      weather.storms
    </wse:Filter>
  </wse:Subscribe>
</s12:Body>

</s12:Envelope>
```

Listing 8.1: WS-Eventing subscribtion example

WS-Eventing implementation

We have used the WS-Eventing implementation by Roman Kiss with only minor changes. The original implementation uses a TCP binding for communication. We have changed this to a Web service HTTP binding to ensure interoperability. In addition to modifying the WS-Eventing implementation we created an event source application and a server application which can subscribe to and consumes events.

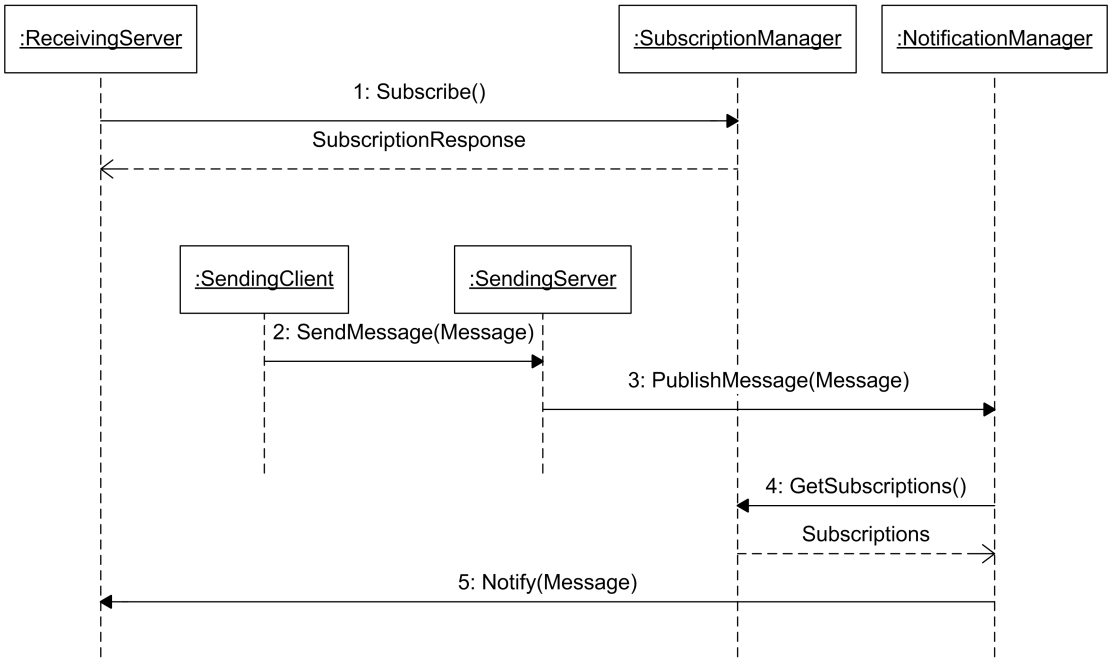

Figure 8.2: Scenario 1 - Web services sequence diagram

In figure 8.2 we see the flow of communication among the parties in our implementation.

1: Subscribe() The server contacts the subscription manager to subscribe to messages. The subscription is topic based and may also additional contain filters. In our implementation we do not use any filters. The duration of the subscription is also specified. A response message is returned confirming the subscription.

2: SendMessage(Message) As described by the scenario there are two servers that are the key communicating parties. To initiate the event source, the sending server, we have a client that creates a message and passes it on to its local server.

3: PublishMessage(Message) As the message from the client reaches the server, the server sees that this message contains information useful for other servers as well and publishes the message by contacting the notification manager.

4: GetSubscriptions() When the notification service is invoked the notification manager contacts the subscription manager to get the subscriptions for message with the topic of the one that just came in. A collection of subscriptions is returned.

5: Notify(Message) The notification manager then contacts all the subscribers of the certain message type by using the information given in the subscriptions.

Compared to the MIP way of communicating this solution is more opaque since you explicitly call the subscription manager to administer subscriptions and the notification manager when a message should be sent to the subscribers.

8.2 Application implementation for Q2 - data request

Cases needing these kinds of integrations are typically programs which has a real-time demands where a user waits for a response. Client applications typically request a service from a server application and waits for the response.

As the sequence diagrams in figure 8.3 and 8.4 show the two systems look very similar when it comes to message flow. This is because we in these diagrams treat the communication, MIP and WCF respectively, as black boxes. That is, implementers of these solutions do not know how the communication takes place.

8.2.1 MIP_Q2App implementation

Due to the nature of the MIP the implementation of this MIP_Q2App is quite similar to MIP_Q1. From an implementation perspective the main difference is the channels it uses. We use the command channels in this implementation. Figure 8.3 show the message flow in this implementation.

1: GetMessage(Request) The client creates a message request that contains information about what message it wants back. This request is sent to the MIP messaging proxy that handles the MIP related issues.

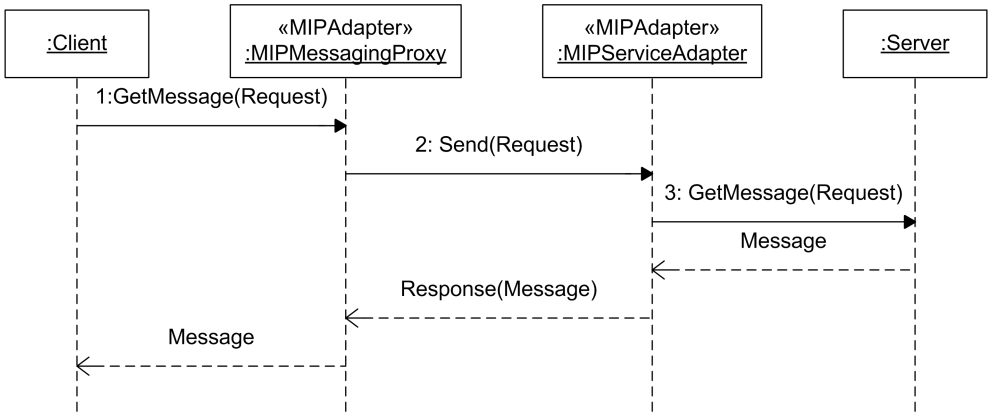

Figure 8.3: Scenario 2 - MIP sequence diagram

2: Send(Request) The messaging proxy sets up a connection on the right channel and sends the message to its local router.

3: GetMessage(Request) At the server side the command channel event handler in the MIP service adapter receives the request and passes it on to the server.

Response message The server finds the requested date and the service adapter encapsulates it in a response message and returns it to the client side.

8.2.2 WS_Q2App

The Web service implementation for Q2 uses Microsoft's Windows Communication Foundation (WCF), which makes it easy to develop Web service. WCF handles all serialization and deserialization of messages in addition to encapsulate a lot of the communication details. Figure 8.4 shows the message flow of this implementation.

1: GetMessage(Request) The client creates a request and passes it on to the message service client.

2: GetMessage(Request) The message service client is a proxy class for the Web service that handles the message requests, and thus the entry point into WCF.

3: GetMessage(Request) After the requests enters WCF it is routed to the correct service, in this case the messaging service. From here the server is contacted to obtain the requested message.

Response message The server returns the message to the messaging service, which returns it over WCF as from a normal method call back to the client.

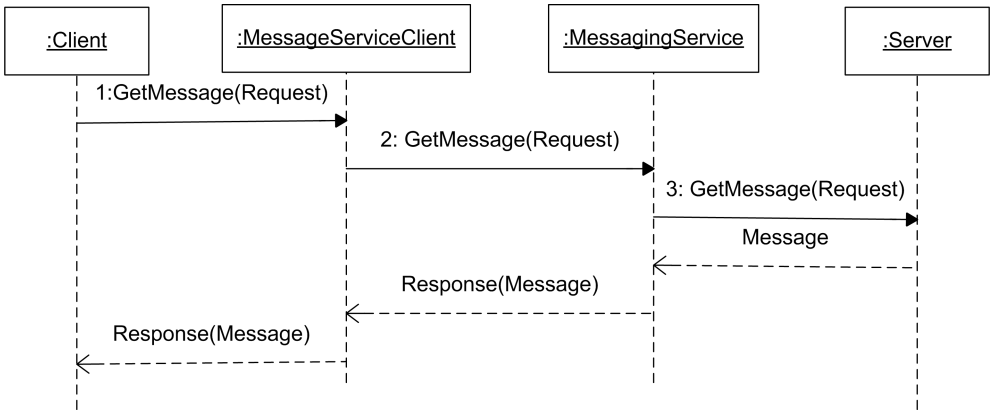

Figure 8.4: Scenario 2 - Web service sequence diagram

Chapter 9

Results for research question 1

In this chapter we describe our effort to collect data to be able to answer research question 1 (see section 5.4). First we design a general test to mimic the server to server notification scenario, described in section 3.2. Then we implement and perform performance test based on our implemented integration applications, described in section 8.1. When performing the test, we partly follow the quantitative analysis cycle (section 7.2). Step 1 corresponds with "Test design", step 2 is the "workload model" section and step 3 is our "measurements". The results from the test are presented in section 9.2.

9.1 Performance test for Q1

Figure 9.1: Overview of the performance test for Q1

Figure 9.1 illustrates the test concept. The integration between the applications is based on the publish-subscribe pattern. The server application 1 offers a service which server application 2 subscribes to. When an event occurs in server application 1, an event message is sent to the subscribers of the service.

In the test, we have only one application subscribing to the service. We also run both applications on one computer. This excludes the time it takes for a message to travel over a network, but the time it takes to process each message is increased

since we process both creating/sending and receiving/consuming messages on a single computer. This means that we get measurements of the whole system rather on the single components.

The following measurements are taken in the test:

Throughput. For this test we define throughput as how many messages that are sent from server application 1 to server application 2 per second.

CPU usage. This is monitored manually by using the Windows Task Manager performance tool. We wish this to be 100% all time when performing the test. The reason for this is the way we designed the test, with excluding the network transfer time and running both publisher and subscriber on the same computer. This means that our measurements includes only the processing time, so to get reliable measurements we need to make sure that the CPU is the bottleneck factor in the test.

