LEABHARLANN CHOLÁISTE NA TRÍONÓIDE, BAILE ÁTHA CLIATH | TRINITY COLLEGE LIBRARY DUBLIN
Ollscoil Átha Cliath | The University of Dublin

Terms and Conditions of Use of Digitised Theses from Trinity College Library Dublin

Copyright statement

All material supplied by Trinity College Library is protected by copyright (under the Copyright and Related Rights Act, 2000 as amended) and other relevant Intellectual Property Rights. By accessing and using a Digitised Thesis from Trinity College Library you acknowledge that all Intellectual Property Rights in any Works supplied are the sole and exclusive property of the copyright and/or other IPR holder. Specific copyright holders may not be explicitly identified. Use of materials from other sources within a thesis should not be construed as a claim over them.

A non-exclusive, non-transferable licence is hereby granted to those using or reproducing, in whole or in part, the material for valid purposes, providing the copyright owners are acknowledged using the normal conventions. Where specific permission to use material is required, this is identified and such permission must be sought from the copyright holder or agency cited.

Liability statement

By using a Digitised Thesis, I accept that Trinity College Dublin bears no legal responsibility for the accuracy, legality or comprehensiveness of materials contained within the thesis, and that Trinity College Dublin accepts no liability for indirect, consequential, or incidental, damages or losses arising from use of the thesis for whatever reason. Information located in a thesis may be subject to specific use constraints, details of which may not be explicitly described. It is the responsibility of potential and actual users to be aware of such constraints and to abide by them. By making use of material from a digitised thesis, you accept these copyright and disclaimer provisions. Where it is brought to the attention of Trinity College Library that there may be a breach of copyright or other restraint, it is the policy to withdraw or take down access to a thesis while the issue is being resolved.

Access Agreement

By using a Digitised Thesis from Trinity College Library you are bound by the following Terms & Conditions. Please read them carefully.

I have read and I understand the following statement: All material supplied via a Digitised Thesis from Trinity College Library is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of a thesis is not permitted, except that material may be duplicated by you for your research use or for educational purposes in electronic or print form providing the copyright owners are acknowledged using the normal conventions. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone. This copy has been supplied on the understanding that it is copyright material and that no quotation from the thesis may be published without proper acknowledgement.

TRINITY COLLEGE
21 NOV 2007
LIBRARY DUBLIN
THESIS
8254

Service Management in Dynamic e-Business Environments

A thesis submitted to the

University of Dublin, Trinity College

for the degree of

Doctor of Philosophy (Computer Science)

Marcus O'Connell

Department of Computer Science

Trinity College

Dublin

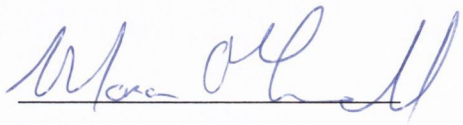
Ireland

October 2006

Declaration

The work described in this thesis is entirely that of the author and has not been submitted as an exercise for a degree at this or any other university.

Signed:



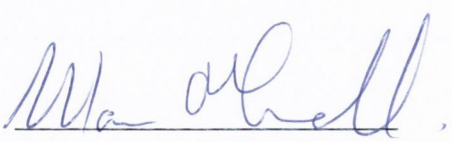
Marcus O'Connell

October 2006

Permission to Lend and/or Copy

I agree that Trinity College Library may lend or copy this thesis upon request.

Signed:



Marcus O'Connell

October 2006

Thesis Summary

Internet technologies are facilitating the development of an environment where companies are outsourcing activities that are not a part of their core competencies to third-party service providers. Customers and service providers can locate and identify each other over the Internet, negotiate the terms and conditions electronically, connect with each other dynamically, transact business and cease their relationship when the service is no longer required. One of the key enabling factors for the provision of services in this environment is the agreement of an electronic contract between the service provider and service consumer. This contract known as a Service Level Agreement (SLA) defines the terms and the quality of the service that is to be provided and the actions to be taken should the agreed service or quality not be met. With the introduction of these contracts, a requirement has arisen for a framework to manage and enable the service provider to meet the guarantees that are defined in each SLA contract.

A problem with the development of a framework for the management of services is providing the capability to support customer requirements that are continuously evolving both in terms of the services being provided and the quality levels associated with those services. A second challenge for such a framework is to enable the service provider to be able to fulfil customer expectations for different service and service quality requirements in a cost effective manner. A service provider should be capable of exploiting economies of scale by sharing service resources across all customers. This reduces both the cost of providing dedicated resources for each customer and the associated management overheads, but however introduces the additional complexity of providing differentiated levels of service quality to each customer.

This thesis describes DSM, a framework for the management of dynamic services in a service provider environment. DSM provides mechanisms necessary for dynamically building specific customer services with associated quality guarantees. DSM allocates service provider resources to customer requests dynamically and uses optimisation techniques in order to do so in a cost effective manner. DSM enables service providers to easily create new service offerings through the composition of existing

resources that are implemented by the service provider while also enabling the service provider to offer the service at defined levels of quality in order to meet customer expectations.

The aim of this thesis is to present the design and implementation of the DSM framework that supports the provision of dynamic services that have associated service level guarantees. DSM enables the service provider to customise SLA contracts according to specific customer requirements while still adopting a dynamic provisioning approach to resource allocation, where resources are dynamically assigned to customer requests from a common pool. By adopting an object representation of each SLA contract in the Service Management Component, DSM is able to aggregate measurement data from each of the resources that comprise the service offering and continually monitors the compliance of each SLA. This approach is then used by the scheduling algorithm adopted by DSM to dynamically allocate resources to best fit the needs of customer requests.

Acknowledgements

First and foremost, I would like to thank my supervisor, Vincent Wade, for supporting me throughout my Ph.D. studies. His support, both in terms of encouragement as well as technical have provided the best possible foundation for this thesis and are very much appreciated. I would also like to thank Prof. Paddy Nixon for advising me during the early stage of my studies and for helping me to initially formulate the basis for this research. Many thanks also to the other academics in the Distributed Systems Group for their help over the years. Thanks also to the staff of the Department of Computer Science for the support and friendship provided by them.

Finally, I would like to thank my whole family for their constant support and encouragement.

Contents

Chapter 1 Introduction **1**

1.1 Dynamic E-Business – Background and Issues 2

1.2 The Thesis 4

1.3 Thesis Contribution 5

1.4 Evaluation Criteria 6

1.5 Roadmap 7

1.6 Summary 8

Chapter 2 Overview of the Research Area **9**

2.1 Virtual Enterprises 10

2.2 Service Level Agreements 12

2.2.1 TeleManagement Forum SLA Management 14

2.3 Management of Dynamic Services 16

2.4 TeleManagement Forum eTOM 20

2.5 Web Services 21

2.5.1 Overview 21

2.5.2 Web Service Level Agreements 22

2.5.3 Web Services Composition 24

2.6 Summary 26

Chapter 3 State of the Art **27**

3.1 IBM Architecture for Service Management 27

3.2 Conformance 29

3.3 SAM 32

3.4 WSLA Framework 35

3.5 METEOR-S 37

3.6 CrossFlow 39

3.7 Oceano 43
3.8 WSAF 44
3.9 Analysis of the State of the Art Technology 46

Chapter 4 A Framework for Dynamic Service Management 48
4.1 Requirements for Dynamic Service Management 49
4.1.1 Dynamic Services 50
4.1.2 Differentiated Levels of Service 51
4.1.3 Optimal Provision of Services 52
4.2 DSM Architecture 54
4.2.1 Overview 54
4.3 Service Manager 56
4.3.1 Service Offering 56
4.3.2 SLA Manager 59
4.3.3 SLA Object 61
4.4 Service Execution Manager 62
4.4.1 Execution Planner 63
4.4.2 Service Scheduler 67
4.5 Resource Broker 69
4.5.1 Resource Availability 70
4.5.2 Monitoring and Measurement 70
4.6 Summary 70

Chapter 5 Framework Design and Implementation 72
5.1 DSM System Architecture Overview 73
5.2 Design and Implementation of DSM 75
5.2.1 Service Manager 75
5.2.2 Service Execution Manager 84
5.2.3 Resource Broker 88
5.3 Business Process Engine Integration 92
5.3.1 Process Deployment 93
5.3.2 Process Invocation 93
5.3.3 Management Events 93
5.3.4 Service Request Events 94

5.4 DSM Implementation 94
5.5 Design and Implementation Issues 95
5.5.1 Design Issues 95
5.5.2 Implementation Issues 96
5.6 Summary 96

Chapter 6 Evaluation 98
6.1 Summary of Thesis Goals 99
6.2 DSM Applications 100
6.2.1 Loan Assessment Application Service 101
6.2.2 Maintenance Planning Application 107
6.2.3 Analysis of Trials 112
6.3 Comparison with other Service Management Architectures 114
6.4 Summary 117

Chapter 7 Conclusion 118
7.1 Thesis Summary 118
7.2 Achievements 120
7.3 Thesis Contributions 122
7.4 Open Research Issues 125
7.5 Summary 125

Appendix A: Related Publications 126

Appendix B: Glossary of Abbreviations 127

Appendix C: Resource Optimisation Model 129

Appendix D: Sample WSLA Template 133

Bibliography 137

List of Figures

Figure 2-1 Dependency Graph 19
Figure 3-1 IBM Service Management Architecture 28
Figure 3-2 Conformance Architecture 31
Figure 3-3 SAM Architecture 33
Figure 3-4 WSLA Framework Architecture 36
Figure 3-5 CrossFlow Architecture 41
Figure 3-6 WSAF Framework 45
Figure 3-7 Features Provided By State of the Art Service Management 46
Figure 4-1 DSM Architecture 55
Figure 4-2 Service Definition 57
Figure 4-3 SLO Information Model 61
Figure 4-4 Probabilistic Invocation 64
Figure 4-5 Parallel Invocation 64
Figure 4-6 Sequential Activation. 64
Figure 4-7 Fastest processor triggered activation 65
Figure 4-8 Synchronise Activation 65
Figure 4-9 Process Weightings 66
Figure 4-10 Scheduler Search Tree 68
Figure 5-1 DSM System Architecture 73
Figure 5-2 Defining a reference to a service resource 76
Figure 5-3 Invoking a service resource in a business process 77
Figure 5-4 Switch Condition with Weighting 77
Figure 5-5 While Condition with Weighting 78
Figure 5-6 Pick Condition with Weighting 78
Figure 5-7 UML of Service Component 80
Figure 5-8 UML of Service Parameter 81
Figure 5-9 UML of Qos Metric 82
Figure 5-10 Aggregating QoS Metrics 83

Figure 5-11 Aggregating the Cost QoS Metric 84
Figure 5-12 Aggregating the Execution Time QoS Metric 84
Figure 5-13 Service Execution Manager Architecture 85
Figure 5-14 Planning Service Execution 86
Figure 5-15 UML of Scheduler 87
Figure 5-16 Executing a Service Instance 88
Figure 5-17 Web Service Endpoint Description 89
Figure 5-18 Monitoring Web Service Execution 91
Figure 5-19 QoS Measurement Handler Deployment 92
Figure 5-20 Measurement Events 92
Figure 5-21 Summary of DSM Components 97
Figure 6-1: Loan Assessment Application Process Definition 103
Figure 6-2: DSM Loan Assessment Application User Interface 105
Figure 6-3 Acxiom Web Service Implementations 106
Figure 6-4 Maintenance Planning Application Process Definition 109
Figure 6-5 Maintenance Planning Application Fault Types 110
Figure 6-6 Comparison of DSM with other Service Management Architectures 116

Chapter 1

Introduction

This thesis presents the concepts and models provided by a framework, called Dynamic Service Manager (DSM), for providing management services to outsourced business processes on the Internet. DSM provides management services to service providers, enabling them to rapidly create new dynamic services that are offered at differentiated levels of quality to each customer. DSM provides novel techniques to facilitate the composition of dynamic services, enabling service providers to offer differentiated services in a cost effective manner.

Central to the DSM framework outlined in this thesis is a system for modelling the service level agreements between the different constituent components that make up the business process.

This chapter introduces the concepts behind dynamic business environments and the issues that need to be addressed when considering the dynamic environment of outsourcing business processes on the Internet. In addition, the goals of the thesis are stated, a summary of its contribution and the criteria for evaluation this thesis are outlined. Finally, a roadmap for the remainder of the thesis is given.

1.1 Dynamic E-Business – Background and Issues

Providing management services, including quality of service guarantees, are becoming increasingly important to organisations outsourcing services to third-parties on the Internet [67]. This thesis describes the design and implementation of a framework for providing management services to online service providers. The framework has a particular emphasis on monitoring quality of service guarantees and providing problem determination and resolution when those services fail to meet the pre-defined and agreed-upon quality guarantees.

The widespread use of the Internet within the business community is leading to a fundamental change in the way that services are being delivered by service providers and consumed by service customers [63]. In particular, there has recently been an extensive and accelerating amount of research and technical specifications enabling the development of interoperable e-Business interactions [24], both in the B2C and B2B domains, using the Web and the Internet as the underpinning technologies [27]. These specifications enable the development of an environment where service customers and service providers can locate and identify each other over the Internet, negotiate the terms and conditions electronically, connect with each other dynamically, transact business and tear down their relationship when it is no longer required. This development of cross-organisation services can be termed Dynamic e-Business (DeB) [5].

Enabling such cross-organisational services, however, does not simply constitute a technical integration problem [63]. A service relationship also constitutes a business relationship between organisations. One of the very important elements of dynamic e-Business is an electronic contract that describes the role of the various parties involved in a DeB environment and the service level agreements (SLA) that are negotiated between them [50]. In particular, an SLA defines the agreement between a resource or service provider and consumer of the resource, regarding quality of service in the operation of a service hosted by the resource provider. In addition to specifying the agreement between the parties, the SLA is used as input to mechanisms

and processes that will be responsible for enforcing and monitoring the agreement [72].

With the introduction of SLAs between the members of a DeB, management services are required to ensure that the service is delivered within the bounds of the SLA [36]. These management services allow the definition and measurement of quality of service (QoS) parameters and service level guarantees in the SLA, and can detect violations during the interaction between a customer and a service provider. The management service is also responsible for problem determination and resolution and for reporting of performance data. While every service provider is solely responsible for the delivery of its service, the management services are independent of the service itself, and as such, can be delegated to a third-party Management Service Provider (MSP) [62].

One of the most significant challenges in managing dynamic services lies in the ability to support services that are continually evolving in structure and service quality levels [29]. To provide fine-grained outsourcing in a cost-effective manner, it is essential to support the automated management of the entire life-cycle of the business relationship, from creating the service offering and provisioning of the service, to the management of the execution of the service, including allocating resources to the service and managing compliance with the agreed-upon SLAs [33].

DSM unlike existing management techniques takes into account the characteristics of a dynamic e-business environment, including:

1. **Dynamic Services.** The framework must be capable of supporting customer requirements that are continuously evolving both in terms of services being provided and the quality levels associated with those services. The service provider must be capable of delivering the required services to each customer.

2. **Differentiated Levels of Services.** Individual customers have the ability to not only define the exact service features that they require, but also the service quality levels associated with each of those services. The service provider

must be capable of providing the levels of service required in a cost-effective manner.

3. **Optimal Provision of Services.** The underlying implementation of provisioned services should be continually managed to ensure the most cost effective implementation for service providers whilst minimising penalties service level violations.

While dynamic allocation of resources and the associated management functionality have been strongly researched, the research to date has however been primarily focused at the networks and resource layers. These approaches have been less successful when applied to the area of application Web services, which are software systems that run over networks. The analysis of the state of the art in Chapter 3 focuses on service management at the application layer. This analysis will show that where service composition has been addressed, resources are typically statically assigned to customer requests. Where dynamic resource allocation has been considered the dynamic composition of service offerings and the dynamic aggregation of measurement data both of which ensure that the overall runtime SLA parameters can be met has not been addressed. Also the current state of the art has yet to propose a unified framework that allows cost effective service provisioning based on the runtime requirements of customers using optimisation techniques.

1.2 The Thesis

This thesis describes the novel techniques that have been adopted in the design and development of the DSM framework to overcome the challenges of business process outsourcing in a dynamic environment. In this thesis a framework for dynamic service management is proposed. In the context of this thesis, a framework is defined as [85]:

> *"... a generic software architecture together with a set of generic software components that may be used to realise specific software architectures."*

The DSM framework provides mechanisms necessary for building customer-specific dynamic services with associated service guarantees. DSM allocates service provider resources to customer requests dynamically and uses optimisation techniques in order to do so in a cost effective manner.

DSM enables service providers to easily create new service offerings through the composition of existing resources that are implemented by the service provider while also enabling the service provider to offer the service at defined levels of quality in order to meet customer expectations. DSM enables the service provider to customise SLA contracts according to specific customer requirements while still adopting a dynamic provisioning approach to resource allocation, where resources are dynamically assigned to customer requests from a common pool. DSM also adopts a dynamic resource allocation approach for service fulfilment and manages the allocation of Web service resources and the prioritisation of service requests in order to meet the obligations defined in the customer SLA contracts. DSM incorporates a scheduling algorithm to allocate resources to customer requests that best fits the requirements defined in each SLA. This thesis, while addressing the issue of dynamic service provision and aggregation of QoS metrics, makes no attempt at providing support for transaction and fault handling across the invoked services.

1.3 Thesis Contribution

The major contribution of this thesis is the novel techniques that have been adopted in the design and implementation of the DSM framework which:

1. *Enables dynamic outsourcing of services to service providers.* DSM automates the processes associated with establishing a service relationship with a provider and manages the agreed-upon SLA across the entire lifecycle of the relationship. In addition DSM minimises the effort with establishing a service relationship, as it performs dynamic allocation of resources to service requests.

2. *Enables service providers to offer differentiated levels of service to each customer.* DSM allows the service provider to offer each customer the same service at unique service levels, depending on customer requirements. By enabling the service provider provision resources dynamically (i.e., on demand), service providers can best utilise their computing resources and can dynamically redistribute resources across multiple customer workloads.