Memory usage. This is also monitored manually with the Windows Task Manager performance tool. Since it is much faster to use the physical memory rather than virtual memory (swapping to hard disk) it is important for the performance that the total memory usage never is over the total physical memory available.

We perform the test by sending 200 messages with different payload sizes. The server application 1 creates and sends the event messages as fast as possible. We therefore designed the sending server function with the algorithm shown in 9.1.

```
for(int i = 0; i < 200; i++)
{
        SendMessage( CreateNewMessage() );
}
```

Listing 9.1: Algorithm for sending event messages

We designed the test to also include the time it takes to create the event message. This is to include the time it takes to serialize the messages.

To see how the payload sizes influences the performance, we perform the test with different payload. We split the payloads in three categories:

- Small messages. Payload size: 1B - 40kB
- Medium messages. Payload size: 40kB - 500kB
- Large messages. Payload size: 0,5MB - 3,5MB

9.1.1 MIP_Q1App performance test

Test design

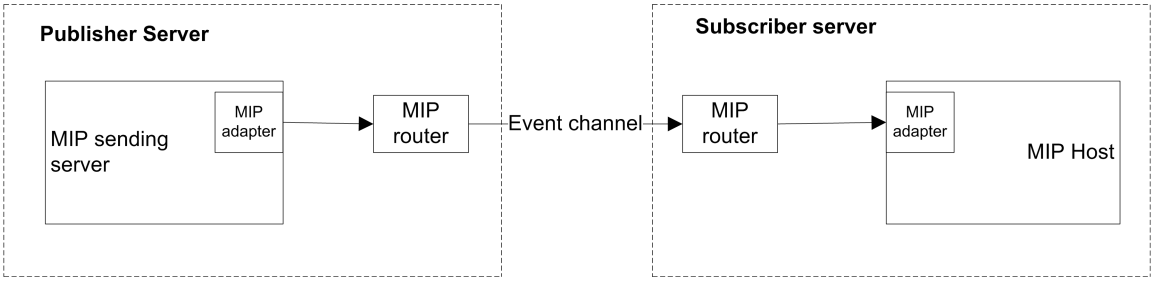

Figure 9.2: Test design: MIP_Q1App

Figure 9.2 illustrates the setup of this test. We use the event channel in MIP to realize the publish-subscribe pattern. When an event occurs in MIP Sending Server it is sent to the MIP router on the event channel via the adapter. The MIP router knows where to forward the message via the dynamic configurations, and sends the message to the Subscriber server MIP router. That router forwards the message to MIP Host which receives the messages via its adapter.

Since the test was performed on one computer, we had to manually configure two MIP routers to form a MIP network on one computer. Both routers are configured to run in server mode. We also configured the routers to turn off logging to screen and hard disk, which increased the performance significantly.

We originally planned to send 1.000 event messages from the MIP Sending server in the test. When we tried to send this many messages, the sending application crashed independently of the payload sizes. We did some testing and found out that the max number of messages we managed to send stable was around 200. We therefore did both this and the WS_Q1App test with only 200 messages. We reported this problem to the company, which created a new version of MIP which fixed this problem. This version also fixed some other performance-relevant issues. When we got the new version, we already had performed the test with the old version. We ran the test again with the new MIP routers. We present the results from both "MIP version 1" and "MIP version 2" in section 9.2.

Workload model

The workload model we used is constructed as a natural model to mimic a real world workload. MIP uses WCF DataContract when serializing the objects to XML

messages. The message that is send consist of one MessageContents object, which has four entities: String namespace, DateTime time, int messageid and String[] payload. The payload is split in 8kB strings so the number of strings in the array depends on the payload size.

Measurements

To measure the throughput of the system, we used the System.DateTime.Now.Ticks functionality in the .NET framework to get time stamps. First we made a time stamp right before the MIP Sending server started to create the first message. The second time stamp was made when the MIP host had received all the 200 messages. Then we used an application using the System.TimeSpan class to calculate the time used to send the messages based on the two timestamps.

When we ran the test, the Windows Performance monitor indicated that the memory usage was high, and dependent on the payload size. When sending large messages (we noticed this from 2MB payload) the memory usage was bigger than the physical memory available on the computer. This lead to swapping to the hard disk, and while this happened the CPU utilization went down. We present screen shots from the Windows Performance monitor in figure 9.7 with 2MB payload messages.

9.1.2 WS_Q1App performance test

Test design

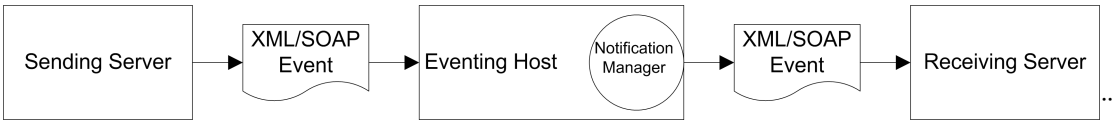

Figure 9.3: Test design: WS_Q1App

Figure 9.3 shows the main components in this test. The end points are the sending server and the receiving server. The Eventing host acts like a central distributer of messages. It consists of two main parts. *Subscription manager* handles all subscriptions to the services it provides, including end point addresses, bindings, subscription expire date etc. When an event is received, the eventing host uses this component to get the end points which subscribes to the service the event is from. The other part is the *notification manager.* This component has the responsibility to distribute the message to all the active subscribers to the service.

When we tested the original application, we had a problem with the notification manager class from the WS-Eventing project our application uses [18]. Our observation was that when the the eventing host received events, it often got an exception:

"Cannont access a disposed object.
Object name: 'System.Net.HttpListenerRequest'."

When this occurred, the subscription was deleted from the subscription manager, making our test corrupted.

We got problems with performing the test with this original version of the notification manager. First of all we often had to send the subscribe message to the Eventing host multiple times before it even worked. Another problem was that we never were able to successfully send 200 of the large messages, and big problem getting measurements with small messages. We therefore think that this solution was too unstable to use in our test. We solved the problem by modifying the original code in the "NotificationManagerService.cs" file. The original code is shown in listing 9.2.

```
ThreadPool.QueueUserWorkItem
(
        new WaitCallback(FireSubscriptionWorker), state
);
```

Listing 9.2: Original code in NotificationManagerService.cs

We modified this line to the code in listing 9.3.

```
FireSubscriptionWorker(state);
```

Listing 9.3: Modified code in NotificationManagerService.cs

This change made the applications work much more stable. We did perform a few tests with the original version to see if our modified version affected the performance significantly. We evaluate the influence the modified version has in section 11.3.

We also used two approached for how the messages were sent. The first version is based on opening and closing the connection between the Sending server and Eventing host for every message sent. The code for this is shown in listing 9.4. We call this version for "'WS_Q1AppIndividual"'.

```
public void SendMessage(MessageTransferRequest message)
{
    using (ChannelFactory<IMessageSending> channelNotification2 = new
    ChannelFactory<IMessageSending>("Notify"))
    {
```

```
            channelNotification2.Open();
            IMessageSending manager = channelNotification2.CreateChannel();
            manager.SendMessage(message);
            channelNotification2.Close ();
        }
}
```

Listing 9.4: SendingServer.cs. Code for WS_Q1AppIndividual: a new connection between Sending Server and Eventing Host is created for each message that is sent.

We observed that the performance of this version was very slow compared to the MIP solution (as shown in section 9.2). We therefore made the alternative sending version which opens the connection to the Eventing Host when the program is initialized and closes it when all messages are sent. The code for this is shown in listing 9.5. We call this version for "WS_Q1AppOpen".

```
public SendingServer()
{
    channelNotification = new ChannelFactory<IMessageSending>("Notify");
    channelNotification.Open();
}
public void SendMessageOpenConnection(MessageTransferRequest message)
{
    IMessageSending manager = channelNotification.CreateChannel();
    manager.SendMessage(message);
}
```

Listing 9.5: SendingServer.cs. Code for WS_Q1AppOpen: The connection between Sending Server and Eventing host is created the initialization of Sending Server.

The differences in WS_Q1AppIndividual and WS_Q1AppOpen only affect the sending mechanisms between the Sending server and Eventing Host.

Workload model

The workload model for this test is created as a natural model and is created in the similar way as the MIP workload. It is based on the .NET MessageContract.

An example of the SOAP message that is sent from sending server is shown in 9.6. The payload is the elements in <mc:MessageContents>. The payload consist of strings which together is on X bytes, split in 8kB long strings. In the example, the payload is on 100.000 bytes, which thus is split in 13 strings named <mc:Payload0..13>.

```
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing">
```

```
  <s:Header>
    <a:Action s:mustUnderstand="1">
    http://www.example.org/test/MessageSending</a:Action>
    <a:To s:mustUnderstand="1">
    http://localhost:33333/OnNewMessage</a:To>
  </s:Header>
  <s:Body xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <mc:MessageContents xmlns:mc="http://www.example.org/test">
      <mc:DateTime>2007-06-12T14:27:58.7232672Z</mc:DateTime>
      <mc:MessageId>1</mc:MessageId>
      <mc:ContentLength>13</mc:ContentLength>
      <mc:Payload0>aaaaa...aaaaa</mc:Payload0>
      <mc:Payload1>aaaaa...aaaaa</mc:Payload1>
         ..
      <mc:Payload12>aaaaa...aaaaa</mc:Payload12>
      </mc:MessageContents>
  </s:Body>
</s:Envelope>
```

Listing 9.6: WS eventing SOAP message

Measurements

We measured the throughput with the same method as with the MIP solution based on the System.DateTime.Now.Ticks and System.TimeSpan functions.

While performing the test, we manually monitored the CPU usage and memory usage. We observed that the CPU usage always was 100%, and the memory usage never was close to 100% of the physical available memory.

9.2 Results performance test Q1

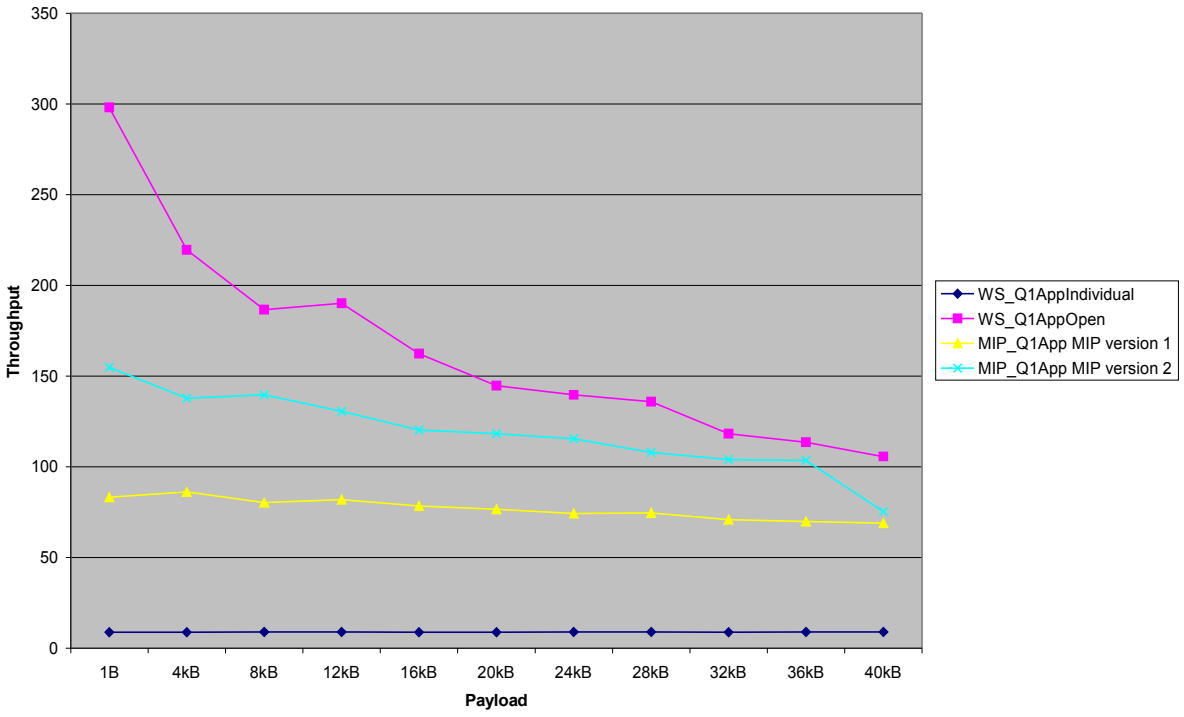

Figure 9.4: Throughput performance test Q1 with small messages

Figure 9.4 show the throughput of small messages. We see that the WS_Q1AppOpen has highest throughput. The MIP_Q1App version2 has second highest throughput, and the differences gets smaller with larger payload. With 1B payload WS_Q1App has 92% larger throughput than MIP_Q1AppOpen, and with 40kB messages the differences is 40%. We also notice that MIP version 2 is faster than MIP version 1 for all the measurements. The WS_Q1AppIndividual has lowest throughput, WS_Q1AppOpen has up to 33,9 times higher throughput.

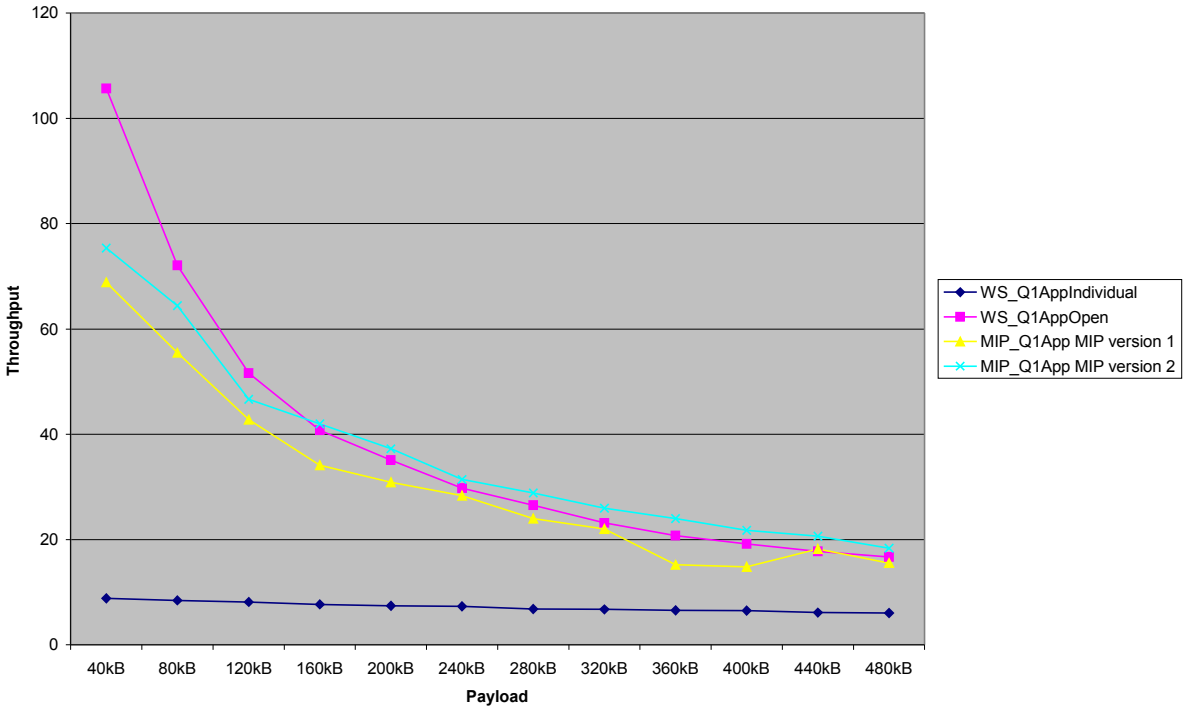

Figure 9.5: Throughput performance test Q1 with medium messages

Figure 9.5 show the throughput of medium messages. The trend from figure 9.4 continues with the differences in throughput between MIP_Q1App version 2 and WS_Q1AppOpen getting smaller with larger throughput. We see that the lines cross with about 160kB payload, so the MIP_Q1App version2 has highest throughput for the measures with larger payload than 160kB. We also notice that the MIP_Q1App version 2 is faster than version 1 in all measures. Also, as with small messages, the WS_Q1AppIndividual has very low throughput.

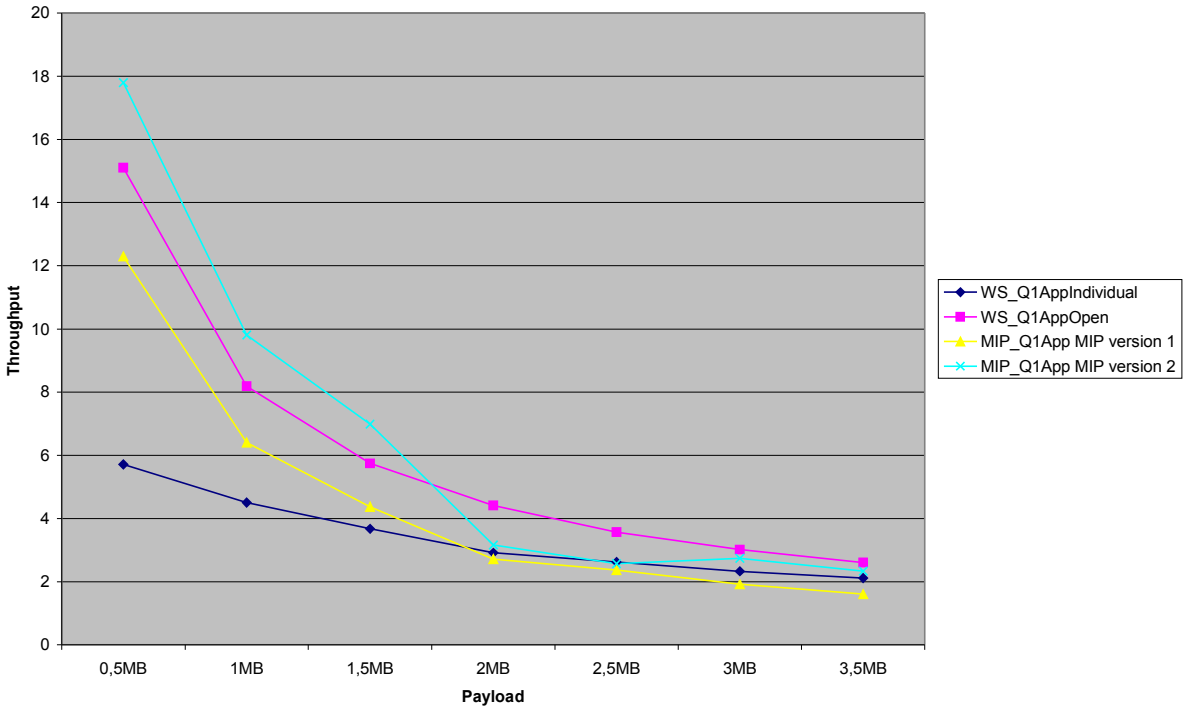

Figure 9.6: Throughput performance test Q1 with large messages

Figure 9.6 show the throughput of large messages. The trend from the previous measures continues until about 1,5MB payload size. For the 2MB payload size, we see that the MIP_Q1App (both versions) suddenly decreases throughput much compared with the WS_Q1Apps. Even the WS_Q1AppIndividual is faster than the MIP_Q1App version 2 with 2,5MB payload.

Figure 9.7 shows screenshot of the Windows Task Manager performance monitor while performing test with 2MB payload for WS_Q1AppOpen and MIP_Q1App version 2. The metrics of interest in this screenshot is the graphs which shows the CPU usage history (graph on the top) and the memory usage (graph on the bottom).

We see that the CPU usage is 100% during the test, and the memory usage is low in the WS applications. For the MIP applications, we see that the CPU usage decreases a lot during the test. We also see that the memory usage is very high, and

above the available physical memory.