3. *Provides the service provider with mechanisms that fulfil service requests in the most cost effective manner.* DSM utilises optimisation techniques to determine a cost effective allocation of resources to customer requests which minimises the penalties that are incurred due to the violation of SLA contracts.

1.4 Evaluation Criteria

A prototype of the DSM framework has been implemented in order to demonstrate that it meets the thesis goals outlined in section 1.3 of this chapter and also to validate its applicability in different application scenarios.

The evaluation of this work includes the development of two applications to verify that DSM meets these requirements. The applications chosen for this thesis evaluation are taken from different application domains with very different SLA requirements and demonstrate how the objectives of this thesis have been met. The first application that has been implemented is a financial services mortgage loan assessment application. The application provides an online service to financial institutions to evaluate a customer's mortgage loan and predict the likelihood of loan pre-payment. The second application that has been implemented is a simulation of a maintenance planning application. The application provides fault resolution capabilities to railway maintenance providers.

The characteristics and functionality of the DSM framework have also been analysed and compared to other works described in the state of the art (chapter 3). From the examination of similar frameworks found in the literature it can be shown that little

support (if any) for the dynamic provision of services is provided by these frameworks. DSM also provides the ability to aggregate measurement data from each of the resources that comprise the service offering and continually monitors the compliance of each SLA to ensure the original requirements are being met. This feature is not typically supported by the leading systems outlined in the state of the art. Typically, their approach is to statically assign resources to each customer rather than dynamically assign resources that bests fits the current customer demand. Where dynamic resource allocation is provided, the overall quality of a composite service that is offered to a customer is not considered.

1.5 Roadmap

The remainder of this thesis is organised as follows:

In chapter 2 an overview of the technologies and concepts in the areas of service management and dynamic outsourcing is presented. An overview of virtual enterprises and details the issues that arise in the outsourcing of services in a virtual enterprise environment is first presented. This is followed by a description of service level agreements and their characteristics. A section of dynamic service management is then presented. Finally a description of web services and web service composition is presented.

Chapter 3 describes the state-of-the-art work developed in the area of dynamic services and service level management. Their characteristics and limitations are also presented.

The DSM framework is described in chapter 4. The primary design influences for the framework along with the requirements for such a framework are discussed. The techniques employed by DSM to facilitate the development of management services in dynamic environments are then presented.

The design and implementation details of DSM are presented in chapter 5. A detailed description of each of the components of DSM is firstly presented. A detailed description of the implementation is also given.

Chapter 6 outlines the evaluation of the thesis. The chapter firstly presents a summary of the goals of the thesis and how each of these goals is to be evaluated. Following on from this, a description of management services provided by DSM in two different domains is presented. Finally, a comparison of DSM with the frameworks found in the literature and presented in chapter 3 is given.

Finally, chapter 7 concludes the thesis. A summary of the thesis, its contribution and the evaluation of its goals are stated. Future work and open research issues are also discussed.

1.6 Summary

This chapter outlined the goals and scope of the work described in this thesis – a description of the approach adopted by DSM. The chapter began by presenting a definition of dynamic e-business and the issues that need to be addressed when providing management services in such an environment. This was followed by an overview of the DSM framework. In addition, the contribution of the thesis and the evaluation criteria were outlined. Finally, a roadmap to the remainder of the thesis was given.

Chapter 2

Overview of the Research Area

This chapter presents an overview of the research area with which this thesis is concerned. The following sections introduce the key concepts and technologies present in the areas of dynamic outsourcing and service management.

The technologies that have been used for implementing a framework for the management of dynamic services, which will be presented in chapters 4 and 5 respectively, are also described in detail. These technologies include Web services and the associated frameworks and standards for Web service composition and the specification of Service Level Agreements (SLAs) for Web services. The technologies described in this chapter are seen as important tools for building a framework for dynamic service management.

The rest of this chapter is organized as follows. The chapter presents an overview of virtual enterprises, their characteristics, and requirements for service management in section 2.1. A detailed review of service level agreements is then provided in section 2.2. Section 2.3 presents an overview of service management techniques that are used by a management service provider in a dynamic e-Business environment. In section 2.4 a description of the Enhanced Telecom Operations Map (eTOM) Business Process Framework from the TeleManagement Forum is presented. An overview of Web services, including Web service level agreements and Web service orchestration is then presented in section 2.5. Finally the chapter is summarised in section 2.6

2.1 Virtual Enterprises

This section presents an overview of virtual enterprises, their characteristics and requirements for management services.

A virtual enterprise is an organisation created from physically distributed constituents which are linked electronically to enable interaction and co-operation normally associated with a centralised enterprise [75, 86]. It is a set of legally independent organisations of varying types who voluntarily co-operate to seize a market opportunity [91]. They are represented by at least one partner to the external world and they agree to produce a common output, e.g. a product or service, based on a common understanding of their business rules and business processes. To achieve this, they should share their resources, core competencies, skills and know-how in order to become quicker, flexible and more global.

The virtual enterprise is achieved through the co-operation of independent enterprises, which together realise the complete field of goods and services [56]. In this structure, each enterprise realises only one special part of the value chain. However, this is no regular co-operation. Partners in a virtual enterprise provide their critical, core competencies. Thus, a virtual enterprise is ideally a combination of best-of-class (complementary) core competencies [90].

The two main reasons for organising activities in the form of a virtual enterprise may be described as [88]:

- an increasing need for flexibility, in addition to which the necessary core competencies can only be obtained on collaborating with external partners
- the need for efficiency by sharing resources with other partners

Virtual Enterprises are globally distributed, and exploit information and communication technologies to support their operation. Information systems allow virtual organisations to monitor feedback and refine their configurations, allowing them to constantly evolve [13].

In a virtual enterprise, the co-operating units typically contribute their core competencies [81]. The independent units contribute partial business processes which are combined together to form the global business process describing the virtual enterprise [45]. Because in a virtual enterprise individual enterprises fulfil only parts of the value chain to produce the product, the main problem in assembling a virtual enterprise is the mapping of the partial processes to the individual enterprises [92]. It can be divided into two phases: In the first phase, a suitable business process partitioning is worked out. In the second phase, the process chain of the virtual enterprise is assembled and instantiated [92].

In a virtual enterprise, the establishment of the business links needs to be achieved in a fast and flexible way to guarantee a short time to market while allowing a dynamic reaction to new customer demands and changing offers of service providers in electronic commerce environments [67]. One approach to achieving this dynamic outsourcing is through the use of workflow management systems [1, 66, 67]. A workflow management system is designed to handle business processes [8, 67], where a business process is a set of one or more interconnected activities which collectively realise a business objective or policy goal, normally within the context of an organisational structure defining functional roles and relationships [111]. A workflow is the automation of a business process, in whole or part, during which documents, information, or tasks are passed from one participant to another, according to a set of predefined rules [111].

A business process is described as cross-organisational if there is the possibility that at least one of its activities is outsourced to a different organisation [67]. A central concept in the notion of a cross-organisational workflow is a *service* [67]. A service comprises any action done by one party, i.e., the service provider, on behalf of another party, i.e., the service requester. The service requester may use the services offered by various external service providers to outsource parts of a cross-organisational workflow. From the point of view of business process management, companies usually define in an agreement the goal of the outsourced activity and the way in which information has to be disclosed to partners [74]. The agreement also describes, amongst other things, the circumstances in which both parties interact in the course of the service. These properties of a business relationship enable both organisations to

have several business partners in a similar relationship. A requester can pass the same part of a business process to different providers, whereas a provider can use the same (internal) process to service different requesters [74].

The life-cycle of a virtual enterprise can be described in three phases [89, 91] as follows: An initial built-in phase during which members of the virtual enterprise are establishing and configuring the linkage between them, including negotiations of electronic contracts and integration of the processes across the different organisations; an execute/manage phase during which the cross-organisational processes are executed and services are delivered to the virtual enterprise customers; and finally a terminate phase during which the virtual enterprise members terminate the contracts and remove the linkage between the different organisations.

2.2 Service Level Agreements

This section presents an overview of Service Level Agreements, their structure, life-cycle and management.

Enabling cross-organisational services requires a business relationship between organisations, defined in a contract. An important aspect of a contract for IT services is the set of quality of service (QoS) guarantees a service provider gives. This is referred to as a service level agreement [70, 72]. A Service Level Agreement (SLA) is defined as a formal, negotiated contract between two parties, viz., a service provider and a customer [107].

In addition to specifying the agreement between the parties, the SLA is used as input to the mechanisms that will enforce and monitor the agreement. It consists of a list of services required by the customer (and hence furnished by the resource provider), and for each service a set of parameters which describe a level of service the customer requires, and that the service provider is able to support [60]. The SLA describes only the resources required by a single customer and the service provider. It is the

responsibility of the service provider to consider all SLAs when guaranteeing a level of service to any individual customer [72].

Representatives of both the service provider and customer prepare the SLA [27]. It documents the agreement between the resource provider and the customer on the values of service parameters of various kinds. The types of service parameters typically include: the services to be supported by the resource provider and a description on how a service can be accessed by the service customer; the workload expectations; and a set of quality parameters defining the QoS guarantees given by the service provider for the service, including penalties that are incurred for violating these parameters.

In order that management services can be provided in a dynamic e-business environment, SLAs must be represented in a format that can be understood by the management service [19, 49]. In order to achieve this, the parameters defined in the SLA are defined as a set of assertions. Each assertion is an atomic group of statements that is agreed upon by the parties defining the contract [47]. At any given time, an assertion may be TRUE or FALSE depending on whether the party is meeting the obligations stated in the assertion or not. Statements in an assertion are made up of *logical predicates* whose value can be uniquely determined. The logical predicates are composed using *variables* as well as *logical operators, quantifiers*, *set operations*, and *constraints* on those variables [33]. Variables may be simple variables (e.g., the current network load), statistical variables (e.g., averages or variances), or trends (time dependent variables such as growth rates). They reflect measures that are meaningful for the operation of the service. An SLA is said to be *in compliance* if all assertions within it are TRUE.

A number of guidelines for SLAs have been published, including those by the ASP Industry Consortium (ASPIC) [15], the Information Technology Association of America (ITAA) [55] and the TeleManagment Forum (TMF) [106, 107]. The TMF's SLA Management Handbook is discussed in detail in the next section. This handbook is of particular relevance to this thesis as it describes the end-to-end SLA required for business applications that are composed of many business services, each of which has an associated SLA.

2.2.1 TeleManagement Forum SLA Management

The TeleManagement Forum SLA Management Handbook [106, 107] assists service providers in developing new services with associated SLA parameters, align SLA parameters to meet Customer requirements and internal processes, assign internal processes according to SLAs, and respond to SLA requirements from other service providers. The SLA Handbook is the result of consideration of what elements of an SLA would assist all parties involved, including customers, service providers and their respective vendors. The SLA Management Handbook provides a structure for defining agreements including the relevant SLA parameters and values can be agreed upon that are achievable and verifiable.

The SLA Management Handbook is structured in four separate volumes, as follows. Volume 1 represents the executive summary for the handbook. Volume 2 outlines the service provider SLA to its customers and other service providers and introduces the SLA parameter framework. This framework is a tool for categorising parameters and organises these parameters into six categories based upon service and delivery technology and upon measures of service performance and measures of average service performance.

The service parameter framework distinguishes between the individual user view of the service and the aggregated view. The individual user view covers service properties such as the service interface or the maximum service down-time that an individual service user could experience during a specified time period. The aggregated view essentially captures the average performance over all service users during a specified time period and may include items such as service billing and aggregate availability. Technology specific service parameters are related to the telecommunications technology supporting the service, particularly when the service offered is a network bearer service. Service-specific performance parameters are typically related to the application supported and include service-specific or application-specific technology parameters such as reliability and availability of computer servers, databases and so on. Service/technology independent service parameters are those that are often, if not always, specified in an SLA. Examples include percentage availability, mean time between failures, outage intensity, mean

time to provision service, mean time to restore service and so on. Other parameters that may be included in this set are integrated or consolidated billing for a set of services, accuracy of billing and payment terms, billing period and security for both service access and delivery of information.

Examples of these SLAs are discussed in Volume 3 of the handbook.

The issues in the provision of end-to-end SLAs for enterprises are addressed in Volume 4 of the handbook. In support of enterprise or business applications, business services are available to facilitate the application. One or more business services are required to support a business application. In turn, business services are supported by network services. Each layer in the hierarchy has scope for an SLA between the lower layer (the service) and the higher layer (the application). In general, there will be a tier of SLAs as the business function is satisfied with different service offerings, and in some instances there will be peer services where the service requires the co-operation of other services of the same type.

A service and its associated SLA are divided into six life cycle stages to clarify the roles of the customer and the service provider. The six life cycle stages are as follows: product/service development, negotiation and sales, implementation, execution and assessment and decommissioning. Each life cycle stage addresses specific operations processes in the TeleManagement Forum eTOM [105] discussed in section 2.4. The SLA life cycle provides a complete process description by delineating interactions between well-defined stages.

2.3 Management of Dynamic Services

This section presents an overview of service management techniques that are used by a management service provider in a dynamic e-business environment.

From a service providers perspective, application service provisioning has brought a new challenge, namely how to manage these services so that high availability is guaranteed [63]. In general, service providers have well-developed network management infrastructures to operate their physical networks consisting of servers, switches, routers, links and so on [59]. Thus, an important objective is to leverage the existing network management infrastructure and enhance it to provide application service management [51, 99]. In order to provide end-to-end application service management for every customer, the service provider has to deploy a service management system in addition to the operational network management system [59].

In traditional network management, the general approach is to partition a physical network environment into monitored domains, each of which is under the supervision of a management system, which are designed according to topological considerations [41]. An application service that comprises instances of distributed components typically belong to multiple monitored domains, therefore requiring the management to span multiple monitored domains [25]. Kar *et al.* [59] propose the concept of *service management domains* which are virtual domains built from resources and relationships pertaining to the monitored physical domains.

Service Management can include additional parties that provide management services, termed a *Management Service Provider (MSP)* [63]. This third-party entity is selected to oversee the electronic contracts formed between the service provider and customer. Typical functions performed by the MSP include the provisioning of the various measurement services, contract monitoring for SLA violation, problem determination and performance reporting [63]. These functions can be performed either by one or by multiple entities. For instance, provisioning and problem determination could be done by one entity which can be an integral part of the Service

Integrator, whereas contract monitoring and SLA violation detection could be done by an agreed upon third party [50].

Providing management services in a dynamic e-Business environment pose a number of challenges for service management, due to the dynamic nature of the e-Business environment. These include [63]:

- Provisioning of management systems in the customer and service provider environments to enable appropriate monitoring. This step needs to be done dynamically as new service providers replace or join existing ones in the dynamic e-Business environment.
- Detection of contract violation, finding the source of the violation and initiating corrective measures.
- Problem determination, which is a difficult enough task in large distributed systems, becomes even more so in the context of dynamic e-Business. This is because of the multiple organisation boundaries involved in the completion of an e-business transaction. In the event of a problem, the management systems of the different organisations have to co-operate by exchanging relevant information (e.g., monitored data, log file extracts and so on.) in order to find the root cause of the problem.

Managing large-scale applications that cross administrative boundaries is a problem because current management solutions either allow partners access to all management information (e.g. by providing remote consoles) or deny access to this information [17]. Inter-domain management in competitive environments places two fundamental requirements on the management system [17]:

1. Service management and diagnosis require the knowledge and view of the end-to-end service. This means that management information has to flow across administrative domain boundaries to provide an end-to-end view.

2. Business requirements restrict information sharing across domains because the details of the service implementation and much of the customer information is considered proprietary by each business.

Meeting these objectives requires that service management systems need to both selectively share information about the component of the overall service while hiding the details [28].

One of the most significant challenges in managing modern enterprise systems lies in the area of problem determinations – detecting system problems, isolating their root causes, and identifying proper repair procedures [20, 41]. Problem determination is crucial for reducing the length of system outages and for quickly mitigating the effects of performance degradations, yet it is becoming increasingly difficult task as systems grow in complexity. An approach to managing this complexity, and thereby simplifying problem determination, is in the study of *dependencies* between system hardware and software components [16, 61].

The basic premise underlying dependency models is to model the system as a directed, acyclic graph in which nodes represent system components (services, applications, OS software, hardware, networks) and weighted directed edges represent dependencies between nodes [26, 108]. An example dependency graph is illustrated in Figure 2-1

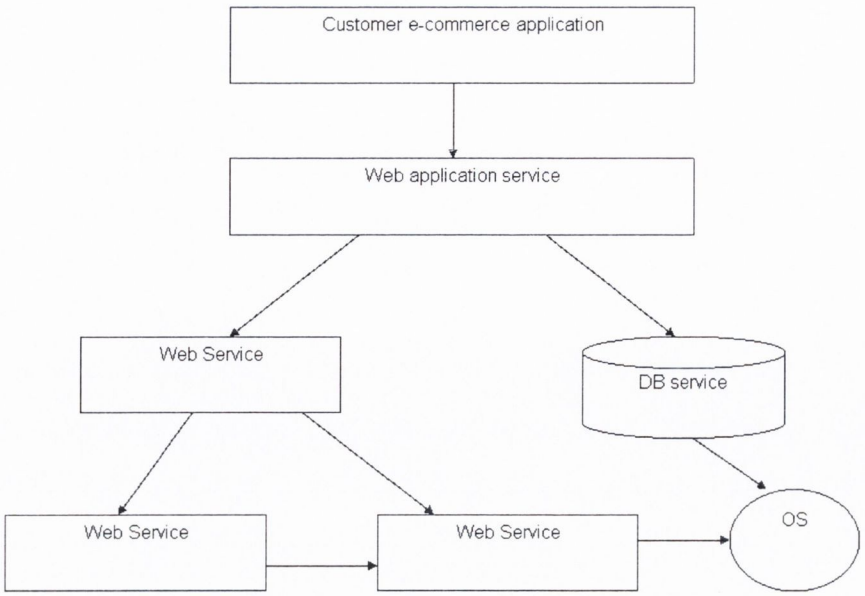


Figure 2-1 Dependency Graph, adapted from [20]

Dependency graphs provide a straightforward way to identify possible root causes of an observed problem – one must simply trace the dependency edges from the problematic node (or entity) to discover all of the potential root causes [62].

The DMTF Common Information Model (CIM) [37, 38] provides an approach to modelling dependencies (being usually perceived as a specific kind of association) which are modelled as classes, allowing inheritance [39]. While the specification addresses the dependency problem, it does not determine dependencies at runtime [62]. The dependencies are specified at the installation phase of the software and the models provide no support once the application gets instantiated.

There are two main approaches to using dependency models for problem diagnosis and root cause analysis [59]. The first is in the context of event correlation systems. In these systems, incoming alarms or events are first mapped onto corresponding nodes of the dependency graph, and then the dependencies from those nodes are examined to identify the set of nodes upon which the most alarm/event nodes depend. The other main technique for using dependency models in root-cause analysis is to use the model graph as a map for performing a systematic examination of the system in search of the root cause of a problem.

2.4 TeleManagement Forum eTOM

The Enhanced Telecom Operations Map (eTOM) Business Process Framework [104, 105] is the ongoing TM Forum initiative to deliver a business process model or framework for use by service providers and others within the telecommunications industry. The TM Forum eTOM describes all the enterprise processes required by a service provider and analyzes them to different levels of detail according to their significance and priority for the business. The focus of the eTOM is on the business processes used by service providers, the linkages between these processes, the identification of interfaces, and the use of customer, service, resource, supplier/partner and other information by multiple processes. The eTOM is a reference framework for categorising all of the business process activities that a service provider will use, in a structured manner that allows these to be addressed at various levels of detail.

Through the use of process decomposition, eTOM takes a structured approach to analysis of a business through consideration of an enterprise's business processes and their internal structure [104]. The process decomposition begins at the enterprise level and defines business processes in a series of groupings. The eTOM represents the whole of the service provider's enterprise environment and is designed in a generic way so as to be organisation, technology and service independent. Conceptually, eTOM can be viewed as having three major process areas: Strategy, Information and Product, covering planning and lifecycle management; Operations, covering the core of operational management; and Enterprise Management, covering corporate or business support management.

The eTOM groups processes into end-to-end processes that are required to support customers and manage the business. The Operations process area contains the direct operations process groupings of Fulfilment, Assurance, Billing, and Operations Support & Readiness. Within the Operations process area, the Assurance Grouping is responsible for the execution of proactive and reactive maintenance activities to ensure services provided to customers are continuously available and performing to SLA or QoS performance levels [103]. It performs continuous resource status and performance monitoring to proactively detect possible failures. It collects

performance data and analyses them to identify potential problems and resolve them without impact to the customer. This process manages the SLAs and reports service performance to the customer. It receives trouble reports from the customer, informs the customer of the trouble status, and ensures restoration and repair.

2.5 Web Services

This section presents an overview of the Web services standard, including the specification of service level agreements for Web services and the composition of Web services through the use of business process technology.

2.5.1 Overview

Web Services have emerged as a new distributed computing framework, moving to address the problem of enabling interactions between heterogeneous applications distributed over the Web in which no single party has complete control.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format, and other systems interact with the Web service by using SOAP XML messages conveyed using HTTP [110].

Web services technology involves a family of XML-based protocols that extensively use XML to describe an operation to execute or data to exchange with another Web service [110]. A group of Web Services interacting together forms a particular Web service application in a Service-Oriented Architecture (SOA) [29, 47]. The advantages of Web services are [32]:

1. Independent as much as possible from specific platforms and computing paradigms.

2. Developed mainly for inter-organisational situations rather than for intra-organisational situations.
3. Easily composable (i.e., composition of Web services does not require the development of complex adapters).

Therefore, Web services can be considered as future unified middleware that can be used for dynamic service-oriented composition of applications built on different platforms [18]. A key goal of Web services is to provide a common representation of applications which use diverse communication protocols and interaction models [32]. In Web services this is achieved by separating abstract application descriptions from protocol bindings. An abstract application description uses XML Schema for describing the data types the application uses and the messages exchanged [7].

Service registries enable both the location of Web services and also the just-in-time integration of service components [32]. The Universal Description Discovery and Integration (UDDI) [109] specification defines the operation of a registry supporting this type of service location.

2.5.2 Web Service Level Agreements

WSLA consists of a flexible and extensible language based on the XML schema for the specification of SLAs for Web services [35]. The WSLA Language Specification [73] defines a type system for the various SLA artefacts.

WSLAs are agreements between a service provider and a customer and as such define the obligations of the parties involved. Primarily, this is the obligation of a service provider to perform a service according to agreed-upon guarantees for IT-level service parameters (such as availability, response time and throughput) for web services and business process-level service parameters. The assertions of the service provider are based on a detailed definition of service parameters including the algorithms – how basic metrics should be measured in systems and how they are aggregated into composite metrics and SLA parameters. In addition, a WSLA can express the operations of monitoring and managing the service. This may include third parties

(such as Management Service Providers) that contribute to the measurement of metrics, supervision of guarantees or even the management of deviations of service guarantees.

A WSLA agreement complements a service definition. While a service description, for example, using WSDL defines the service interface relationship between a service and its using application, the WSLA defines the agreed performance characteristics and the way to evaluate and measure them [64]. Service descriptions are input to the design and implementation of the service system and the client application using its service. The WSLA provides input to the measurement and management system of an organisation that checks and manages an organisations compliance with a WSLA. The instrumentation of service and service-using application can be instrumented to gain measurements for the evaluation of the WSLA [73].

The specification of WSLA assumes an abstract model of the runtime management of a WSLA [73]. It is assumed that the measurement and management functionality is divided into three groups of functionality: The measurement functionality, which receives measured metrics from the system's instrumentation; the set of metrics that are used in the guarantees of the WSLA are made available by the measurement function as SLA parameters; and the condition evaluation function that evaluates the guarantees of the WSLA which are defined as predicates over SLA parameters. The WSLA has as yet a limited model of management function [64]. Management implements actions that are invoked upon guarantee violations. The subsequent course of action to remedy the problem is not the subject of the WSLA specification.

One or both parties may choose to commission a part of the WSLA management activity to other parties. These third parties are called supporting parties. They are defined in the WSLA along with the service customer and service provider, the signatory parties of the WSLA. Supporting parties are sponsored by either signatory party or by both of them. Supporting parties can act in any combination of the following roles: A measurement service that implements a part or all of the measurement function required by one or both signatory parties; a condition evaluation service; and a management service.

2.5.3 Web Services Composition

Web services based on the service oriented architecture provide a suitable technical foundation for making business processes accessible both within enterprises and also across enterprises [3, 69]. This section describes two XML based standards for assembling a number of Web services to form a business process.

2.5.3.1 BPEL4WS

BPEL4WS [54] (Business Process Execution Language for Web Services) is a workflow-based composition language for Web services. It supersedes XLANG [80] and WSFL [68] as a standard for Web services flow specification. It defines a notation for specifying business process behaviour based on Web services. The composition of Web services consists of providing logic around a set of interactions between the composition and the Web services that participate in it [100]. These interactions are simply invocations to the operations offered by the services.

The BPEL4WS specification introduces several types of primitive activities [68]: to allow for interaction with the applications being composed (invoke, reply, and receive activities), wait for some time (the wait activity), copy data from one place to another (the assign activity), indicate error conditions (the throw activity), terminate the entire composition instance (the terminate activity), or do nothing (the empty activity). Data is available in a global container.

These primitive activities can be combined into more complex algorithms using structured activities. These are the ability to define an ordered sequence of steps (the sequence activity), the ability to have branching using the common "case-statement" approach (the switch activity), the ability to define a loop (the while activity), the ability to execute one of several alternative paths (the pick activity), and finally the ability to indicate that a collection of steps should be executed in parallel (the flow activity). Within activities executing in parallel, one can indicate execution order constraints by using *links*.

The BPEL4WS workflow model includes the ability to scope activities and specify fault handlers and compensation handlers for scopes. A fault handler gets execution when an exception arises, for example through the execution of the *throw* activity; compensation handlers are triggered due to faults or through *compensate* activities that force compensation of a scope.

In BPEL4WS, entities are interacting with the composition, including the Web services being composed, are viewed as partners. Each partner may support some functionality described using WSDL port types. BPEL4WS supports compositions that describe *abstract processes* [65]. Typically, they define a set of business operations to be performed in a particular order, and the role played by each enterprise in the overall process. The BPEL4WS specification defines a restricted syntax suitable for expressing such abstract processes. This enables a choreography to match services instances to the descriptions dynamically and handle incompatibilities between services.

Choreographies of Web services can in turn be offered as a Web service through BPEL4WS [6]. It enables third-party service-providers to create and offer such choreographies, composed of publicly available services and it allows enterprises to expose different choreographies of their internal services to different clients [82]. Furthermore, it enables recursive composition, which in turn enables inter-workflow interaction, higher levels of reuse and additional scalability.

2.5.3.2 OWL-S

OWL-S [9, 34], formerly DAML-S is an initiative to provide an ontology markup language expressive enough to semantically represent capabilities and properties of Web services. OWL-S aims to make Web service descriptions more computer interoperable, thus enabling automation of a variety of tasks including Web service discovery, invocation and composition. The structure of the OWL-S ontology consists of a service profile for advertising and discovering services, a process model which gives a detailed description of a service's operation and a service grounding which provides details on how to interoperate with a service via message exchange.

Each service can be viewed as a *Process* and its *Process Model* is used to control the interactions with the service [98]. It categorises three types of processes. The first type is an *atomic process*, which does not have any sub-processes and can be executed in a single step. The second type is a *simple process*, which is cannot be invoked as it is used as an abstraction for representing an atomic or composite process. A *composite process* is the third type, which is decomposable into sub-processes. A composite process uses several control constructs to specify how inputs are accepted and how outputs are returned.

While it may be considered that OWL-S and BPEL4WS overlap in the functionality provided, it can be seen that OWL-S augments the functionality provided by BPEL4WS to include tasks such as dynamic partner binding and semantic integration [77], and is particularly suited to environments where the number of compositions of Web services is large and cannot be pre-defined [84]. BPEL4WS however is more suited to structured process compositions, where the process structure is defined manually in advance of the composition executing. This structured composition is best suited to the service composition requirements outlined in chapter 1.

2.6 Summary

This chapter presented an overview of virtual enterprises, their characteristics, and requirements for service management. This was followed by a detailed review of service level agreements. Following on from this, an overview of dynamic service management techniques were presented. Finally a review of Web services was presented, including an overview of the definition of SLAs in Web services and the composition of Web services through business processes.

Chapter 3

State of the Art

The following sections present a number of frameworks and architectures that provide service management mechanisms that have been particularly influential to this thesis. The systems discussed in this chapter focus primarily on providing service management mechanisms across the e-Business value chain rather than at the network and resource layers. These systems are discussed in terms of their overall architecture and their application to dynamic e-business environments. During this survey study, a series of limitations have been identified that prevent the use of state-of-the-art technology to provide support for dynamic service management. These limitations are targeted by a new proposal of dynamic service management, which will be described in the next chapter in this thesis.

3.1 IBM Architecture for Service Management

This section describes an architecture proposed by IBM for dynamic service management [63].

The architecture consists of two parts: (1) *A service Production System,* which hosts or implements the service and (2) a corresponding *Contract Management System* that deals with the monitoring and management of the contractual obligations defined in the contract clauses.

The Service Production System is an application server that consists of a Java Servlet [101] engine, a monitoring and management interface and an administration console that facilitates local management tasks. The application server hosts the servlets that implement the service for the user, which is typically exposed as a Web service interface.

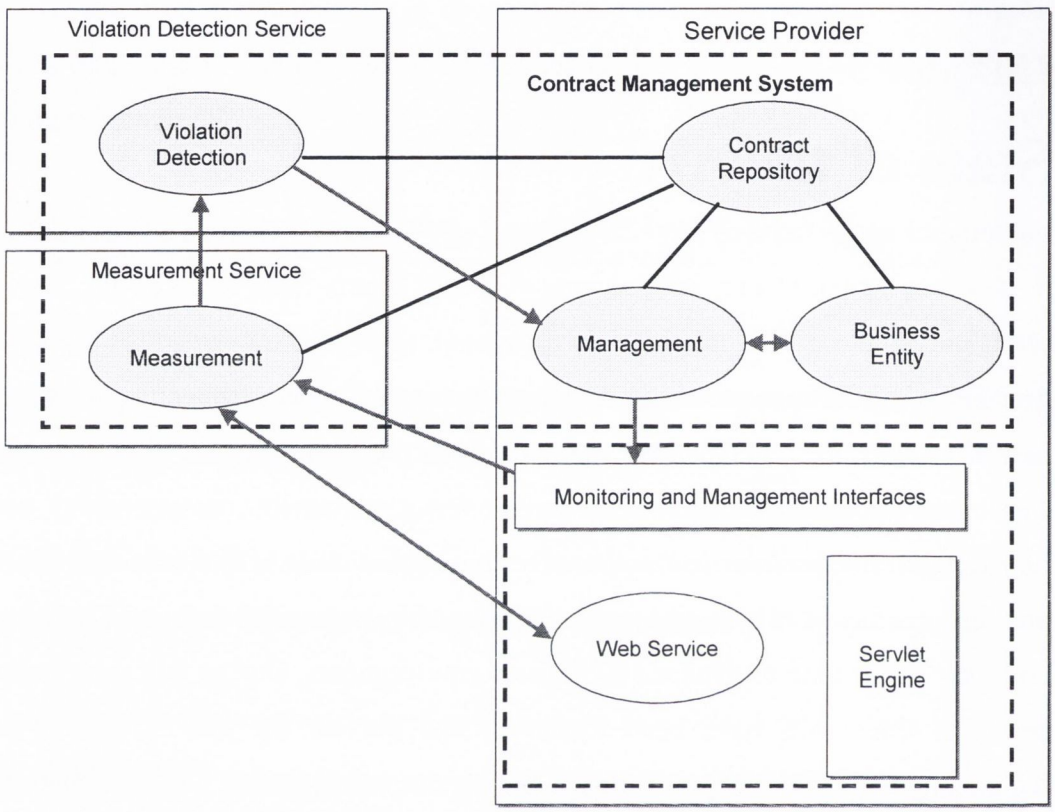


Figure 3-1 IBM Service Management Architecture (from [63])

The Contract Repository maintains the contracts of an organisation. Once the contract establishment functionality has signed a new contract, it is submitted to the contract repository. The repository also maintains links about all other functional components and external partners that are involved in the management of the contract. It is the entry point for administrative activities associated with contracts.

In addition to internal components, there are also an external *Violation Detection Service* and a *Measurement Service* involved in the measurement and management of the contract. The service provider runs his *Service Production System* and the internal components of the corresponding *Contract Management Syste*m. In a different

configuration there could be no external roles involved – all functionality implemented by the provider – or even multiple measurement and violation detection services.

IBM Service Management provides a framework for SLA management and compliance monitoring. It allows SLA contracts to be defined and managed whereby QoS parameters are defined as quantifiable metrics to that can be measured by a Measurement Service. In addition, it provides management services to take action when a SLA violation has been detected, including the re-allocation of resources to different services in order to minimise the penalties incurred as a result of the SLA violation. It does not however support resource assignment to individual customer service requests dynamically in order to best optimise the service delivery and minimise service violations. As a result, resource re-allocation only takes place on service violation and is not an integral part of the service management. Therefore, the framework can not be considered to manage the service provision in an optimal manner.

3.2 Conformance

Conformance [17] is a prototype implementation of an architecture targeted to allow automatic verification of quality of service guarantees as described in a contract. The implementation of Conformance is web-based with inter-domain communications using HTTP.

All external interactions to a given domain happen though one or *more Contract Verification Interfaces* provided by the domain. Service level agreements offered by the domain define the nature of information provided at the interface and the security parameters needed for the interaction. The architecture provides a recursive and hierarchical model for communication in a federation, thus providing a scalable solution. Inside the domain, the service manager directs the collection of management information using the service specific data contained in the service model. The data collection is guided by the contracts in the contract repository.

The *service model* includes a description of the service that this administration is managing. The service model provides the following functionality: it identifies the various components that enable a service. For example, if the top-level service being managed is e-mail, the service model would list the components as the email server host, the networks connecting the email host to the Internet, the email application itself, the name server used to resolve hostnames to Internet addresses and so on.; secondly it expresses interdependencies that exist among the different elements of the service. From the above email example, all components identified in the service model should function properly for the email service to work. The interdependencies capture the cause and effect between the components of a service; thirdly it identifies the measurements that are available from each component. Thus, the email server could identify the number of email transactions, and active measurements could be used to get an estimate of the response time seen by email clients.

The *contract repository* contains a set of contracts, which this domain has with its providers and customers. It contains information on how to validate an incoming contract verification request and places constraints on what data may be accessed from outside the domain as well as how that data is computed.

The *service manager* is the engine responsible for directing the verification task. It has the knowledge of how to evaluate an incoming contract verification request. This would involve interfacing with the contract repository to get the details of the contract, and interfacing with local resources and local system and service management modules to collect information needed to verify the contract. If evaluation of a contract has dependencies on other external contracts then it uses the contract verification interfaces provided by those external domains to collect the data.