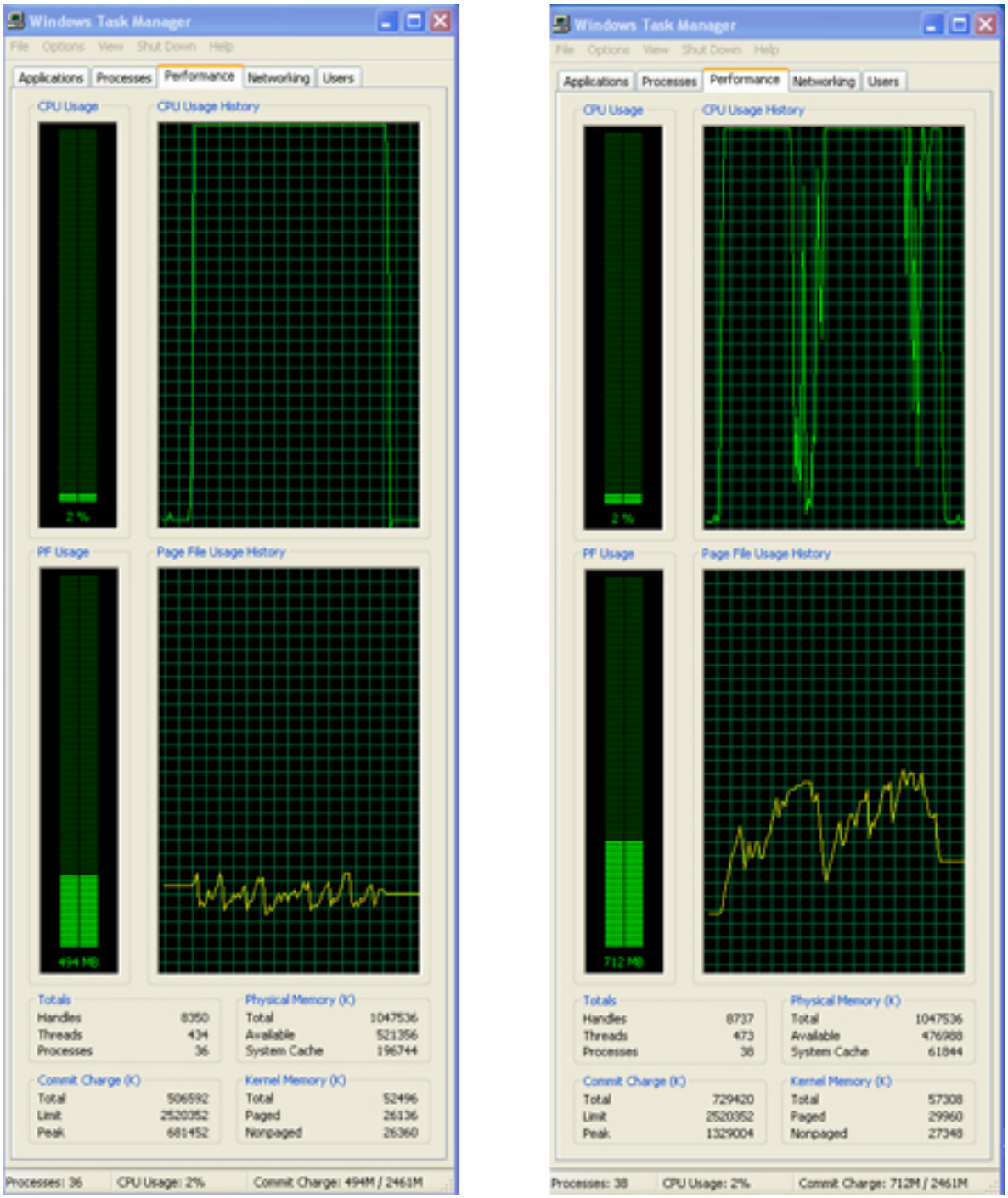

Figure 9.7: Screenshot of CPU and memory usage while performing test with 2MB payload. Left is the WS_Q1AppOpen test and right is the MIP_Q1App version 2 test.

Chapter 10

Results for research question 2

This chapter describes our effort to collect data for research question 2 (section 5.4), and is organized the same way as chapter 9 with first giving a general description of the performance test to be able to get measurements for the metrics to Q2. Then we give detailed descriptions of the test we performed which follows the quantitative analysis cycle from section 7.2 before we present the results from the test.

10.1 Performance test for Q2

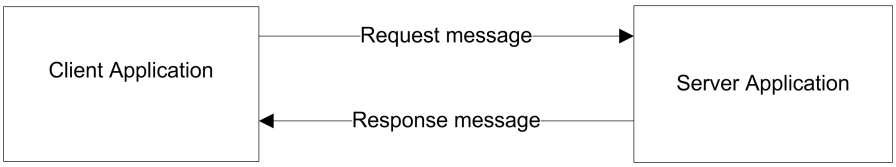

Figure 10.1: Overview of the performance test for Q2

Figure 10.1 illustrates the concept of this test. A client application sends a request message (requesting a service) to the server application, the server process this message and sends a response message back to the client. In the test we measure the following metrics:

Response time. This is how long it takes from the client starts to create the request message until it receives the response message from the server.

CPU usage. This is monitored manually by using the Windows Task Manager performance tool. Ideally this is 100% all time when performing the test. The reason for this is the same as with the CPU usage in the Q1 test.

Memory usage. This is also monitored manually with the Windows Task Manager performance tool.

We use a high performance timer class to be able to get exact measurements for response time in this test. This C# class is based on the project described in [27]. An alternative we first tried to use the Windows API function GetTickCount(). This function only gets resolutions on 1ms, which was too imprecise for our test.

To see how the sizes of the message response payload affect the response time, we performs the test with different payload sizes. We classify the payload sizes in three categories:

- Small messages. Payload size: 10B - 40kB
- Medium messages. Payload size: 40kB - 500kB
- Large messages. Payload size: 0,5MB - 3MB

In the test, the client and server applications run on the same computer to exclude the time it would take to send the message between the machines on a local area network or over the Internet. This excludes an uncertainly variable - the time it takes to transfer the messages over the network. We also had a restriction with only having one client sending requests to the service. This way we make sure that there does not build up a queue in front of the service, which would increased the response times. The results from this test are therefore the best-case scenario. When executing the performance test, 10.000 messages was sent synchronously, and we measured the response time for each request.

10.1.1 MIP_Q2App performance test

Test design

Figure 10.2 shows the test setup. MIP Client creates and sends a request message to the MIP Host via two MIP routers. The MIP Host processes the message and sends back a response message via the routers.

To be able to run this test one a single computer, we had to manually configure two MIP routers to form a MIP network on one computer. The router on the client side is running in client mode, and the router on the server side is running in server mode. We also turned off logging on the routers to improve the performance.

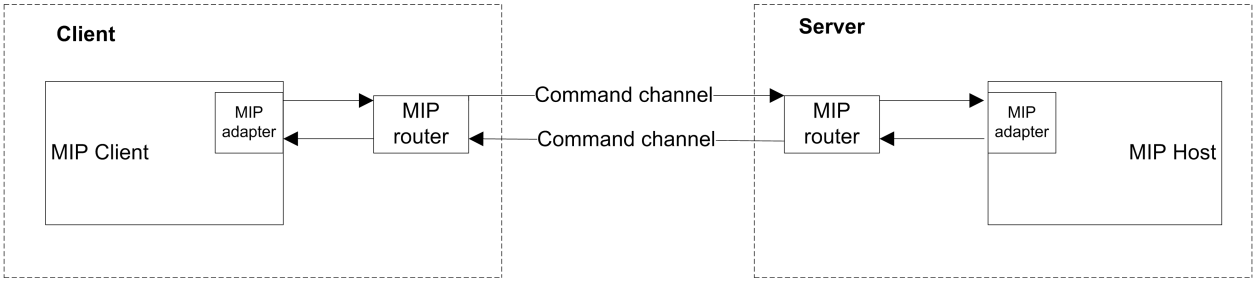

Figure 10.2: Test design: MIP_Q2App

Workload model

The workload model is created as a natural model, basically the same way as with scenario 1. There are two messages that are sent:

MIP Request message consist of one MessageContents with a int messageId. This int tells the server which message it requests. The MessageContent also has an string[], which is empty in the request message.

The MIP response message consist of a string status and a MessageContents. This is the same as with the request, but with the payload filled. The payloads consist of 8kB long strings, so the number of strings is dependent of the total size of the payload.

Measurement

When measuring the response time, we started the counter right before the request message was created in the client application. The response time thus includes the time it takes to create the request message, the time it takes to send the message to the MIP host via the two routers, the time the server uses to process the message and send back the response message in the same way. The response messages are initialized when the MIP Host starts.

During the test we monitored the CPU usage and memory usage. We observed that the CPU usage always was at 100% during the test, and the memory usage was never close to 100% of total physical memory.

10.1.2 WS_Q2App performance test

Test design

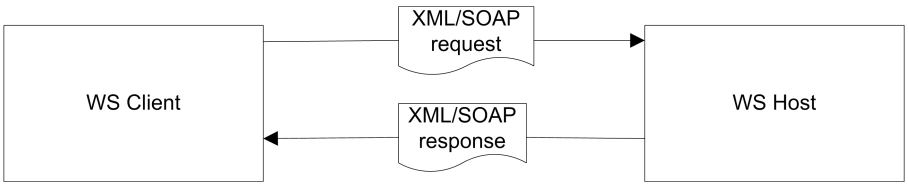

Figure 10.3: Test design: WS_Q2App

Figure 10.3 illustrates the design of the test. Basically the WS Client creates and sends a request message to the WS Host which sends back a pre created response message.

Workload model

The messages are created with the .NET WCF DataContract as a Web service/-SOAP message. In the same way as with the MIP request/response implementation, there is two messages:

The request message consist of a MessageContent entity with three attributes: String namespace, int messageid and a empty string[] payload.

The response message is similar as the request message but with the payload filled with 8kB long strings.