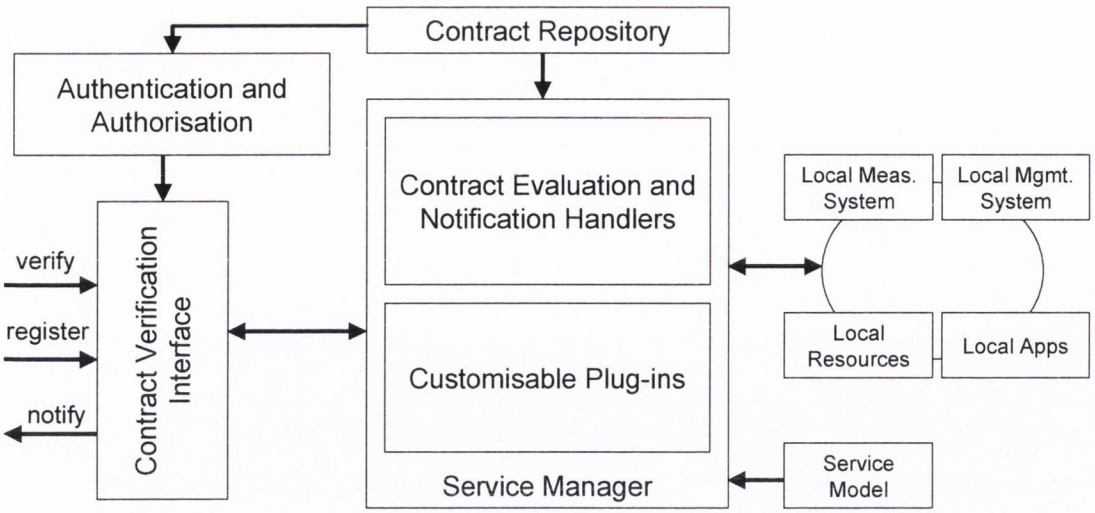


Figure 3-2 Conformance Architecture (from [17])

The service model describes the service implemented by the domain as well as the dependencies between service components. The System Dictionary, which is a part of the service manager, contains the abstract view of the service model in terms of high-level service attributes that are offered to customers. The System Dictionary is used to identify attributes as well as meta-information about the attributes such as which measurement plug-in is used to obtain the attribute value, how often the measurement is made, what parameters are necessary to measure it and so on.

For each customer SLA, information about the guarantees as well as information about the service components which impact the service offered to a given customer are placed in the customer SLA Database. A Contract Definition Language (CDL) is used to specify the assertions in the form of a template that uses the attributes and thresholds as parameters.

Objects that contain contract templates form the core of the engine. Because the same contract could be offered to multiple customers, customer-specific thresholds, bounds, and system parameters are filled in the template at contract evaluation time. This allows the templates to be shared across multiple customers.

Conformance provides a framework than enables both SLA monitoring and the selective sharing of management information across domain boundaries. While it supports the monitoring of SLA compliance against agreed targets, it does not support the dynamic allocation of resources in order to best meet these targets. Furthermore, Conformance does not address the creation of composite service offerings and the associated aggregation of measurement data. These two limitations make Conformance unsuitable as a framework for the management of dynamic services.

3.3 SAM

SAM is an SLA contract execution manager and SLA management system developed by IBM at the T.J. Watson Research Centre [21, 22]. SAM's technologies provide the following capabilities:

1. Enable the provider to deploy an effective means of capturing and managing contractual SLA data as well as provider-facing Service Level Management (SLM) data.
2. Assist service personnel to prioritise the processing of action demanding quality management alerts as per provider's SLM objectives.
3. Automate the prioritisation and execution management of single-task or multi-task SLM processes on behalf of the provider, including assigning SLM tasks to service personnel and/or service agents.

In SAM, the set of service level management related contractual data is referred to as the SLA Data. Associated with this SLA data is a corresponding set of provider facing (i.e., the customer is unaware) SLM Data. For example, a managed storage services contract that offers virtual disk space to a customer with an availability target requires the provider to manage the mapping between the virtual disk space and the corresponding real storage processes. The SLA Data in this case comprises all data attributes associated with the availability of the virtual disk, including required capacity, availability target, pricing and so on. The SLM Data includes the data attributes associated with this mapping (the physical storage, servers' identities, their

allocated capacity, and so on.). While the SLM Data and the SLA-SLM data relationships must be managed by the provider aswell, such non-contractual implementation details need not be exposed to the customer.

SAM defines a high-level information model that facilitates an iterative, top-down approach to capturing and managing the SLA Data based upon the purposes (or the semantics) of the needed contract data elements.

Figure 3-8 presents the principal components in SAM. After an SLA contract is established, the SLA Data are imported into SLA Data storage via an SLA contract processor based upon the Semantic Model of SAM. The SLA Data are then linked with the necessary internal SLM Data vial the SLM data management framework. Every contract is bound with a unique SLA management object (SMO) in SAM, which interacts with the other service management components to manage the execution of the contract (e.g., processing contract-specific on demand resource-provisioning requests).

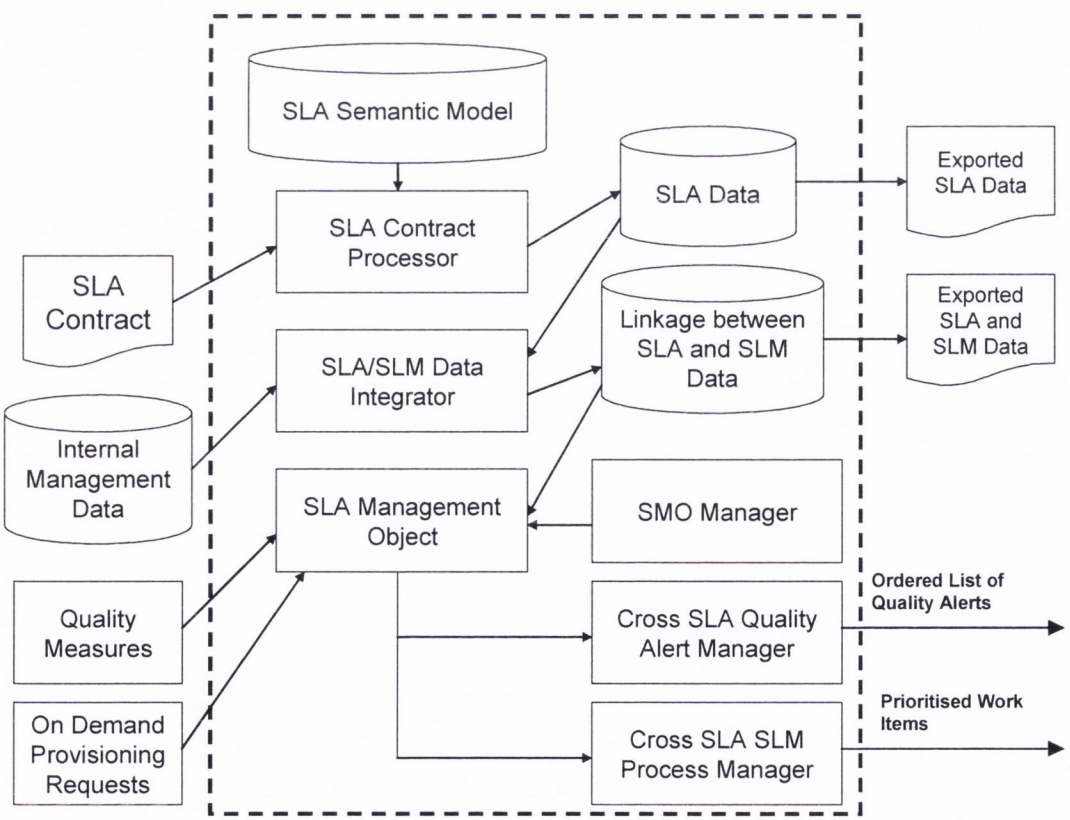


Figure 3-3 SAM Architecture (from [21])

The Cross-SLA Quality Alert Manager gathers quality alert data from all of the SMOs, normalises and aggregates the penalty functions across all contracts, and maintains one or more ordered lists of (active) quality alerts in terms of business impact.

The Cross-SLA SLM Process Manager gathers SLM process execution management requests from all of the SMOs, normalises and aggregates the penalty functions across all contracts, and continually optimises the execution of the SLM process instances based upon the provider's business objectives. The SLM processes are prioritised not only according to the penalty functions but also according to the needed resources for each step of the processes when the resource requirements are available.

When receiving a contract-specific SLM event from the Cross-SLA Event Manager, the SMO processes the event according to its own SLA and SLM Data. If a new quality alert must be generated, the SMO assesses the exposed business impact of the alert on the basis of the contractual terms and the latest quality measurement statistics for the current service level and associates the alert with a penalty function in accordance with the time the alerted condition will be removed. The penalty functions for the alert from all of the affected SMOs are normalised and aggregated by the Cross-SLA Quality Alert Manager.

SAM supports a number of algorithms for prioritising quality alerts. The Earliest Higher Penalty Time-Complete Order (EHPT-CO) algorithm sorts all of the exposed penalty points in chronological order on the basis of their respective penalty time. The Earliest Higher Penalty Time-Partial Order (EHPT-PO) algorithm takes into account only the earliest penalty point of each SLM event. Instead of sorting the penalty points by their respective penalty times, the Higher Penalty Rate First-Partial Order (HPRF-PO) algorithm sorts the earliest penalty point of each quality alert on the basis of the ratio of extra penalty amount and time to the penalty point.

SAM provides a framework that supports SLA management with QoS specification at the application level. The framework provides techniques to aggregate QoS measurement data into QoS metrics specified at the business level. In addition, SAM provides a set of algorithms to minimise penalties based on prioritisation of fault

resolution. However, the SAM cross-SLA manager only addresses penalty minimisation post SLA violation. It does not address service provision optimisation as a mechanism to minimise incurred penalties. Furthermore, SAM does not address the area of dynamic service composition in order to maximise the utilisation of the service provide resources.

3.4 WSLA Framework

The WSLA Management Framework [33] has been developed by IBM to provide differentiated levels of Web services to different customers through the use of automated management and service level agreements. It supports dynamic outsourcing (on demand) where service customers can dynamically form outsourcing relationships with providers, and the execution environment supports the required functionalities for managing the SLA life cycle.

For runtime processing, the contract documents are represented as contract objects. The system provides a set of predefined contract object types as follows:

1. **Basic**. This type is used for storing general purpose contract objects that have no relation to other contract objects, for example, SLA basic templates.
2. **Provider Contract**. This type enables the aggregation of services from service suppliers (for a service provider, its own providers are knows as suppliers). It is possible to create logical units that aggregate all services from the same supplier. Both service supplier and service provider can add services to provider contacts. This contract object type can be extended to cover internal cost.
3. **Offer**. This type is sued by the service provider as a template to create offerings. It can be empty when created and filled in as needed, for example, during contract negotiation. To allow full flexibility, different service operations may be linked to different rating models.

4. **Usage Contract.** This contract object type represents the instance of a consumer contract for a particular service. All further service-related actions are prescribed by this contract. The service operations un the usage contract are linked to the provider contracts that aggregate the service operations.

A WSLA template associated with an offering defines the QoS properties of the service. Conceptually, a template is a WSLA document that contains fields to be filled in during the subscription process. In addition, a template contains a set of constraints on the fields that express the QoS guarantees associated with the service.

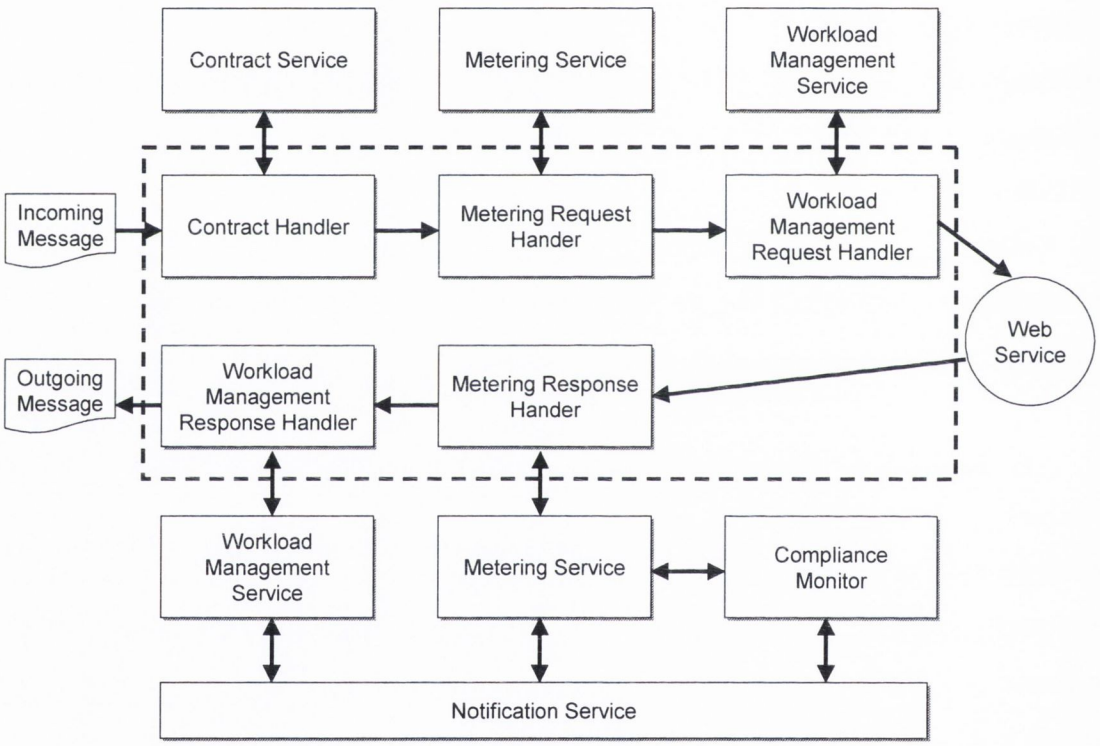


Figure 3-4 WSLA Framework Architecture (from [33])

Figure 3.9 presents the architecture for the WSLA Framework.

The WSLA Framework provides a framework that manages the delivery of Web services to customers based upon the agreed SLA. It supports differentiated levels of service, where by each customer can subscribe to different quality levels of the same service. From the implementation side, the WSLA Framework supports dynamic resource provisioning whereby resources are allocated to customer requests as required. The WSLA Framework does not support true differentiated services however, as this functionality is implemented as a set of request queues – one for each service level – with a weighted round-robin approach to processing the requests currently in the queues. A Global Resource Manager monitors the execution of the queues and periodically adjusts the weightings of the queues based on the current load of the system. Furthermore, the WSLA Framework does not support the concept of service composition for the creation of new, dynamic, services which are an aggregation of existing service provider resources.

3.5 METEOR-S

The METEOR-S Web Service Composition Framework (MWSCF) has been developed as a part of the METEOR-S project at the University of Georgia [4, 23, 98].

The framework is an extension of workflow management developed as a part of the METEOR project [97] to deal with the problems of semantic Web service description, discovery and composition, In particular, the METEOR-S project associates semantics to Web services, covering input/output, functional/operational descriptions, execution and quality, and exploits them in the entire Web process lifecycle encompassing semantic description/annotation, discovery, composition and enactment (choreography and orchestration) of Web services.

The framework has defined Semantic Process Templates which allow for the semantic definition of each activity defined in the process. With the process template, an executable process can be generated with each activity bound to a concrete Web service implementation that conforms to the semantics of the activity. The framework supports the binding of each activity to a WSDL file and a relevant operation in it. Therefore, if the service interface or implementation changes, as long as the URL of the WSDL and the name of the operation do not change, the framework can associate the activity with that operation.

There are four main components to the MWSCF framework, as follows: the Process Builder; the Discovery Infrastructure; XML repositories; and the Process Execution Engine. The process builder constitutes a designer and a process generator. It provides a graphical user interface to design process templates and passes it to the process generator, which uses the discovery infrastructure and data in XML repositories to convert the template into an executable process. The METEOR-S Web Service Discovery Infrastructure is used to access a community of Web service registries and semantically search for Web services. The generated executable process is then executed using a process execution engine.

At present, the MWSCF only supports deployment time binding of activities to Web services, primarily due to the use of a business process engine that only supports deployment time binding [98]. The framework requires that each Web service that is registered with the UDDI is linked to a semantically annotated WSDL description and that the WSDL descriptions contain an *endpoint description* that have QoS details for all the operations in the service. Each of these QoS parameters in the requirements description of an activity is given a weight. For every service that is discovered for an activity, the QoS compatibility is checked and using a weight for each QoS parameter, the ranking value of the service is calculated for the activity.

Through the Web Service Composition Framework, METEOR-S supports dynamic Web service composition through abstract business process definition. Resource allocation takes into account the QoS requirements for each of the activities, as specified in the business process. However, METEOR-S does not address the issue of resource allocation based on the QoS parameters agreed with the service customers,

which is defined as the aggregation of the QoS metrics measured for each resource in the composition. Furthermore, it does not take resource capacity constraints into consideration when allocating resources to multiple concurrent requests from service customers.

3.6 CrossFlow

CrossFlow is a European research project of the 4th ESPRIT Framework that researches cross-organisational workflow support for virtual enterprises [30, 31, 71].

The CrossFlow project has four main aspects, as follows:

Dynamic Service Outsourcing. The co-operation between partners is based on a dynamic outsourcing paradigm with service consumers and service providers. Compatible business partners find each other through a matchmaking facility.

Contract-based service specification. A detailed service specification in the form of a contract is the basis for a tightly-linked co-operation implementing the service provision from service provider to service consumer. The definition of the interaction in the contract is independent of the specific enactment technology of the organisations.

Fine-grained, advanced interaction. The interaction level between service consumer and provider is at a fine grained and a high semantic level, enhanced by the availability of a set of advanced co-operation support services.

Contract-dependent generation of enactment infrastructure. The enactment infrastructure that connects the information systems of service provider and consumer is dynamically set up according to the contract and a specification how the contract is to be implemented and supervised.

The CrossFlow contract model provides the conceptual structure that describes the tight collaboration of service consumer and provider in a virtual organisation. The design of the model includes concepts for representing the structure of the outsourced service process described by the contract, high-level concepts for monitoring and controlling this process in a cross-organisational context, and concepts for flexible use of contracts.

The **Global Contract Model** is composed of the following sub-models:

Concept Model. All concepts that are used in the contract must be defined clearly, creating a concept space in which other contract issues can be specified.

Process Model. The process model describes the internal structure (schedule) of the workflow process implementing the service at the contract level. The process schedule is composed of process elements i.e., the individual activities and transitions. The process is modelled in a way that allows the provider to map it to its internal process and allows the consumer to understand the sequence of events for monitoring purposes and make decisions based on this knowledge for control purposes.