Measurements

When measuring the response time, we started the counter before the request message is created. The response time then includes the time it takes for the client to create the request XML document, the time it takes to send the document to the server, the time the server use to process the message and send back the response message to the client. We measured the response time with different payload sizes on the response message, and each test sent 10.000 requests to the service.

Also in this test we monitored the CPU usage and memory usage. The CPU usage was always at 100% and the memory usage was never close to 100% of physical available memory while performing the test.

10.2 Result performance test Q2

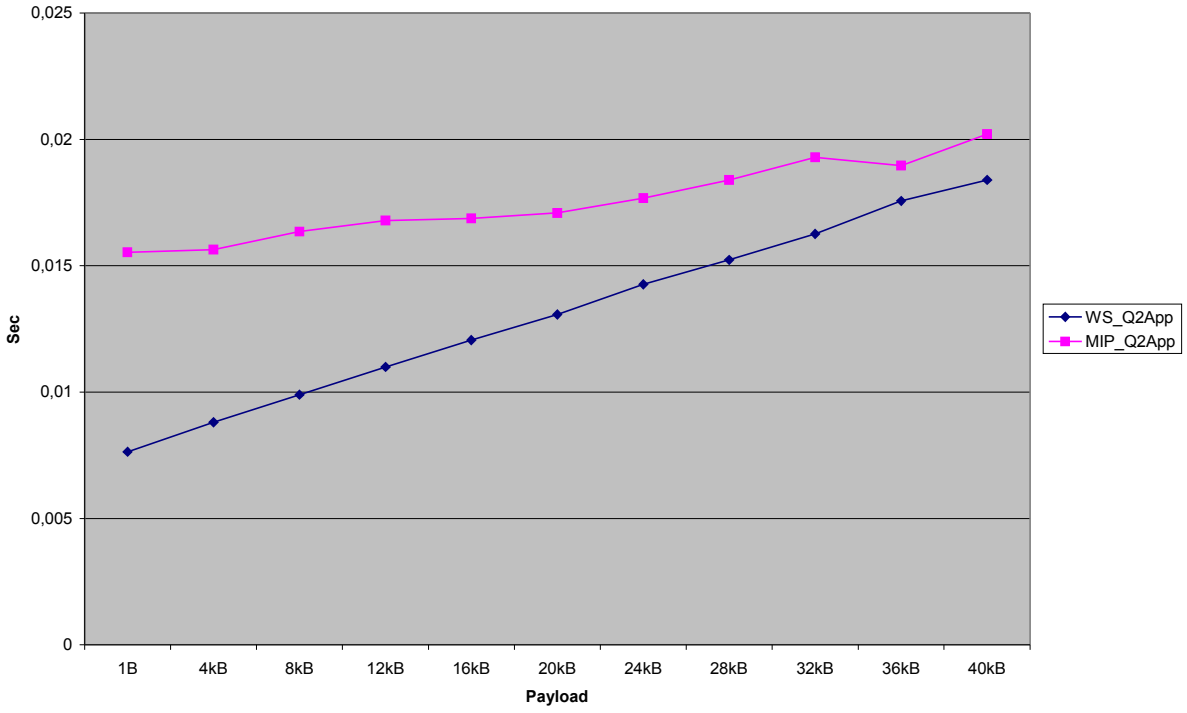

Figure 10.4: Average response time in the test with small messages

Figure 10.4 shows the average response time with small messages based on 10.000 measurements. We see that MIP_Q2App has about two times higher response time than WS_Q2App in the measure for 1B payload. As we see in the graph, the lines are very linear and the WS_Q2App has higher gradient. For the largest payload measure we see that the differences has decreased so the MIP_Q2App have about 1,1 times higher response time.

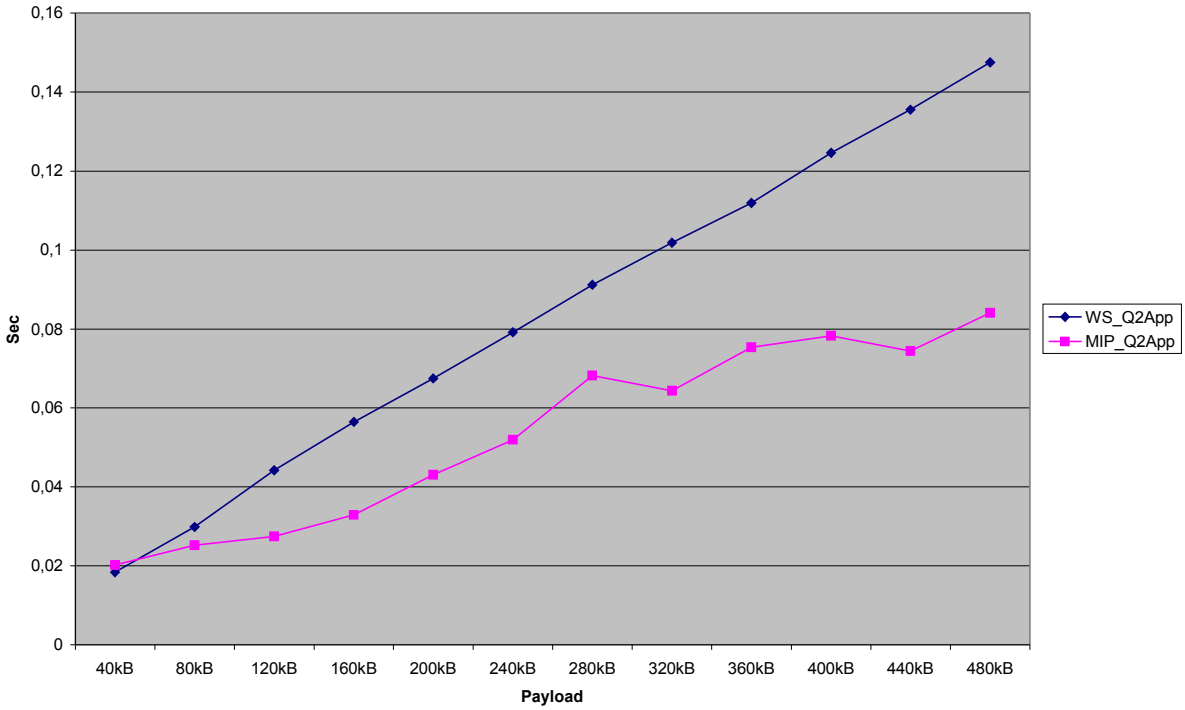

Figure 10.5: Average response time in the with medium messages

Figure 10.5 shows the average response time with medium messages based on 10.000 measurements. We see that the differences in response time are small with the smallest payload sizes, and that the lines crosses with about 45kB payload. The graphs shows that the measurements are very linear, with the WS_Q2App having largest gradient. With the largest payload the response time for WS_Q1App is 1,75 times the response time of MIP_Q2App.

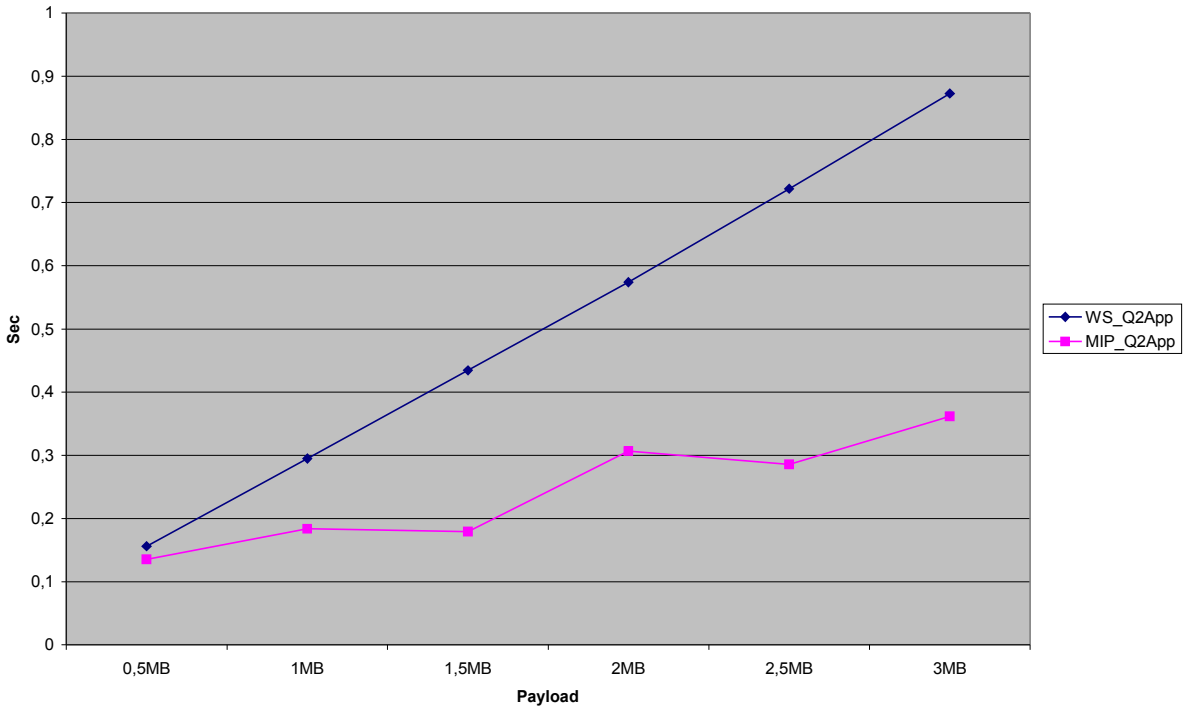

Figure 10.6: Average response time in the test with large messages

Figure 10.6 shows the average response time with large messages based on 10.000 measurements. We see that the trends from figure 10.4 and 10.5 continue by the increasing gap between WS_Q2App and MIP_Q2App. For the largest payload we see that WS_Q2App has 2,14 times as long response time as MIP_Q2App.

Chapter 11

Evaluation of the results

In this chapter we will discuss the results obtained in chapter 9 and 10. In the GQM framework, this is the same as answering the "'Q"' part. Following the performance analysis method from chapter 7.2, we will perform the forth step - create performance models. We will also discuss the research outcome, giving a more general recommendation of which of the two integration strategies we think is best for the company. Last in the chapter we will evaluate the validity of our performance tests.