Enactment Model. The enactment model provides concepts to represent the advanced co-operative support that is offered during service enactment. Co-operative enactment support can be composed of a number of elementary services, like service execution monitoring, service execution control, remuneration support, authentication support and so on.

Usage Model. The usage model defines manners in which the contract can be used. The simplest case is where once contract is made to start one instance of the service immediately. Other possibilities are contract made to start multiple executions of the service, or contracts made to reserve the resources of the provider for a service execution at a later moment. The usage model describes the different usage possibilities of the contract and their conditions.

Natural Language Description. The natural language description is a piece of text that is not meant for electronic interpretation, but for human reading. This piece of

text can be used to describe the service in an easily understandable way and to refer to the legal context of the transaction.

The CrossFlow architecture supports both contract making and contract (service) enactment. Figure 3-10 provides an overview of the CrossFlow architecture.

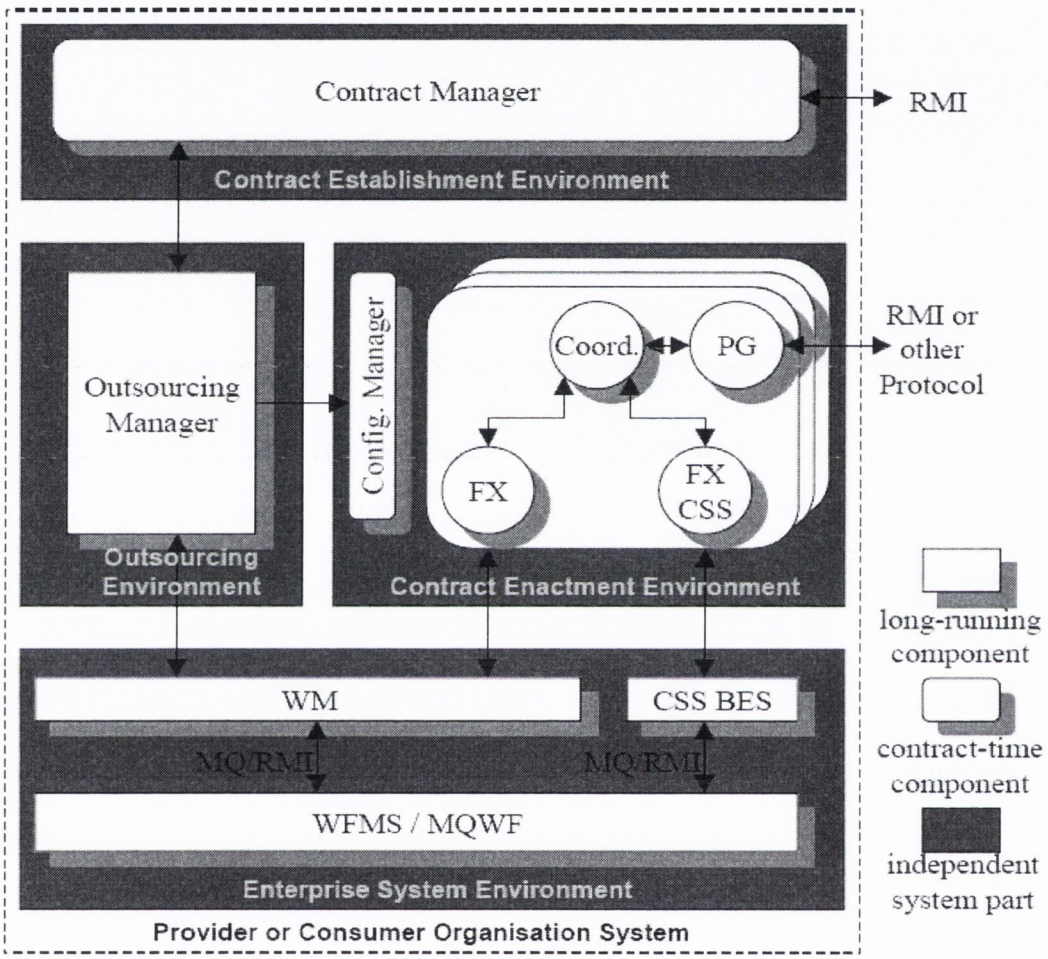


Figure 3-5 CrossFlow Architecture (from [30])

The lifecycle of a service outsourcing consists of the following phases:

Contract Establishment

Advertising and searching for a matching business partner is supported by an advanced service trader, based on the CORBA Trading Service. Unlike the CORBA

trading service however, it supports a bi-directional matchmaking process where both customers and providers can describe what they offer and require of each other.

When the provider is ready to receive requests for enactment of a process, it notifies the Contract Manager of its readiness. The Contract Manager selects a pre-existing Contract Template that describes the services and its associated QoS guarantees, work schedule, monitoring and control points as provided by the service. Appropriate values for these service guarantees including the cost of the service must then be determined. These will be decided according to the capabilities of the enactment infrastructure, the resources that the provider is willing to assign to the enactment, and the price associated with the resources. The service description is then advertised with the trader.

Dynamic Infrastructure Configuration

Once a contract has been made between service consumer and provider, a dynamic contract and service enactment architecture is set up in a symmetrical way for both partners. The Internal Enactment Specification is an organisation-specific blueprint that specifies how the contract is to be enacted. It defines which internal resources can be used in which way. It also provides the mapping between the external and internal details of the business process and its related data.

Contract Enactment

Once the setup has been completed, the consumer can initiate the enactment of the outsourced business process by contacting the provider. Any monitoring information agreed upon in the contract to be provided from the provider to the consumer can either be sent as a notification or requested by the consumer.

Dynamic Infrastructure Disposal

When all the administrative processes have been completed and both sides are satisfied with the provision and consumption of the service, the infrastructure created earlier can be dismantled.

CrossFlow provides a framework that supports cross-organisation business process integration through the use of electronic contracts. The framework exposes a public definition of the service being provided by the service provider and registers this with a service directory (trader), while hiding the implementation details from the service consumer. CrossFlow does not support the dynamic provisioning of resources to service requests as the service implementation is configured on a per-contract basis and resources are committed to electronic contracts at contract establishment and infrastructure configuration stage. CrossFlow provides monitoring of service delivery relative to the QoS guarantees defined in the agreed electronic contract, but it does not provide any management services should the SLA parameters be violated. Therefore, it does not attempt to address optimal service delivery for the service provider, nor does it support highly dynamic services.

3.7 Oceano

Oceano [14] is a prototype of a highly available, scaleable, and manageable infrastructure for an e-business computing utility and enables multiple customers to be hosted on a collection of shared resources. The hosting environment is divided into secure domains, each supporting one customer. These customer domains are dynamic: resource assigned to them may be augmented when load increases and reduced when load dips. This dynamic resource allocation reduces hosting costs while providing a mechanism to guarantee contracted SLAs.

Monitoring agents local to the managed servers collect server load and performance metrics. These agents may monitor system level information or may monitor application or middleware specific metrics, such as a plugin to a web server. The agents issue events when thresholds are exceeded. Events are correlated to identify root causes, which are reported to a central Resource Director. Oceano contains a state based rule engine that correlates network and SLA events. The Resource Director is an event driven system and is responsible for planning resource allocation and recovery actions. The main controls available are load sharing via dynamic allocation and de-allocation of servers, and throttling incoming requests.

Oceano provides a framework for service execution monitoring and management in order to provide service quality levels that comply with contracted SLAs. While Oceano primarily focuses on the system-level resource management, middleware and application level resource management is also supported. However, Oceano does not address the issue of allocating available Web service resources to each customer request in order to meet SLA requirements. Instead the Resource Director component focuses on the allocation of physical hardware in order to best meet customer needs. In addition, the framework does not look to address the creation of service offerings through the composition of existing Web service resources.

3.8 WSAF

WSAF [78, 79] is an agent-based framework for service selection in open environments. The framework uses a policy language to capture service consumer's and provider's profiles, and provides algorithms to select services based on those policies, It also provides the capability to dynamically capture data about service performance with respect to various customisable QoS dimensions. Based on this, service-based applications are dynamically configured at runtime to choose the *best* services with respect to each participant's preferences.

The framework attaches a software agent, an autonomously-configured software component, to each Web service. This agent proxies the service for its consumer, exposing the same interface as the service but additionally enabling other functionality. Instead of directly communicating directly with the service, the service consumers now communicate with the agent. Figure 3-6 presents the architecture of the WSAF Framework. The agents reside in a server that the consumers know about *a priori.* By having the agents on a server, the computational burden of the agent is decoupled from the consumer and also, importantly, the agent can expose a Web service interface to the client.

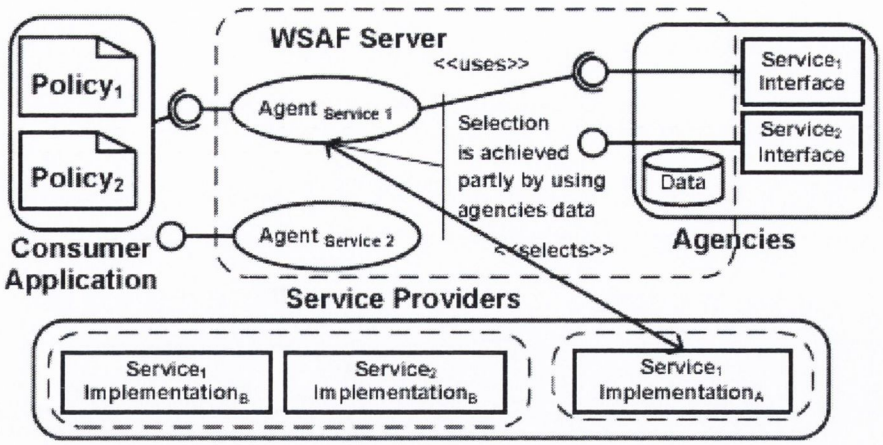


Figure 3-6 WSAF Framework (From [79])

The framework takes a different approach to QoS specification and monitoring than explicit contractual documents such as Web service level agreements (WSLA) [73]. The QoS specification is defined as an ontology that allows the matching of services semantically and dynamically. Service providers have different policies foe each service implementation they publish. The policy captures the provider's advertised QoS policy for a service, indicating the level of commitment of the provider to the advertised policy. Similarly, a customer preference policy specifies a set of services and ontologies and a set of QoS policies.

The WSAF framework defines a matching algorithm that is used to match consumer policies to advertised provider service policies. The service agent automatically configures itself with the correct behaviours based on the QoS ontology so that interactions between the consumer and the service are monitored and recorded in the appropriate agencies.

By using an agent-based approach, the WSAF framework enables a service provider to dynamically assign resources to customer requests in order to meet the QoS requirements of the customer. In addition, the WSAF framework models resource selection and allocation as an optimisation problem. The framework does not attempt to address service composition and the associated aggregation of QoS parameters. Therefore, this makes the framework unsuitable for managing dynamic services that are created through the composition of existing Web service resources.

3.9 Analysis of the State of the Art Technology

Our study of the state-of-the-art has shown that where service composition has been addressed, resources are typically statically assigned to customer requests. Where dynamic resource allocation has been considered the dynamic composition of service offerings and the dynamic aggregation of measurement data both of which ensure that the overall runtime SLA parameters can be met has not been addressed. Also the current state of the art has yet to propose a unified framework that allows cost effective service provisioning based on the runtime requirements of customers using optimisation techniques.

Figure 3-7 shows the features desired to be present in a dynamic service management framework in three different areas: dynamic services, differentiated services and resource management. These features are then compared with the ones provided by the existing state of the art frameworks.

		IBM Service Manager	Conformance	SAM	WSLA Framework	METEOR-S	CrossFlow	Oceano	WSAF
Dynamic Services	Composite Services					✓	✓		
	SLA Management	✓	✓	✓	✓		✓	✓	
	Dynamic Resource Assignment			✓	✓	✓			✓
Differentiated Levels of Service	Customer Specific Service Levels	✓		✓			✓		✓
	Service Customisation	✓	✓				✓		✓
Optimal Provision of Services	Dynamic Resource Provisioning				✓	✓		✓	
	SLA Driven Delivery Optimisation			✓					✓

Figure 3-7 Features Provided By State of the Art Service Management

Several frameworks have been identified, including IBM Service Manager and Conformance, and CrossFlow that provide service management and compliance monitoring according to the SLAs that have been agreed with the service customers. However, the management services provided by these frameworks are for fixed resources that are dedicated to each service customer. They do not provide support for the management and allocation of resources to each customer request, thus making their use for dynamic service management unsuitable.

The issue of service composition has been addressed by a number of frameworks, including CrossFlow and METEOR-S. However, their support is either limited to fixed resources, as in the case of CrossFlow, or allocating resources based on the QoS requirements that have been specified in the business process definition. The frameworks do not support the dynamic allocation of resources based on the composite runtime QoS requirements of each service request.

A number of frameworks, including WSLA, WSAF and METEOR-S address the issue of dynamic resource allocation based on QoS requirements. However, the resource assignment approaches provided by these frameworks only consider the QoS requirements at the resource level, and not provide support for composite QoS parameters that are defined as an aggregation across multiple resources.

The features necessary for providing support for dynamic service management were taken into account of a new architecture, called DSM, which is presented in the next chapter of this thesis.

Chapter 4

A Framework for Dynamic Service Management

This chapter introduces DSM, a framework for the management of services that is designed to address the requirements of supporting dynamic services as outlined in Chapter 1. The approach adopted by DSM results in a very flexible architecture that supports services that are highly dynamic in nature, is capable of supporting customer specific services and service-levels, and manages the provision of those services in an optimal manner.

The chapter begins by describing the main requirements that are imposed on the architecture. The architecture adopted to address these requirements is then described in detail. The approach adopted in the design of DSM results in a very flexible and extensible architecture capable of supporting dynamic, customer specific services atop common utility computing resources.

4.1 Requirements for Dynamic Service Management

This section describes the technical requirements imposed on the DSM framework to enable the framework manage services in a dynamic e-business environment (DeB). These requirements include the following:

- **Dynamic Services.** The framework should be capable of supporting customer requirements that are continuously evolving both in terms of services being provided and the quality levels associated with those services. The service provider must be capable of delivering the required services to each customer.

- **Differentiated Levels of Services.** Individual customers have the ability to not only define the exact service features that they require, but also the service quality levels associated with each of those services. The service provider must be capable of providing the levels of service required in a cost-effective manner.

- **Optimal Provision of Services.** The underlying implementation of provisioned services should be continually managed to ensure the most cost effective implementation for service providers whilst minimising penalties service level violations.

The fulfilment of these requirements represents an important challenge in the development of a framework for managing services in a dynamic e-business environment. Each of these requirements is discussed in detail in the following sections.

4.1.1 Dynamic Services

A key requirement in the management of services in a DeB environment is the ability to support services that are continually evolving in structure and service quality levels [58].

For example, the ability to outsource certain business operations allows each business to focus on its core activities and competencies. In order to meet the requirements of fast changing market conditions (i.e., in order to be effective, efficient, and flexible), it is expected that outsourcing will evolve into "dynamic outsourcing" of applications and business processes, whereby service consumers determine programmatically the service provider and/or service attributes, that is, quality of service, to use [5]. It is expected that service providers will provide a catalogue of services that can be reviewed online [52]. Customers may select a service package with various options such as availability, guarantees, security, coverage, and response time [40]. After the customers review the set of services or service packages, the service provider would verify the order request with aspect to various technical issues. Technical issues include whether the thresholds as specified in the SLA are achievable and the question of whether the required resources for the provision of the requested services are available.

To provide fine-grained outsourcing in a cost effective manner, it is essential to support automated management of the entire life-cycle of the business relationship: creation of service offering, creation of SLAs with possible negotiation, provisioning of applications and environments, and monitoring of SLAs, both for dynamic allocation of resources and for compliance.

Typically, less agile service providers may depend on fixed allocation of resources to customers, whereas a more efficient provider can dynamically redistribute resources across multiple customer workloads. An application service provider [43] involves dedicating resources to each customer service [21]. This implies a fixed cost to the provider for providing and managing these dedicated resources. The cost is typically passed on to the service customer as a recurring charge, regardless of the load on

these resources, and this, the extent to which the customer makes use of the resources. This arrangement is inflexible for the service provider, and, as a result, unduly expensive for the customer [96].

A more efficient service provider can add or remove computing resources from the service dynamically in order to meet the current or anticipated load. Dynamic computing services are therefore provided on demand. The contract between the customer and provider defines charges for services at a relatively file grain. The terms and conditions of the delivey of the service are defined in a contract, including the definition of the service, the rating model (specifying how the service is priced), and the performance guarantees on various aspects of the service.

4.1.2 Differentiated Levels of Service

Typically, services providers have offered few customer-neutral service functions atop a common service delivery infrastructure. This exploits economy of scale better than pursuing a high degree of customisation of its service functions for every potential customer [22]. A credible study on a leading IT service provider's SLA reporting cost, for example, has shown that several millions of dollars could be saved annually by reducing the cost of generating the monthly reports for 100 high-valued customer-specific SLA contracts by no more than 20 percent [22]. Thus, it is important for a successful service provider to be able to satisfy its customer's demand for customer-oriented IT outsourcing functions with high-quality services and to fulfil all of its SLA commitments based upon business objectives.

However, this customer-neutral approach for establishing SLAs seems effective only for primitive e-business services. The other classes of e-business SLA contracts normally require non-trivial customer-specific customisation of the provider's service offerings to accommodate a customer's unique data or business process management needs. When the number of such customer-specific SLA contracts grows, the difficulty of cost-effectively meeting service level management objectives increases.

One of the most pressing challenges facing a service provider, therefore, is to manage the execution of all of its SLA contracts in business terms (e.g., minimising financial penalties for service-level violations, maximising service level measurement based customer satisfaction metrics) [95].

Where a customer has had the choice on service level quality, this has typically been at fixed quality levels – e.g., low / medium / high quality. Existing frameworks do not address the requirement for each customer to specify unique QoS parameters in the agreed SLA.