11.1 Discussion of the results

11.1.1 Discussion Q1

Our first research question (Q1) was formulated as: *Which integration solution has best performance in a publish-subscribe scenario?* Our discussion is based on the performance test we performed, described in chapter 9.

In the test for Q1, we used two different versions of both the Web service and the MIP applications. For the Web service applications the difference was in the way the messages was sent to the Notification Manager. We see that the throughput for WS_Q1AppIndividual is much lower than WS_Q1AppOpen, and that the throughput is almost constant for small and medium sized messages. Based on this we see that our implementation of the sending mechanism is a main contribution to the low throughput for WS_Q1AppIndividual. The results show that WS_Q1AppOpen is competitive to the MIP applications, while WS_Q1AppIndividual has by far lower throughput. For the two MIP application test, the difference was use of two different version of MIP routers. The results show that there are significant differences, and version 2 gives best throughput. For further evaluation we will use WS_Q1AppOpen

(referred as WS_Q1App from now on) and MIP_Q1App with MIP version 2.

The results in section 9.2 showed that the WS_Q1App had higher throughput than MIP_Q1App for messages with small payload sizes. With messages with 160kB and higher payload, MIP_Q1App had highest throughput.

The MIP_Q1App has a high loss of throughput on the 2MB payload size. Our observations show that the CPU usage decreases while performing test with payloads equal to and larger than 2MB, and the memory usage is very high. We therefore believe the reason for the decrease in throughput is caused by depletion of available physical memory on the test computer. As the computer needs to start using other types of memory, this creates a bottleneck which affects the result.

To find out what caused the high memory usage in the MIP applications, we debugged the programs by making them simultaneously print the number of message sent and received. This showed that the MIP sending application sent all messages before the first message was received at the other end. The Web service application sent only a few messages before the first messages was received. This may explain the high memory usage on the MIP, as queues of messages is created in the MIP routers. Because of this, we do not think the throughput results for MIP with message size equal to and larger than 2MB are representative. This may be an unfortunate result caused by our test design with more than one router per computer and the resource contention beween them.

In the quantitative analysis cycle (section 7.2), step 4 is to develop performance models based on measurements. We found a formula for the throughput by using the least squares fitting technique:

$\mathrm{TP}_{WS\_Q1App}(\mathrm{P}) = \frac{1}{1{,}0492\times10^{-7}P+0{,}0065}$
$\mathrm{TP}_{MIP\_Q1App}(\mathrm{P}) = \frac{1}{8{,}8426\times10^{-8}P+0{,}0081}$

, where P is bytes of payload in the message. The quality-to-fit parameter for these functions was 0,9992 and 0,9971 respectively, on a scale from 0,0-1,0 where 1,0 is best.

To answer the research question Q1, we see that the performance is dependent on the message size that is sent. The Web service applications has best throughput with messages with payload up to 160kB, while the MIP applications has higher throughput with larger messages with larger payload. The throughput can be given by the performance models we created.

11.1.2 Discussion Q2

Research question 2 (Q2) is defined as: *Which integration solution has best performance in a request-response scenario?* Our discussion for this research question is based on the performance test described in chapter 10.

The results from section 10.2 show that the MIP_Q2App has lower response time than WS_Q2App for messages with payload size smaller than 45kB, and the opposite for larger messages.

Looking at all three graphs of the measurements, we see that the measurements plots are very linear. We use this to create a quantitative performance model for these applications, serving as step 4 in the quantitative analysis cycle (section 7.2):

$\mathrm{T}_{WS\_Q2App}(\mathrm{P}) = 2,72673 \times 10^{-7}P + 0,009447$
$\mathrm{T}_{MIP\_Q2App}(\mathrm{P}) = 1,13314 \times 10^{-7}P + 0,022010638$

, where P is bytes in the message sent. The correlation coefficient was 0,999 for the Web service formula and 0,980 for the MIP formula.

To sum up; we see that the Web service applications has lower response time than the MIP application with messages less than about 45kB. The MIP application is fastest with messages larger than 45kB.

11.2 Integration strategy recommendation

As stated in section 5.6, we will make a recommendation for which of the two strategies we think is best suited. In this section we evaluate the solutions in a broader perspective than for the two previous research questions. Even though performance is an important attribute, it is not the only one to base a choice of integration solution on. As listed in section 3.3 coupling, security and configuration should also be taken into account.

When we evaluate the performance of the two strategies we have used for our implementations, we can generally say that Web services is better for smaller sized messages. MIP and the Web services approach intersected at approximately 160 kB and 45 kB for Q1 and Q2 respectively. On questions about anticipated message size our industry partner has not been able to give us any good answers. We believe that these two scenarios may be somewhat different when it comes to message size. For Q1 we believe that message sizes typically will be in the small or medium seg-

ment, while they for Q2 more often may be in the upper end of the scale. When using the publish/subscribe communication pattern, messages will be sent when a change to an entity is made. This may generate many messages, but each of them will contain a limited amount of data. As a reference we may mention that a full page of text in A4 format contains about 4500 characters, which translates to 4,5 kB.

In the case of Q2 the message size is harder to predict. A message may be large or small depending of the request and the service processing it. However, in a service-oriented environment services should be business services, and thus have limited responsibilities. A service called *GetEntireDatabase* would typically not be a business service. We therefore believe that the message size also for this research question and scenario will be limited, but possibly larger than for Q1.

Since our main focus has been on performance we do not have any quantitative data for other quality attributes or issues important to our industry partner. Neither have our implementations made use of all the functionality of MIP nor our Web service solutions tried to replicate it. However, there are some factors that we are able to say something about.

Loose coupling. Web services are inherently loosely coupled. Clients can independently of operating system and programming environment in theory access a Web service. For clients to use MIP they are dependent on having a MIP router running locally and implementing a MIP adapter. If we look at MIP as a system, clients are quite loosely coupled to the routers. Routers can for example implement security without the clients knowing about it.

Security. For Web services using Windows communication foundation, security attributes can be achieved by setting configuration options for each service. For MIP security is administered by the routers, which is an advantage for the implementers.

Configuration. Without any infrastructure, Web services require a deal configuration and administration of service addresses and such. In our case the subscription manager does some of this job by being an intermediary between the senders and receivers. A UDDI service could also be helpful. MIP has solved the configuration issue by having routers share configuration among them, and also acts as a catalog of available services for applications.

All this taken into consideration, in addition to what we know about our cooperating software company, we claim that in their concrete situation the MIP is the better choice. Factors like that they have control over the applications that are the main targets for an integration solution, and these applications run on the same platform,

contribute to this statement. For instance, if they where making an infrastructure without knowing the applications that should be integrated, we think that choosing the solution with adapters using class libraries would be a bad idea. In their situation this works well.

If we look away from the factors in the concrete situation of our cooperator, we believe that a solution based on Web services is fully competitive for the scenarios we have investigated. This is also supported by our performance tests. The infrastructure provided by MIP, may also be realized for Web services. In addition Web services have the advantage of openness making integration with existing services an easier task. If the integration need can be expected to grow, as solution with Web services would put a lot less constraints on the new applications in the integration.

Even though the technology is in place, there is still a large piece of work to be done when it comes to using it. Having a technology based on messages in XML does not give you a service-oriented architecture. As explained in chapter 2, service-orientation is about aligning business with technology. That business services are created is important to take benefit of the available integration solutions. This will be one of the challenges the software company we have cooperated with will be faced with in future.

	Web services	**MIP**
Prerequisites	Implementation dependent.	Microsoft .NET run-time libraries
Coupling	Platform indepentent.	Services coupled with routers. Dependent on using the entire system.
Security	Specified for each service.	Handled by routers.
Performance	Better for smaller messages.	Better for larger messages.
Configuration	Has to be handled by clients or infrastructure, i.e. UDDI.	Handled by routers.
Run-time costs	Low.	Low.

Table 11.1: Summary of differences between Web services and MIP

11.3 Validity evaluation

An important issue concerning results from performance tests is if the results are valid. First of all the results should be valid within the context the tests are per-

formed. It might also be interesting to be able to generalize the results to a broader context. Wohlin et al. [31] classifies four types of validity schemes: conclusion, internal, construct and external validity. Below we discuss some threats within each of these schemes, and point out how we have taken this into consideration while performing the performance test or how they might present a possible threat to our results for Q1 and Q2. We first evaluate the validity of our results which is the same for both Q1 and Q1, and then we point out specific threats for each of the research questions.

11.3.1 Validity evaluation Q1 and Q2

Conclusion validity

Conclusion validity is concerned with issues which might affect the conclusions from the performance test and the performance test result. Threats to this might be errors in the way we implemented our applications and designed the tests. This kind of threats might influence the outcome (measurements).

- Reliability of the measurements. A threat may be if programs running in the background on the test computer may have affected the performance test results because they demands CPU time. We have tried to neutralize this threat by rebooting the computer before we started testing, and had as few processes as possible running in the background.

- The overhead of the messages that are sent is different because of the technologies used, and therefore the overall message sizes may be different. The payload we sent has been the same indifferent of the technologies that the messages have been created with.

Internal validity

Internal validity is concerned with that the relationship between the test and the result is not caused by a factor we do not have control over or a factor we have not measured.

- By running all tests on a single computer we have eliminated any uncertainty that could be introduced by using several computers and communication over a network.

Construct validity

Construct validity is concerned about the relationship between the theory (how the study is planned) and observation (how the research is executed). The study is based on a top-down GQM approach, as described in chapter 5. By first defining the goal with the study, and then defining research questions, the goal with the execution of the study has been clear before we started designing the performance tests. Also since we planned which metrics to measure before executing the tests, we are sure that the execution of the study reflects the planned study.

- A threat to our result may be the way we performed the MIP tests on one computer. The idea behind a MIP router is that it shall be one router per computer. In our test design, we configured two routers to be able to run on one computer.

External validity

External validity is concerned with generalization. Can the results be generalized to a larger context outside the scope of the study?