4.1.3 Optimal Provision of Services

A service provider may become more cost-effective by exploiting economics of scale similar to traditional public utilities. Hence the service providers can be referred to as "utility computing providers" [33]. The way computing utilities are contracted is fundamentally different from application hosting as practised in recent years [22]. An application hosting service involves dedicating resources to that service. This implies a fixed cost to the provider for providing and managing these dedicated resources. The cost is typically passed on to the service customer as a recurring charge, regardless of the load on these resources and, this, the extent to which the customer makes use of these resources. The arrangement is inflexible for the service provider and, as a result, unduly expensive for the customer.

Consider a public utility model for service delivery, such as electric power. Electricity is always available when needed and customers are billed only for the power they actually consume. SLAs in utility contracts dictate the expected reliability of the power delivery and the maximum power a customer may draw. The customer load varies in time, so the utility service provider is able to allocate power dynamically to minimise costs and meet the SLAs [33]. A utility computing provider can operate using the same principle. Computing resources can be added or removed from the service in order to meet current or anticipated load. Utility computing services are therefore provided on demand, following public service utility model [21]. The

contract between the customer and provider defines charges for services at a relatively fine grain.

In order to optimise on service delivery costs, the service provider may not allocate all required resources in advance. Instead, it may dynamically allocate resources depending on customer usage levels. The customer load varies over time, so the service provider is able to allocate resources dynamically to minimise costs and meet the defined SLA requirements.

SLA-driven automatic and autonomic provisioning will allow the service provider to allocate resources where and when they are needed, minimising delivery costs.

The service provider should be capable of supporting a number of different types of service provisioning, depending on infrastructure capability, types of services, and business models. The objective of provisioning is to allocate sufficient resources in order to avoid violations of service level guarantees. The main resource provisioning scenarios are [94]:

- **Dedicated resource provisioning.** Provisioning dedicated resources allocated to a single consumer. When a subscription request is received, resources are provisioned considering the service specifications versus the anticipated load.
- **Per-SLA virtualised resource provisioning.** In an environment in which consumed resources area virtualised (and underlying physical resources are shared in support of multiple SLAs), the provisioning system evaluates whether an additional SLA can be supported with the resources available to this virtualised system. The system may also add new resources to this shared pool based on the aggregate anticipated load of all supported SLAs.
- **Dynamic resource provisioning.** In this, the most sophisticated on-demand environment, resources are allocated to services as needed. An initial set of resources is provisioned for a service, and the SLA system monitors the performance of that service. When the load changes, new resources are allocated or de-allocated dynamically, in order to minimise the service delivery costs while meeting SLA objectives.

Resource allocation may not be for purely technical reasons. Resource allocation also takes into account various business objectives, such as profit optimisation and customer satisfaction (e.g., no more than a certain number of violations during a given period, independent of penalties) and this must be taken into account by the service management framework [22].

4.2 DSM Architecture

This section describes DSM, the service management architecture proposed in this thesis. The requirements detailed in the previous section dictate a service management framework that supports services that are highly dynamic in nature, is capable of supporting differentiated levels of service for each customer which is constructed on tope of common utility computing resources, and manages the runtime execution of those services in an optimal manner. The following sections detail the main components that form the basis of a framework to support these requirements. While the presence of each component is explained in its own right, a determination of whether the framework meets the requirements imposed in the previous section will be presented in Chapter 6, in the context of an overall architectural evaluation.

4.2.1 Overview

The DSM framework is designed to be situated within a service provider environment where it manages the fulfilment and optimisation of services to the service provider customers. DSM enables dynamic outsourcing (on demand), where service customers can dynamically form outsourcing relationships with the service provider as required. It enables the service provider to offer the same service at different quality levels to each customer, depending on customer requirements. Finally, it manages the provision of these services in an optimal manner so as to maximise compliance with the agreed-upon service levels whilst minimising the cost of implementing the service.

Figure 4-1 illustrates a high-level architecture of the implementation of DSM, in which functions are grouped into three key components: SLA Manager, Web Service Execution Environment and Resource Broker.

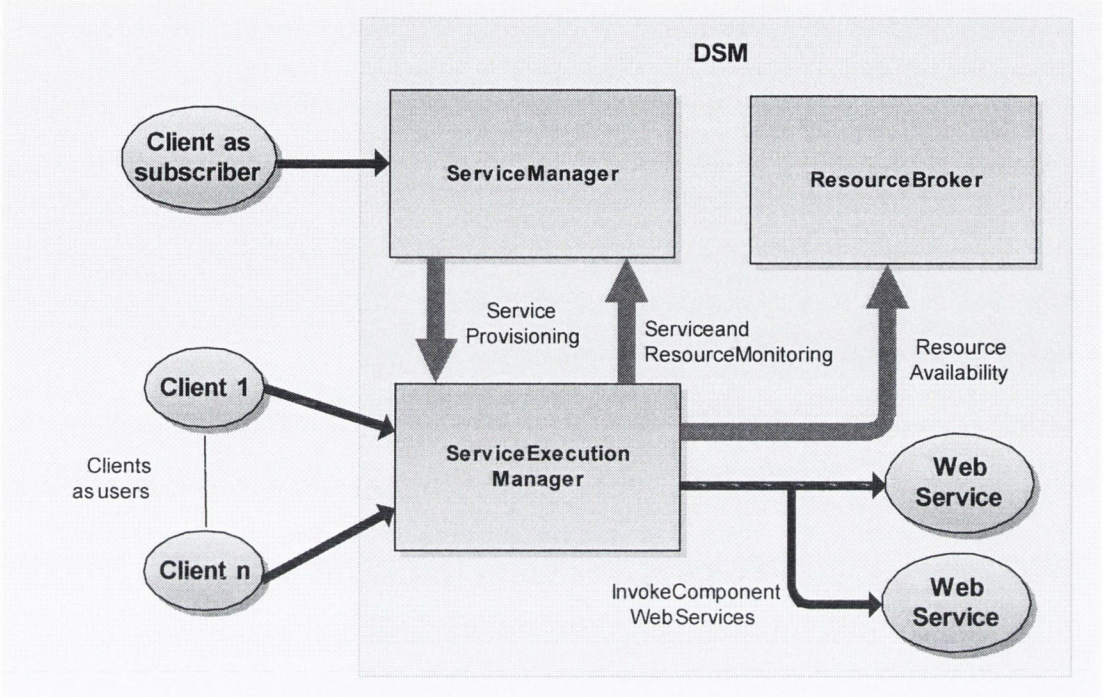


Figure 4-1 DSM Architecture

All customer interactions other than the service invocation itself are managed by the SLA Manager component. A customer order for a service offering, referred to as a service subscription in DSM, results in the creation of an SLA instance as a part of a contractual agreement. The subscription is tracked during the fulfilment process to make sure that the service level guarantees agreed upon in the SLA are adhered to.

The Service Manager is responsible for the execution of individual instances of a composite web service that has been requested by a customer. The Service Manager receives requests by customers for a service. It authenticates the requester and validates the details of the SLA associated with the service request before scheduling the execution of the request.

The Service Broker is responsible for maintaining the SLAs and other information associated with component services. These include both services that are implemented directly by the service provider and those that are outsourced to other providers (suppliers).

The remainder of this chapter will explain each of these components in detail, illustrating how they fit together to create the full architecture. As the discussion continues, it will be shown how the architecture provides management services which satisfy the unique set of requirements outlined in the previous sections. During the discussion of each individual component, the manner in which that component fulfils these needs will be highlighted.

4.3 Service Manager

This section presents the Service Manager, the component of DSM which manages all relevant aspects of services offered by the service provider and their associated SLAs.

4.3.1 Service Offering

The service provider creates service offerings which include both a description of the service and the QoS levels associated with the service.

Service Definition

In order to be able to manage services in an optimal manner, DSM provides a level of abstraction between the service that is offered to clients and its implementation.

The services that are implemented by the service provider are treated as utility computing resources, which can be dynamically assigned to customers as required. Each service resource is exposed as a Web service with a WSDL interface. As a customer invokes the service offering, DSM dynamically allocates service resources

to the fulfilment solution. This approach allows the service provider to easily create new service offerings on top of their existing service components.

The service provider creates a service offering by firstly providing a description of the service functionality to be offered. This is published to customers as a Web Service Description Language (WSDL) interface and can also be registered with a service directory, such as UDDI. The service implementation is defined in abstract terms, whereby the resources implementing the service are described only in terms of the required functionality, and not the actual resources implementing the service. This level of abstraction allows for a late binding of resources to service request from a customer and ensures that that service resources can be allocated to customer requests in an optimal manner.

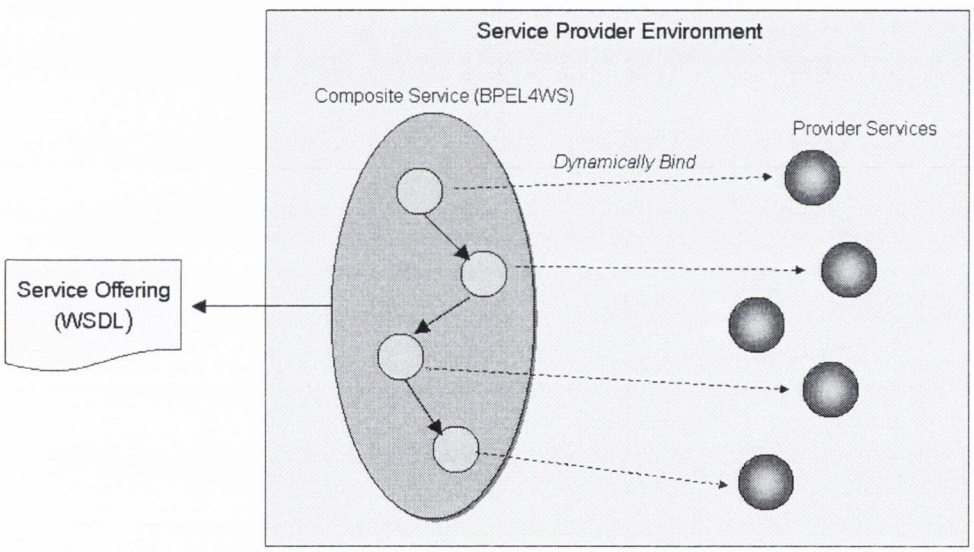


Figure 4-2 Service Definition

The service definition described above provides service template where each activity in the process definition refers only to the functional requirements for that activity and not an actual service implementation. During the service execution as the business process is carried out resources are dynamically bound to the process instance. The actual resources are selected by the Service Execution Manager. This is discussed in detail in section 4.3.

Service Level Agreement Template

Associated with each service offering is a Service Level Agreement template. DSM supports the following QoS parameters in an SLA, however, the architecture is extensible and further QoS metrics can be added by the service provider should these be required. Every SLA parameter refers to one QoS metric, which in turn may be an aggregate of other QoS metrics using an application specific aggregation algorithm.

QoS parameters supported include:

1. **Time**. This is a common and universal measure of performance. For composite services which are constructed as business processes it can be defined as the total time needed by an instance to execute.

2. **Cost**. This is defined as the amount of money the service client has to pay for the service.

3. **Reliability.** The reliability of a service is a probability that a request is correctly responded within the maximum expected time frame. The value of Reliability is a computation from historical data about past invocations of the service.

4. **Availability.** The availability of a service is a measure of the amount of time the service is available over a defined period.

5. **Reputation.** The reputation of a service is a measure of its trustworthiness. It mainly depends on end user's experiences of using the service.

DSM enables the service provider to associate these quality parameters with composite service offerings that are offered to customers. This allows the service provider to define quality parameters in terms of business objectives that are aggregated over time and also define application-specific *composite metrics* that are computed as the aggregation of a number of resource-level QoS measurements using an application-specific *aggregation algorithm.*

Finally, ramifications, in terms of financial penalties, of not meeting the quality standards are also included in the SLA. The overall penalties incurred for not meeting the guarantees defined in the SLA is computed as an accumulation of each of the individual penalties that are incurred. The SLA also captures the concept of how the SLA targets can change over the course of provisioning the service and how the associated penalties change. This is discussed in section 4.3.3.

4.3.2 SLA Manager

The SLA Manager is responsible for maintaining SLA's with each customer. It supports the following processes across the lifetime of a service relationship. This ensures that the service is rapidly provisioned by the Service Manager.

Creating an Offering

When creating an offering, the service provider creates a service template that incorporates functionality from its portfolio of service resources which is defined as a business process template.

Associated with the service offering is an SLA template which outlines the service specific QoS parameters and service levels that can be provided with the service. The service offering defines the functional requirements for how the service offering is to be fulfilled and a business process definition for how the service resources are to be orchestrated to implement the service offering,

Once the offering has been created by the service provider it is made available to customers, typically by publishing in a service directory.

Subscribing to an Offering

Customers agree on the terms and conditions defined in an offering by subscribing to it. The subscription process results in the creation of a usage contract that represents

the subscription, and it is used for all further processes as the context to the contractual relationship.

In the course of the subscription process, a customer retrieves the QoS metrics offered by the provider, aggregates and combines them into various SLA parameters, defines the service levels for every SLA parameter, and submits the SLA to the service provider for approval. A typical example on the customer side is to define thresholds for response times or throughput according to the price the customer is willing to pay. On the provider side, typical business actions are to decide whether the SLA is acceptable as a whole or whether the customer-specified thresholds are too restrictive.

In the main, the usage contract incorporates the properties of the offering.

Provisioning the Service Instance

The following three provisioning models are supported by DSM. Whilst the optimal approach is to only allocate resources dynamically to each service requests, there may be a requirement to either allocate a specific resource to an individual customer, or set of customers, which is done when the service is initially provisioned.

1. **Dedicated Resource Provisioning.** In this category dedicated resources are allocated to a single consumer.

2. **Per-Service Resource Provisioning.** In this category, resources are virtualised, and shared across multiple services. However, resources are allocated to each service when the service is provisioned.

3. **Dynamic Resource Provisioning.** In dynamic resource provisioning, resources are not allocated to a service in advance of the service execution. Resources are allocated on demand during the execution of the service.

4.3.3 SLA Object

The SLA Object (SLO) is an information model in DSM that captures and maintains service level management data based on the requirements of the contract between the customer and the provider.

Once a customer has subscribed to a service, an SLO is created which is used to represent the QoS parameters of the service delivery and monitor compliance with the agreed SLA parameters.

Figure 4.3 presents an outline of the SLO information model in DSM which describes the relationships between the elements of the SLO information model in the form of a UML diagram.

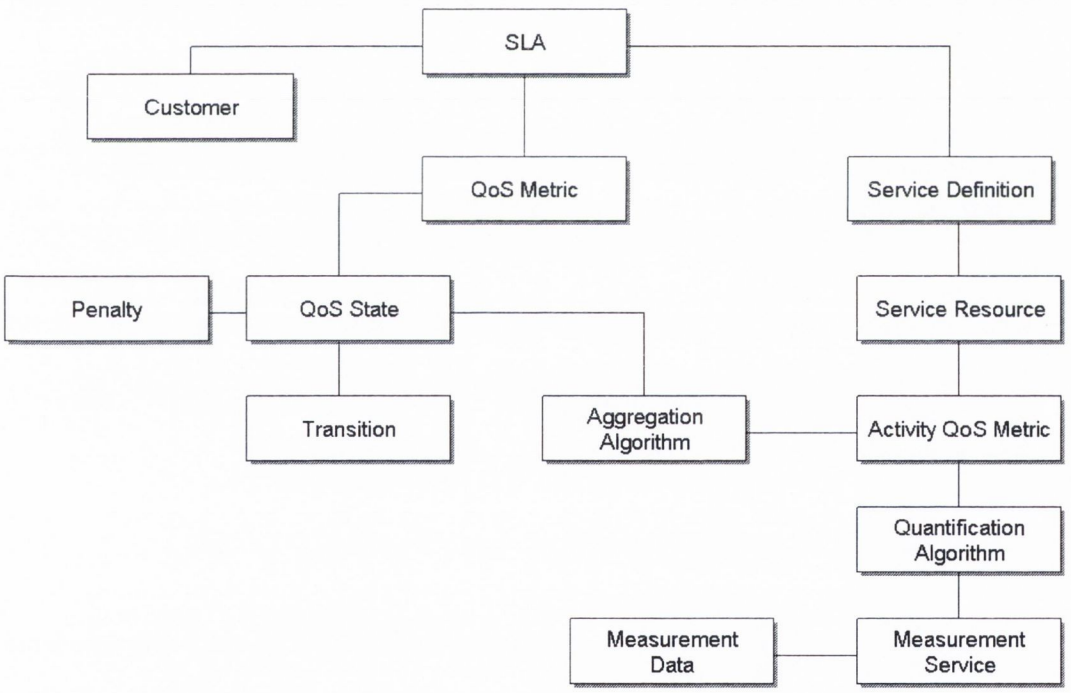


Figure 4-3 SLO Information Model

In the above information model, an SLA defines a series of QoS Metrics and a Service Definition. Each QoS Metric is defined according a number of States which

enables the SLO model to capture how the QoS Metric can change over time (For example, an SLA may state that failure to meet the QoS target incurs a penalty of 20% of the monthly service cost. Failure to meet the QoS target more than 3 times in any one month will subsequently incur a penalty of 25% of the monthly service cost.). The Transition associated with each QoS State defines the rules as agreed between the customer and the service provider that cause a change of state of the QoS Metric. Associated with each QoS state is a Penalty. Each QoS metric will contain at least one state where the QoS Metric is in compliance with the agreed SLA. In this case, the Penalty is zero. The Service Definition is composed of a set of Service Resources. The Service Resources define the functional and non-functional QoS Requirements for each of the component resources that comprise the service offering. Associated with each Activity QoS Metric is a definition of the measurement data that is required to evaluate this QoS Metric. The Aggregation Algorithm defines how each Activity QoS Metric is aggregated as a part of the SLA QoS Metrics.

4.4 Service Execution Manager

The Service Execution Manager is responsible for handling individual service requests and providing a fulfilment solution for that request.