It is hard to generalize some of our results because the MIP source code is not publicly available. The Web service applications are implemented based on commonly used standards and technologies. We can therefore say that the Web service applications can be generalized to represent Web service based applications in general. The MIP can be seen as an example of a middleware integration application. However, there are very big differences between these kind of applications, so we can not say that this can be representative in general.

Even for our cooperator it may be hard to generalize the results from the MIP evaluation. MIP has not yet been released and during our project, we have had multiple versions of MIP. In our implementations and test, we have found several errors which have been fixed. Because of this, there is reason to believe that there might be done other changes to the software in the future. This might affect the performance of future versions of MIP. For example, we noticed significant differences in the throughput with two different versions in the test for Q1.

11.3.2 Validity evaluation Q1

Conclusion validity Q1

- Because of instability issues in one of the test programs, we were only able to send 200 messages in our test program. The number of messaged sent may be a threat to the validity of our results from the tests for Q1.

Internal validity Q1

- Modified NotificationManager. Because of stability issues for the WS_Q1App, we had to modify the code in NotificationManager, as described in section 9.1.2. We did perform tests to see if there were significant differences in performance with the original and modified code. To be sure that all the 200 messages were received in the unmodified version, we had to perform this test in debug mode in Microsoft Visual Studio. We were not able to get any results from messages with payload size above 64kb. Table 11.2 shows the result from this test. We see that there are not big differences in throughput between the two versions.

Payload	Stable	Unstable
4kB	172,2	167,8
8kB	153,6	158,5
16kB	137,7	141,6
62kB	108,0	114,8
64kB	74,0	78,6

Table 11.2: Throughput modified and unmodified NotificationManager.cs

- Memory usage. While performing the tests for Q1, we monitored the memory usage to be sure that the performance was not affected by too little physical memory available. While performing the MIP_Q1App test, we observed that the applications use very much memory. For message sized 2MB and higher we observed that the memory usage was above 100% of the physical available memory. This causes a threat to the results of large messages in this test since the memory usage for WS_Q1App never was close to 100%. This difference most likely affected our results for message sizes over 2MB.
- The CPU usage was monitored during the test. The usage was always 100% for WS_Q1App for all message sizes. We observed that the CPU usage went down for the test with message sized 2MB and larger in MIP_Q1App. This most likely affected our results for message sizes over 2MB.

Construct validity Q1

- Our test design for Q1 may have favored WS_Q1App when using large payload sizes (over 2MB). This is because the MIP solution, as pointed out before, uses a lot of memory. This may be because the MIP routers do not get enough CPU time or priority to process all the messages they get before the next message is delivered.

11.3.3 Validity evaluation Q2

Conclusion validity Q2

- Reliability of the measurements. For Q2, we used a freely available hi-performance timer class. A threat to the validity may if this timer did not give the reliable measurements we expected it did.

Chapter 12

Conclusion and further work

In this chapter we will summarize and conclude the study. We will also discuss possible ways to continue the work presented in this report.

12.1 Conclusion

Our goal with the study was to evaluate two integration strategies mainly with respect to performance. Based on the company's evaluations of strategies, we chose two strategies to evaluate. The first is created by the company, which in the future will serve as their integration infrastructure. The other strategy is based on Web service technologies. To evaluate, we created test applications which simulates two typical scenarios with integration of two applications each. We used this to perform performance evaluations to answer two research questions. We followed four steps in a method called quantitative analysis cycle in the performance evaluation.

Our performance evaluation results showed that the Web service test applications we implemented are faster than the MIP applications while sending small messages. With large messages the situation is the inverse. Based on the measurements, we created quantitative performance models for the applications which describe the throughput for the first applications, and the response time for the second.

Even though the MIP solution is more complex than the Web service approach we chose, our evaluation comes to that a Web service-based integration infrastructure will be competitive to this. However, for the homogenous integration environment of our cooperator, we believe that their solution is the better choice. This evaluation is based on the requirements from the company.

Our contributions of the study are mainly the recommendation for the company based on the performance evaluation and the general evaluation of the two strategies. We have provided data for throughput and response time for applications using MIP. We have also provided performance models of the MIP-based applications we have implemented, which can serve as a tool to be able to predict throughput and response time in similar integrations.

In a broader context, a contribution is the data collection for throughput and response time for applications using Web services. The comparisons of integrations based on Web services and MIP also serves as example of the performance of Web services versus other middleware. The performance models we have created may be useful to use to predict throughput and response time in similar integrations based on Web services.

12.2 Further work

The MIB has one main functionality which we have not tested with respect to performance. The reliable eventing functionality of MIP is a complex mechanism to guarantee that a message which is sent on the publish-subscribe channel is delivered. This is guaranteed even if the two involving applications never is "online" at similar times. A further work could be to implement similar functionality in a Web service eventing solution. It could be possible to implement such functionality in the eventing solution we used for our test application for Q1. We have provided an example of how such a solution could be implemented in appendix D.

As described in chapter 7, there are three types of performance models. Making analytical performance models of the performance of MIB and Web services could be an alternative way of performing a comparison case study.

For the performance evaluations we have performed, there are several other test which could be performed as supplement to our results. One way could be to try to perform evaluation in a more realistic setting. By implementing multiple clients (on multiple computers) with one publisher or one "response-server", it could be possible to get more reliable measurements for throughput. Also this would open up possibilities to get response times which include the delay with transferring the messages on a network.

Appendix A

Glossary

API	Application programming interface
BPM	Business process management
DCOM	Distributed component object model
COM	Component object model
CORBA	Common object request broker architecture
CRM	Customer relationship management
EAI	Enterprise application integration
ERP	Enterprise resource planning
ETL	Extraction, transformation and loading
GQM	Goal Question Metric
MIP	Message Integration Product
MIP_Q1App	MIP application for Q1
MIP_Q2App	MIP application for Q2
MOM	Message-oriented middleware
OLTP	Online transaction processing

QOS	Quality of service
RMI	Remote method invocation
RPC	Remote procedure call
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery, and Integration
W3C	World Wide Web consortium
WAN	Wide area network
WCF	Windows communication foundation
WS	Web services
WS_Q1App	Web service application for Q1
WS_Q2App	Web service application for Q2
WSDL	Web Services Description Language
XML	eXtensible Markup Language

Appendix B

Test computer technical specification

In all the performance test we used a computer with the following specifications:

Model	AOpen 1557GLS
CPU	Intel Pentium M 1,7GHz
Graphic Card	ATI Radeon Mobility 9700
Memory	1024 MB RAM
OS	Microsoft Windows XP SP2

Appendix C

Test results data

Bytes	**RTWS Q1App**	**RTMIP Q2App**	**TPWS Q1App Individual**	**TPWS Q1App Open**	**TPMIP Q1App-1**	**TPMIP Q1App-2**
1	0.0076	0.0155	8.8095	298.0782	83.2135	154.8158
4 096	0.0088	0.0156	8.8290	219.4642	86.0829	137.7327
8 192	0.0099	0.0164	8.8682	186.6471	80.2058	139.6590
12 288	0.0110	0.0168	8.8801	190.2023	81.8494	130.5310
16 384	0.0121	0.0169	8.8095	162.3678	78.3186	120.3087
20 480	0.0131	0.0171	8.8173	144.7191	76.5182	118.1730
24 576	0.0143	0.0177	8.8525	139.6590	74.2425	115.4407
28 672	0.0152	0.0184	8.8329	135.8588	74.5196	107.9527
32 768	0.0163	0.0193	8.8134	118.1730	70.8200	104.0169
36 864	0.0176	0.0190	8.8447	113.4730	69.8295	103.4779
40 960	0.0184	0.0202	8.8329	105.6679	68.8663	75.3632
81 920	0.0299	0.0253	8.4624	72.0983	55.4757	64.4234
122 880	0.0442	0.0275	8.1416	51.6053	42.7650	46.6618
163 840	0.0565	0.0329	7.7050	40.7576	34.1389	41.9564
204 800	0.0675	0.0431	7.4077	35.0988	30.8675	37.2598
245 760	0.0792	0.0519	7.3155	29.7634	28.3280	31.4013
286 720	0.0912	0.0682	6.8185	26.4871	23.9751	28.8602
327 680	0.1019	0.0643	6.7516	23.1416	21.9948	25.9704
368 640	0.1119	0.0754	6.5609	20.7601	15.2104	23.9751
409 600	0.1246	0.0783	6.5265	19.2031	14.8045	21.7315
450 560	0.1355	0.0745	6.1243	17.7364	18.2053	20.6314
491 520	0.1475	0.0608	6.0391	16.6984	15.5539	18.3559
524 288	0.1562	0.1351	5.7142	15.1068	12.2976	17.7997
1 048 576	0.2948	0.1837	4.5092	8.1883	6.4051	9.8091
1 572 864	0.4344	0.1793	3.6732	5.7422	4.3653	6.9854
2 097 152	0.5740	0.3066	2.9117	4.4135	2.7194	3.1625
2 621 440	0.7219	0.2856	2.6243	3.5650	2.3679	2.5710
3 145 728	0.8727	0.3617	2.3317	3.0150	1.9183	2.7441

Table C.1: Test result data

Appendix D

Web services reliable messaging

For implementing reliable messaging in Web services there are serveral approaches. Reliable messaging could be about ensuring that the messages reaches the destination in a given order, or having lost messages resent. This is explained in the WS-ReliableMessaging [11] specifictaion. However, this specification does not take larger problems into account, such as if a computer shuts down. In figure D.1 we have created a sketch of how a system using Web services and a persitent message store could work in a publish/subscribe scenario.