The Service Execution Manager evaluates the service request in terms of the following contexts in order to allocate resources and schedule the execution of the service process:

1. **Current service request**. The Service Execution Manager evaluates the QoS parameters required for each service fulfilment in order to allocate resources to the request.
2. **Service requests across lifetime of the customer SLA**. The QoS parameters specified in the SLA typically span multiple service requests and the current status of these QoS parameters need to be evaluated when deciding resource requirements for this service request.

3. **Current workload in the service provider.** The service provider will typically be handling requests from multiple customers simultaneously, and limited resources need to be allocated in an optimal manner across all executing processes.

The Service Execution Manager provides a two step process when allocating resources to a service request: *Execution Planning*; and *Service Execution Scheduling*. These two steps are discussed in sections 4.4.1 and 4.4.2.

As a process instance executes, the Service Execution Manager determines the most likely execution path of the process, and as such, which activities, or service resources, are most likely going to execute. Using this information, the Service Execution Manager is able to predict the likely outcome for each of the QoS parameters over the execution of the entire process. This information is used to evaluate the QoS requirements of each of the activity resources as the process executes.

Given the QoS requirements for each of the activities in the process, the Service Scheduler best allocates resources to service requests in order to minimise the cost of providing the service.

These two stages are discussed in the following sections.

4.4.1 Execution Planner

In order to calculate the QoS requirements for each activity in the process, DSM estimates the most likely execution path of the process. The Execution Planner determines the most likely execution path and from that, which activities in the process are most likely to execute.

During the execution of a business process, the process composition can be described by the following patterns:

1. **Probabilistic Invocation**: A probability value (weighting) on an outgoing arrow from A to B indicates that A invokes B with that probability.

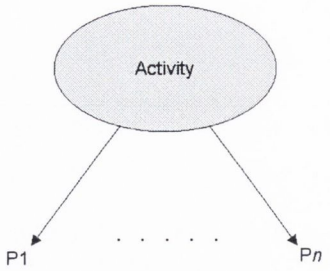


Figure 4-4 Probabilistic Invocation

2. **Parallel Invocation (fork):** A Web service can require that a set of its successors are invoked in parallel.

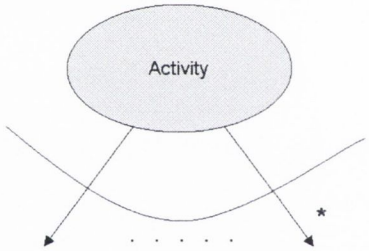


Figure 4-5 Parallel Invocation

3. **Sequential Activation:** A Web service is activated as a result of the completion of one of the set of mutually exclusive predecessor Web services.

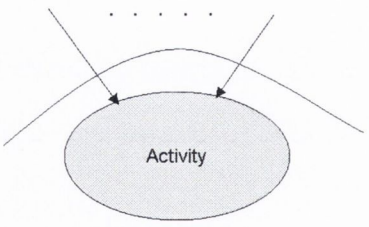


Figure 4-6 Sequential Activation.

4. **Fastest-processor-triggered activation:** The first predecessor to complete activates a Web service.

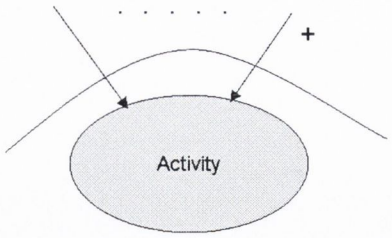


Figure 4-7 Fastest processor triggered activation

5. **Synchronised Activation:** A Web service is activated only when all of its predecessor Web services have completed.

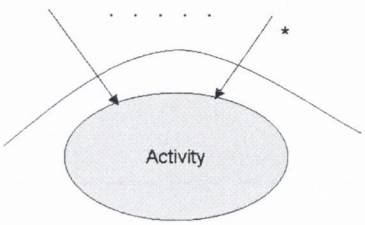


Figure 4-8 Synchronise Activation

Figure 4-9 illustrates an example of a service composition that defines weightings on the conditional branches. The transitions for Web service B to Web services C and D do not define probability weightings, so both C and D execute in parallel. In the transition from Web service C to Web services E and F, the transition to E has a weighting of P1 and the transition to F has a weighting of P2, with P1 + P2 = 1. Finally service G is activated when service D completes and when either E or F completes, depending on which path the execution took.

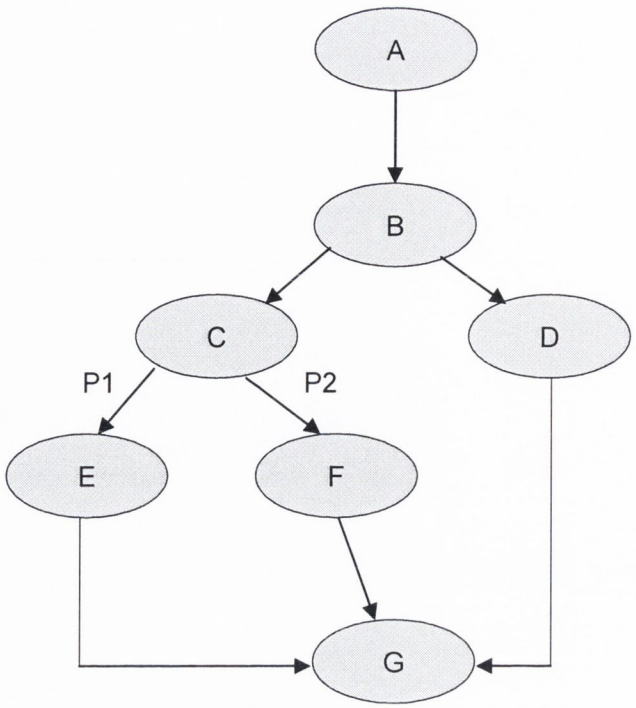


Figure 4-9 Process Weightings

For the execution of this process instance, the QoS metrics that are calculated as an aggregation for each resource that comprises the service composition, namely, the *time* and the *cost* attributes, need to be considered.

The time metric is calculated as the total time to execute each of the resources in the composite service. For Figure 4-9, this is calculated as:

$$T = t_A + t_B + \max(t_D, t_C + P1t_E + P2t_F) + t_G$$

The maximum function is used as both the transitions from B to C and B to D execute in parallel. Therefore, to calculate the overall time for the execution of the process, the greater of the two execution times for the two parallel branches should be taken.

The cost metric is calculated as the sum of the costs of each of the Web services that executes. For Figure 4-9 this is calculated as:

$$C = c_A + c_B + c_C + P1c_E + P2c_F + c_D + c_G$$

In the above scenarios, weightings are associated with the probabilistic invocations. Initial values for these weightings are defined by the service provider at service creation time, but Service Execution Manager is capable of adjusting the weightings over time based on previous executions of the service.

4.4.2 Service Scheduler

The output of the Service Planner is a set of Activities, or jobs, that need to be executed for a service request. This is a probabilistic set of jobs based on the probability weightings at each predicate in the process graph and are likely to change as the process actually executes.

The Service Scheduler uses this jobs list as the basis for scheduling the execution of the service instance. Should the execution flow change during execution, the Service Scheduler can accommodate the change and re-estimate the schedule. This is discussed in the following section.

The Service Scheduler is a cross-SLA execution optimiser, and given a set of number of service requests concurrently executing, with each request consisting of a set of jobs, the Service Scheduler is responsible for allocating resources to the jobs so as to fulfil the service requests. This resource assignment problem can be classed as an extension of the task-processor problem [93], which is known to be an NP-Hard problem [48]. However, DSM utilises optimisation techniques to provide a solution to this problem.

The scheduling algorithm implemented in DSM uses the Branch-and-Bound search algorithm [57] to optimise the allocation of service resources to jobs. The Branch and Bound algorithm derives models with reduced bounds that are repeatedly solved until an overall solution is found. In Branch-and-Bound, a set of enumerative methods are applied to solve discrete optimisation problems [57]. The original problem, referred to as a "root problem" is bounded from above and below. If the bounds match, a solution has been found. Otherwise, the feasible region is partitioned into sub-regions. The

sub-regions constitute feasible regions for sub-problems, which become children of the root problem in the search tree. Any node of the search tree with a solution that exceeds the global upper bound can be removed from consideration. As the tree is searched, all nodes are either removed or solved. A full discussion of the Branch and Bound algorithm is provided in Appendix C.

An initial solution to the problem, called the *incumbent*, is found quickly using a "greedy search" approach. Whilst this may not be an optimal solution to the problem, it sets an upper bound when traversing the search tree. If the value of a node exceeds the upper bound defined by the incumbent, it is clear that all child nodes of this node will exceed the upper bound, and as such, the solution will not be "as good" as the incumbent. Therefore, this node can be pruned from the search tree.

The root node of the search tree is where no jobs are assigned to any resources. Each subsequent child node in the search tree represents the assignment of a resource to a service job.

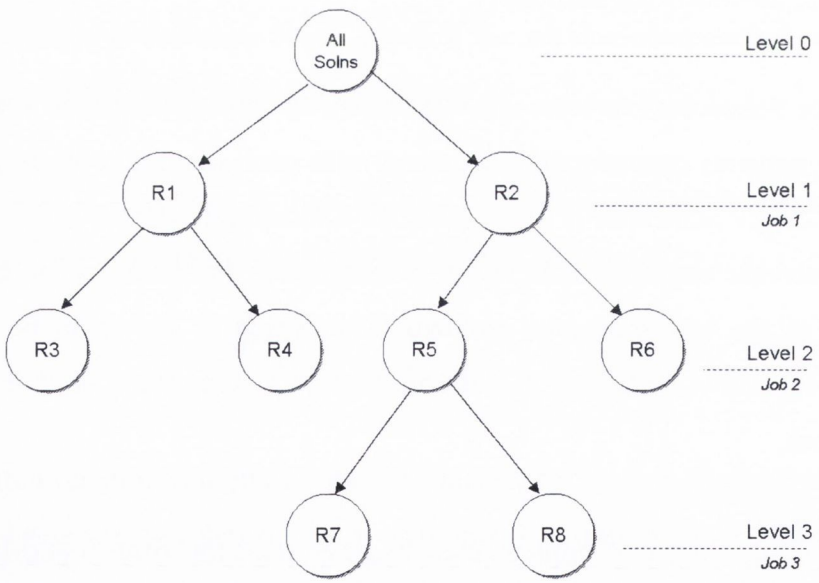


Figure 4-10 Scheduler Search Tree

As the search tree is traversed, the score at the best leaf found is stored and its score is set to be the new lower bound, B. Whenever a node is reached whose score is worse than B, the tree is pruned at that node, i.e., its sub-tree will not be searched since it is

guaranteed not to contain a leaf with a score better than B. When the node is pruned, backtracking occurs. This process determines which node to explore next if the current node is pruned. This process will systematically enumerate all remaining live nodes in the search tree until a global optimal is found. Using this approach, Branch and Bound ensures that an optimal solution can be found to the problem without having to exhaustively traverse the entire search tree.

The search is terminated when it is found that there are no further nodes can be expanded – i.e., all nodes that have not been pruned are leaf nodes.

The Scheduler maintains a list of unassigned jobs. While there are still unassigned jobs, the scheduler chooses the job with the earliest start time first and looks to assign a resource to that job.

In order to calculate the value for each node and determine whether it should be pruned or explored further, the Scheduler utilises the SLO object associated with that service request. The Scheduler obtains the SLA parameters associated with the resource and utilises the service-specific aggregation algorithms defined in the SLO, along with the penalty implications to obtain a cost value for that node.

4.5 Resource Broker

The Resource Broker is responsible for maintaining information about the *Service Resources* that are implemented by the service provider. The Service Resources are exposed as a Web service interface which can be aggregated into composite services by the service provider as a part of a service offering. The Resource Broker provides a directory service which enables the Execution Manager to locate resources when allocating resources to customer requests. The Resource Broker also provides measurement services, enabling DSM to capture QoS measurement data from each resource as it executes.

4.5.1 Resource Availability

The Resource Broker maintains a directory of resource services that is available for allocation to customer requests. This is queried by the Service Scheduler to obtain candidate resources to assign to customer service invocation. Resources are registered with the directory along with a description of the resource capabilities, for example, the estimated execution time and cost of invoking the resource. This enables the Service Scheduler to calculate the impact each resource will have on the overall aggregate SLA parameters.

4.5.2 Monitoring and Measurement

The Resource Broker monitors service execution and delivery with respect to the QoS parameters for each of the service resources. While the Service Manager monitors and aggregates SLA data in order to determine compliance with customer SLAs, the Resource Broker monitors the QoS parameters associated with the execution of each service resource. This is an important requirement to be able to identify root-cause problems with a service delivery over time. Furthermore, should the service resource be further outsourced to a supplier, the Service Broker can quickly calculate penalties to be imposed on the supplier provider.

4.6 Summary

This Chapter has described in detail the main contributions provided by this Theses, which is the DSM dynamic service management framework. In particular the DSM framework provides:

1. Support for services that are provisioned dynamically.
2. Support for differentiated levels of service.
3. Provisioning of the service to the customer in an optimal manner that is most cost-effective to the service provider.

After listing the requirements imposed on the architecture, the chapter has introduced the component-based structure of the architecture. The role of each component in the overall task of providing mechanisms for dynamic service management has been established.

The assessment of whether a service management system built according to these principles meets the requirements imposed is presented in chapter 6, in the context of an overall architectural evaluation.

The next Chapter details the design and implementation of a framework which was used to realise the architecture.

Chapter 5

Framework Design and Implementation

This chapter presents a detailed description of the design and implementation of the DSM framework that has been implemented as part of this thesis. This prototype implements each of the concepts and algorithms that were described in Chapter 4.

This remainder of this chapter is organised as follows: An overview of the system architecture of DSM is presented in section 5.1. Section 5.2 presents a detailed description of the design and implementation of DSM. Section 5.3 discusses the integration with the business process engine, which is responsible for the execution of the composite web services. This is followed in section 5.4 with a description of the implementation environment and technologies for DSM. Section 5.5 discusses the issues that arose during the design and implementation of DSM. Finally, the chapter is summarised in section 5.6.

5.1 DSM System Architecture Overview

This section describes the main components of DSM and the functionality provided by each of these components. The DSM framework is designed to be located within the service provider environment where it manages the interaction with each of the customers and the fulfilment of the services to the customer.

Figure 5-1 presents an overview of the system architecture of DSM. There are three main components in DSM: the Service Manager, the Service Execution Manager and the Resource Broker. The following presents a description of each of these components.

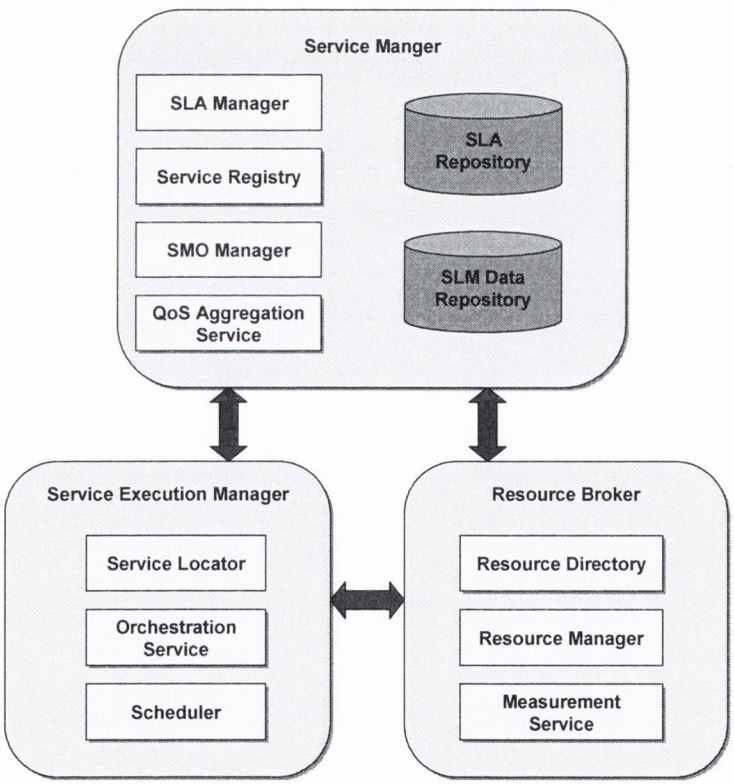


Figure 5-1 DSM System Architecture

Service Manager

The DSM Service Manager is responsible for maintaining the services that are defined by the service provider, along with the subscriptions to those services by each

customer. Service definitions are registered with the Service Registry component, which provides a directory service for customers to locate services. The service definition comprises a Web Services Definition Language (WDSL) document describing the functionality provided by the service. Associated with each service definition is an SLA template, which is structured as a partially completed WSLA document. A subscription to a service results in the creation of a WSLA between the customer and the service provider, which specifies the exact SLA parameters associated with the service provision. This results in the creation of a Service Management Component (SMC) internally in the service provider, which is used for the management of the provision of the service to the customer. The QoS Aggregation Service is responsible for the aggregation of resource-level QoS metrics received from the Resource Broker.

Service Execution Manager

The DSM Service Execution Manager is responsible for handling requests from customers for services and the fulfilment of those requests. The Orchestration Service component receives the customer request and locates the SMC object associated with this service request from the Service Manager. The SMC contains a definition of the service composition, which is passed to the Execution Planner, which in turn produces an execution estimate for the request. The Scheduler takes the execution plans for each of the currently executing service requests and allocates component resource instances to each of the activities, using the SLA Parameters defined in each associated SMC to determine the resource allocation with the minimum cost implications. The Orchestration Service invokes the process execution using the Process Engine Service Provider Interface (SPI).

The Orchestration Service receives execution status events from the SPI as each process instance executes. If the actual execution path of a process instance deviates from the planned execution path, the Orchestration Service invokes the Execution Planner to re-estimate the execution path of the process. The scheduler subsequently re-generates a revised schedule for the new list of schedulable activities. As the process instance is executed, the SPI sends a request for a resource allocation to an activity definition to the Queue Manager. The requests are stored in a queue buffer

until the resource allocated to the request by the Schedule becomes available for execution.