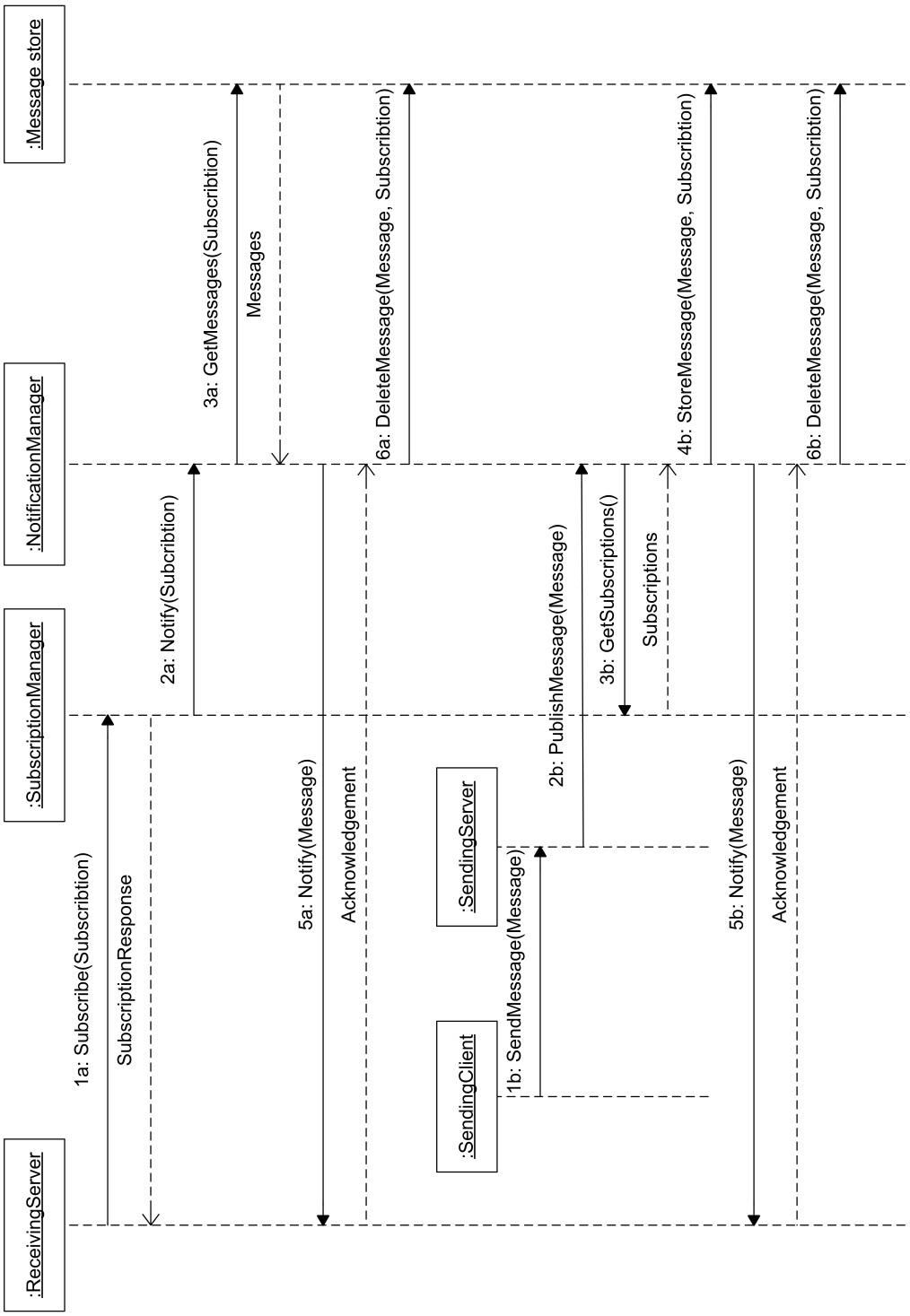

Figure D.1: Reliable messaging for Web services

Appendix E

Zip archive description

With the thesis, we provide a Zip archive with the source code for the Web services applications, and all the collected performance test data. The following is a copy of the ReadMe.txt file provided:

**********ReadMe.txt**********
This archive contains the source code for the Web service application described in the Master Thesis for Nils Torstein Øvstetun and Trond Smaavik.

Alle data.xls contains all the relevant data in a single spreadsheet.
Repons tid scenario 2.xls contains all collected data for the Q2 applications.
Througput scenario1_2.xls contains all data for the Q1 Applications

The src-folder is organized in the following subfolders:

WS_Q1App:
The source code for the eventing applications. To run the solution, first start the EventingHost project. Then RemoteServer and SendingClient is started, where number of messages and size can be set. For the sending application, output is in a textfield. For the receiving application, output is written to d:output.txt file.

To calculate time used to send the messages, the TicksToTime program can be used.

WS_Q2App:
This source code is for the applications used in the request-response applications. The WSHost project contains the code to the server application. The WSClient project contains the code to the client application. The message size, and number is set in the code. The output is written to the console.

WSEventing:
This source code contains the code for the eventing part of WS_Q1App. This is originally from http://www.codeproject.com/soap/WSEventing.asp. In this code, small modifications have been made.

TicksToTime: This is a small application used to measure the throughput for WS_Q1App.

Bibliography

[1] A model driven architecture for enterprise application integration. In *HICSS '06: Proceedings of the 39th Annual Hawaii International Conference on System Sciences*, page 181.3, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Elisa Bertino, Jason Crampton, and Federica Paci. Access control and authorization constraints for ws-bpel. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 275–284, Washington, DC, USA, 2006. IEEE Computer Society.

[3] Kyle Brown and Michael Ellis. Best practices for web services versioning, January 2004. http://www-128.ibm.com/developerworks/webservices/library/ws-version/.

[4] Luis Felipe Cabrera, Christopher Kurt, and Don Box. An introduction to the web services architecture and its specifications, october 2004. http://msdn2.microsoft.com/en-us/library/ms996441.aspx.

[5] Ethan Cerami. *Web Services Essentials (O'Reilly XML)*. O'Reilly, February 2002.

[6] Shiping Chen, Bo Yan, John Zic, Ren Liu, and Alex Ng. Evaluation and modeling of web services performance. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 437–444, Washington, DC, USA, 2006. IEEE Computer Society.

[7] F. McCabe M. Champion et al. D. Booth, H. Haas. Web services architecture, 2004. http://www.w3.org/TR/wsarch/#whatis.

[8] Venu Datla and Katerina Goseva-Popstojanova. Measurement-based performance analysis of e-commerce applications with web services components. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 305–314, Washington, DC, USA, 2005. IEEE Computer Society.

[9] N. Deason. Sip and soap. http://tools.ietf.org/id/draft-deason-sip-soap-00.txt.

[10] Craig Crithchley et al. Don Box, Luis Felipe Cabrera. Web services eventing (ws-eventing), March 2006. http://www.w3.org/Submission/WS-Eventing.

[11] Ruslan Bilorusets et al. Web services reliable messaging protocol (ws-reliablemessaging), February 2005. http://www.ibm.com/developerworks/library/specification/ws-rm/.

[12] John Evdemon. Principles of service design: Service versioning, August 2005. http://msdn2.microsoft.com/en-us/library/ms954726.aspx.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, January 1995.

[14] I. Gorton, D. Thurman, and J. Thomson. Next generation application integration challenges and new approaches, 2003.

[15] Venkat N. Gudivada and Jagadeesh Nandigam. Enterprise application integration using extensible web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 41–48, Washington, DC, USA, 2005. IEEE Computer Society.

[16] Ashok K. Harikumar, Roger Lee, Hae Sool Yang, Haeng-Kon Kim, and Byeongdo Kang. A model for application integration using web services. In *ICIS '05: Proceedings of the Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 468–475, Washington, DC, USA, 2005. IEEE Computer Society.

[17] Matjaz B. Juric, Bostjan Kezmah, Marjan Hericko, Ivan Rozman, and Ivan Vezocnik. Java rmi, rmi tunneling and web services comparison and performance analysis. *SIGPLAN Not.*, 39(5):58–65, 2004.

[18] Roman Kiss. Ws-eventing for wcf (indigo), June 2006. http://www.codeproject.com/soap/WSEventing.asp.

[19] Boris Lublinsky. Defining soa as an architectual style, January 2007. http://www-128.ibm.com/developerworks/webservices/library/ar-soastyle/.

[20] Kevin J. Ma and Radim Bartos. Performance impact of web service migration in embedded environments. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 409–416, Washington, DC, USA, 2005. IEEE Computer Society.

[21] Daniel A. Menasce, Lawrence W. Dowdy, and Virgilio A. F. Almeida. *Performance by Design: Computer Capacity Planning By Example*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[22] A. Raut, A.; Basavaraja. Enterprise business process integration. In *TENCON 2003. Conference on Convergent Technologies for Asia-Pacific Region*, volume 4, pages 1549–1553, 2003.

[23] Jaeho Ro, Sunhee Park, and Shim Yoon. Proposing web service-based business models by development phase. In *NWESP '05: Proceedings of the International Conference on Next Generation Web Services Practices*, page 431, Washington, DC, USA, 2005. IEEE Computer Society.

[24] Paul Kulchenko Robert Cunnings, Simon Fell. Smtp transport binding for soap 1.1. http://www.pocketsoap.com/specs/smtpbinding/.

[25] Stewart Robinson. *Simulation: The Practice of Model Development and Use*. John Wiley & Sons, 2004.

[26] Scott Seely. *SOAP Cross plattform Web serice development using XML*. Prentice Hall PTR, 2002.

[27] Daniel Strigl. High-performance timer in c#, August 2002. http://www.codeproject.com/csharp/highperformancetimercshar.asp.

[28] David Strommer. Best soa definition: Beat schwegler, June 2005. http://cs.jaxdug.com/blogs/davidstrommer/archive/2005/06/13/706.aspx.

[29] David Trowbridge, Ulirch Roxburgh, Gregor Hohpe, Dragos Manolescu, and E.G. Nadhan. *Integration Patterns*. Microsoft, 2004.

[30] F.I. Weyuker, E.J. Vokolos. Experience with performance testing of software systems: issues, an approach, and case study. *Software Engineering, IEEE Transactions on*, 26(12):1147–1156, 2000.

[31] Claes Wohlin, Per Runeson, Martin Höst, Maguns C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experrimentation in software engineering, An Introduction*. Kluwer Academic Publishers, 1 edition, 2000.