Resource Broker

The DSM Resource Broker is responsible for the interaction with each of the resource Web services which implement the functionality required by the composite services. These resources are the applications and services that are implemented by the service provider which are exposed as a Web service interface. The Resource Directory provides a directory listing of the services which is utilised by the scheduler to locate services requested by the composite web service definition. The Measurement Service monitors the execution of individual Web services and relays back measurement information to the SMC associated with the service request. The Measurement Service uses Measurement Adapters which monitor for specific metrics.

5.2 Design and Implementation of DSM

This section presents a detailed description of the design and implementation of each of the components of DSM.

5.2.1 Service Manager

The Service Manager is responsible for publishing the services offered by the service provider and providing the mechanisms for a customer to subscribe to a service offering and agree an associated SLA with the service provider. This includes the definition of the service offering and an associated SLA template, and a component model for managing the service fulfilment.

5.2.1.0 Service Definition

A service is created by the service provider through the definition of a business process orchestration of the Web service resources that are implemented by that service provider. The service orchestration is defined using the Business Process Execution Language notation for Web Services (BPEL4WS) [54] for the specification of the business process.

BPEL4WS utilises the concept of partners to define the different parties that interact with the business process in the course of executing the process [65]. Each partner link is characterised by a partner link type [65]. This information identifies the functionality that must be provided by the partner service [68], i.e., the WSDL `portTypes` that are required by the business process. Each partner definition defined in abstract terms and the actual partner service is determined dynamically within the process. BPEL4WS uses a notion of endpoint reference to represent the dynamic data required to describe a partner service endpoint [53]. Each activity in the process refers to the definition of a partner as an implementer of certain WSDL `portTypes` and the WSDL operation being targeted [54]. The partner link defines the operations and message types that make up the interface to the service using `portTypes` in WSDL.

The service definition utilises the proposed WS-Addressing specification [112] which defines an endpoint reference [53] that allows for one of a number of available services to be dynamically bound as the process executes.

The partner for each of the service activities is defined in the BPEL definition is defined as illustrated in Figure 5-2.

```
<partnerLink name="LoanService"
      partnerLinkType="services:LoanService"
myRole="LoanServiceRequester"
partnerRole="LoanServiceProvider"/>
```

Figure 5-2 Defining a reference to a service resource

In this example, the service `LoanService` is an abstract service that is bound dynamically by the DSM Execution Manager to a concrete service implementation at runtime.

The `partnerLink` is then invoked by an activity definition in the process using the invoke construct, as outlined in Figure 5-3.

```
<invoke partnerLink="ncname" portType="qname" operation="ncname"
inputVariable="ncname"? outputVariable="ncname"?
standard-attributes>
```

Figure 5-3 Invoking a service resource in a business process

The process definition defines weightings at each point in the process where conditional branching may occur. This is utilised by the DSM Execution Planner to estimate the most likely execution path of the process. DSM extends the definition of the following structured activities of the BPEL4WS language.

Switch Activity

The Switch Activity supports conditional behaviour that consists of an ordered list of one or more conditional branches. The first branch whose defined condition is met is selected and provides the activity performed for the switch [54]. Figure 5-4 illustrates the definition of the Switch Activity which has been extended to incorporate a weighting on the condition.

```
<switch standard-attributes>
standard-elements
<case condition="bool-expr" weight="integer">+
activity
</case>
<otherwise>?
activity
</otherwise>
</switch>
```

Figure 5-4 Switch Condition with Weighting

While Activity

The while activity supports repeated execution of a specified iterative activity. The iterative activity is performed until the given Boolean no longer holds true [54]. Figure 5-5 describes the While Activity definition which has been extended to incorporate the weighting of the condition.

```
<while condition="bool-expr" standard-attributes
      iteration-count="integer" weight="integer">
standard-elements
activity
</while>
```

Figure 5-5 While Condition with Weighting

Pick Activity

The `pick` activity awaits the occurrence of one of a set of events and then performs the activity associated with the event that occurred. The occurrence of the events is often mutually exclusive (the process will either receive an acceptance message or a rejection message, but not both). If more than one of the events occurs, then the selection of the activity to perform depends on which event occurred first [54]. Figure 5-6 illustrates the specification of the Pick Activity which has been extended to incorporate the concept of a weighting on the condition.

```
<pick createInstance="yes|no"? standard-attributes>
standard-elements
<onMessage partnerLink="ncname" portType="qname"
operation="ncname" variable="ncname"?>+
<correlations>?
<correlation set="ncname" initiate="yes|no"? weight="weighting">+
</correlations>
activity
</onMessage>
<onAlarm (for="duration-expr" | until="deadline-expr")>*
activity
</onAlarm>
</pick>
```

Figure 5-6 Pick Condition with Weighting

The public interface to the service is defined using the Web Service Description Language (WSDL) [113], which describes the service in a common XML grammar [69]. The completed service definition is then registered with the DSM service registry which makes it available for subscription by customers. The WDSL document describes the following information about the service so that it can be invoked by the service customer [113]:

- Interface information describing the public functions to invoke the service.
- Data type information for all message requests and message responses.
- Binding information about the transport protocol to be used.

- Address information for locating the specified service.

Service Level Agreement Template

An SLA template is created by the service provider and accompanies the service definition that is made available to potential customers. The SLA template is used as a starting point during the service subscription process and the agreement of an SLA between the customer and the service provider.

An SLA Template is created by the service provider as a partially completed WSLA document (an example of which is presented in Appendix B), which contains a number of fields that are filled in when the SLA is agreed between the service customer and the service provider. It may additionally define constraints on those fields which impose restrictions on the values that the SLA parameters can take. This is used by the service customer when requesting the service and by the service provider when validating the SLA. SLA parameters are defined for each operation defined in the WSDL description.

Every SLA parameter defined in the SLA refers to a QoS Metric, which is a measurable property of the service. A QoS metric referred to may be either calculated as an aggregation of the resource-level QoS metrics used in the service definition, such as the *time* and *cost* QoS. Additionally, a QoS metric may be a composite metric that is calculated from other QoS metrics using a specific aggregation function. For example, composite metrics may define average availability or minimum throughput of the service.

The completed SLA document is then submitted to the service provider for validation and provisioning.

5.2.1.1 Service Management Component

Once the subscription process has completed and an SLA has been agreed between the customer and the service provider, a Service Management Component (SMC) is created which is used to manage the fulfilment of the service. The Service Management Component is an object representation of the SLA that is internal to the service provider and captures the management information required to fulfil the service implementation. The SMC is used by the Service Execution Manager during the fulfilment of all requests for the service by the customer. The SMC is implemented by the class `ServiceComponent`, as illustrated in Figure 5-7. The class `ServiceComponent` maintains a reference to the service customer, as defined by the class Customer. This customer identifier is used to authenticate the customer when a service request is received.

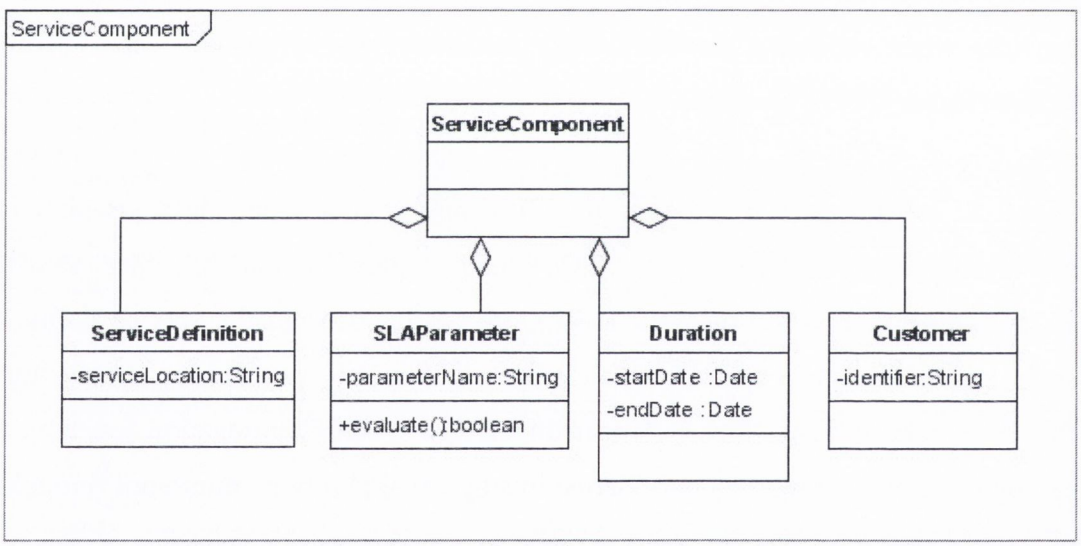


Figure 5-7 UML of Service Component

The SMC maintains information on the validity period for the SLA, captured in the class Duration. Each SLA has a `startDate` and `endDate`, outside of which requests for the service from the customer will not be fulfilled. The `ServiceDefinition` class maintains a reference to the definition of the service that is covered by the SLA. The `serviceLocation` attribute contains the URL where the BPEL4WS service description can be located. Each SLA Parameter agreed in the SLA is represented by

the `SLAParameter` class. The UML for the `SLAParameter` and related classes are presented in Figure 5-8.

Each SLA Parameter references a single QoS metric, defined by the class `QoSMetric`, which is a measurable property of the service provided by DSM. The evaluation of each QoS Metric to determine whether it fits within the bounds of the SLA parameter is defined in the class `Predicate`. This provides a `test()` function which returns a Boolean, indicating whether the SLA parameter bas been violated. The Predicate class provides base conditional evaluation functionality and is designed to be sub-classed to provide application-specific condition evaluation.

Each SLA Parameter may be defined in terms of a number of *states* over the lifetime of the SLA agreement. This encompasses the definition within the SLA Parameter, that a penalty implication, captured by the `Penalty` class may increase should the SLA Parameter be violated a given number of times within a defined period. The `transition` class captures the threshold limits which cause the SLA Parameter to transition from one state to another.

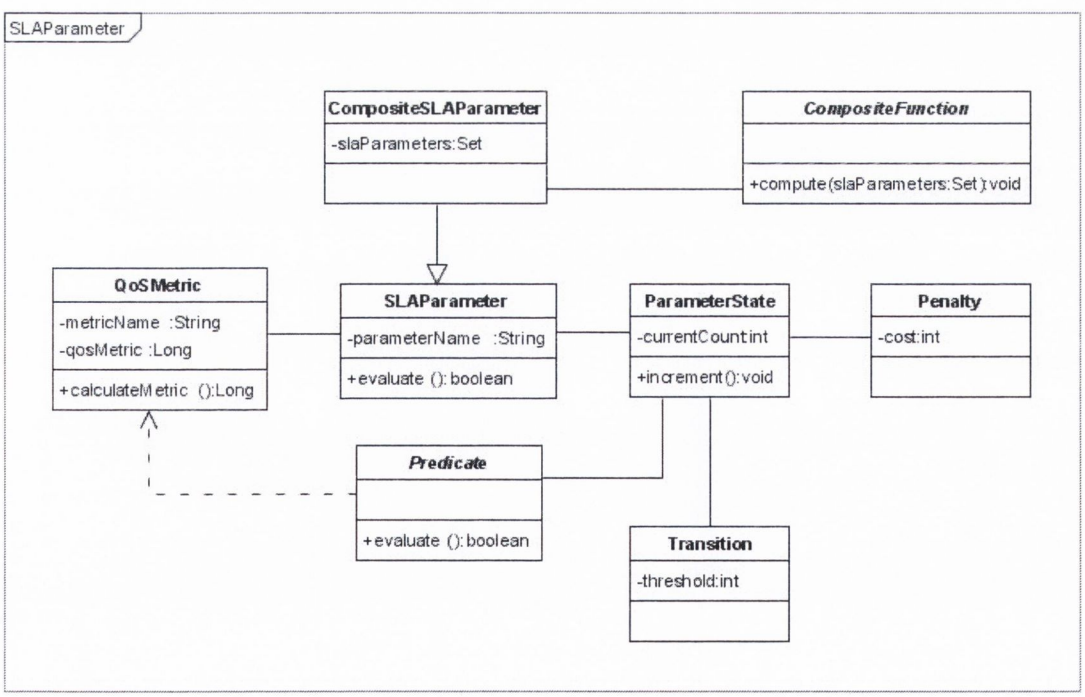


Figure 5-8 UML of Service Parameter

SLA Parameters defined in the SLA may be further aggregated into *Composite SLA Parameters*. A composite SLA parameter uses a `CompositeFunction` to aggregate a number of SLA parameters. Composite functions include the average, minimum, maximum of the SLA Parameters included in the composite SLA parameter.

DSM supports the definition of QoS metrics that are calculated from the aggregation of the measured QoS metrics from each of the activities of the business process. This includes the *execution time* and *execution cost* QoS metrics. The UML for this QoS aggregation is presented in Figure 5-9. Each `QoSNode` represents an activity in the business process and is responsible for aggregating the QoS data from the preceding activities using a QoS-specific aggregation algorithm, which is implemented in the base class `NodeAggregation`. Each QoS metric that is aggregated from activity process measurements subclasses the `NodeAggregation` class to define the specific metric aggregation algorithm.

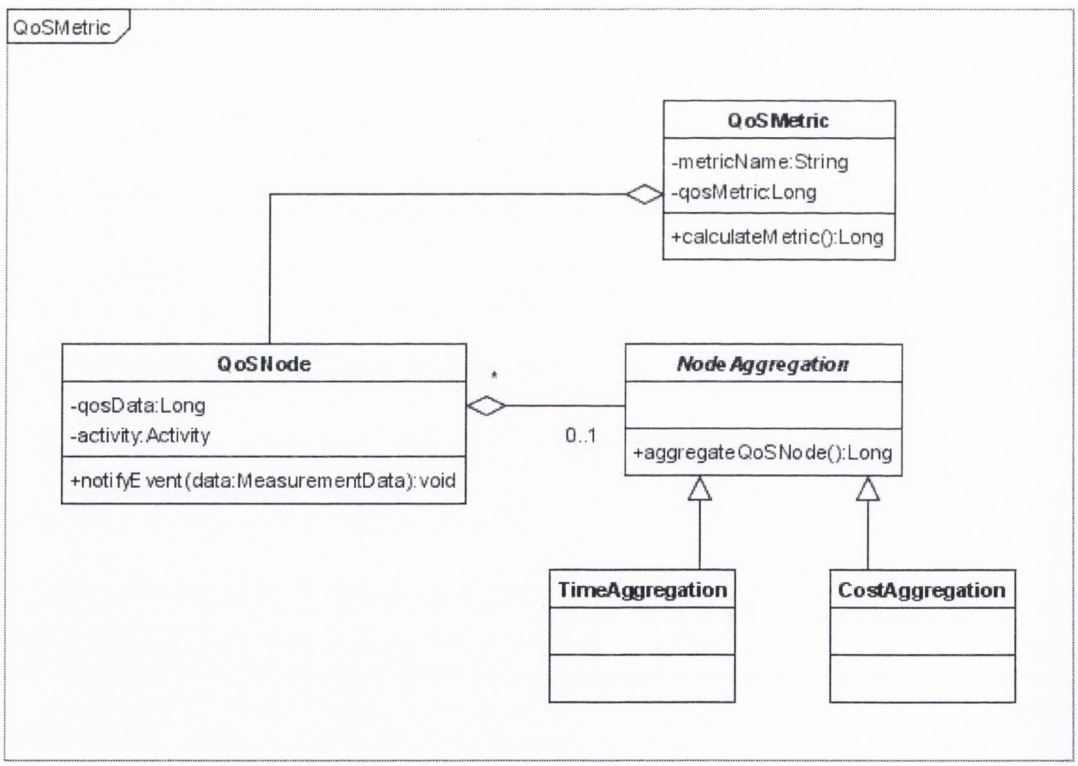


Figure 5-9 UML of Qos Metric

This design captures the aggregation of QoS data both when activities are executed in sequence and also when there are multiple activities being executed in parallel, depending on the requirements of each QoS metric.

In the case of the *cost* QoS metric, the aggregation algorithm includes the cost of executing each activity, regardless of the executing sequence. This is described in Figure 5-11. For the *time* QoS metric, the overall value of the metric is the total executing time for the business process. For activities executing in sequence, the aggregation algorithm sums the execution time for each of the activities. For activities that execute in parallel, the aggregation algorithm only includes the longest execution time across the activities that are executing in parallel. This is described in Figure 5-12.

```
public Long getQoSValue() {

        Long metricValue = 0L;

        // aggregate the QoS parameters of the child nodes.
        // The aggregation type s specific to the NodeAggregation
        // subclass.
        if(qosAggregator() != null) {
            metricValue = this.qosValue
                + qosAggregator.aggregateQoSNodes();

            return metricValue;
        }

        // there are no more child nodes to include
        else {
            return this.qosValue;
        }
```

Figure 5-10 Aggregating QoS Metrics

```
public Long aggregateQoSNode() {

        Long metricValue = 0L;

        java.util.List childNodes = getChildNodes();
        java.util.Iterator it = childNodes.iterator();

        while(it.hasNext()) {
            QoSNode node = (QosNode)it.next();
            metricValue += node.getQoSValue();
        }

        return metricValue;
    }
```

Figure 5-11 Aggregating the Cost QoS Metric

```
public Long aggregateQoSNode() {

        Long longestDuration = 0L;

        java.util.List childNodes = getChildNodes();
        java.util.Iterator it = childNodes.iterator();

        while(it.hasNext()) {
            QoSNode node = (QoSNode)it.next();
            if(node.getQoSValue() > longestDuration ) {
                longestDuration = node.getQoSValue();
            }
        }
        return longestDuration;
    }
```

Figure 5-12 Aggregating the Execution Time QoS Metric

5.2.2 Service Execution Manager

Figure 5-13 presents the architecture of the Service Execution Manager. All client service requests are received by the Orchestration Service which is responsible for the fulfilment of those requests. When a request is received, it is authenticated in a manner as discussed in [87] whereby a filter is used to intercept the request prior to the request being passed to the Orchestration Service. A client identifier is subsequently included with the request so that the Service Management Component instance for this service request can be located by the Orchestration Service.

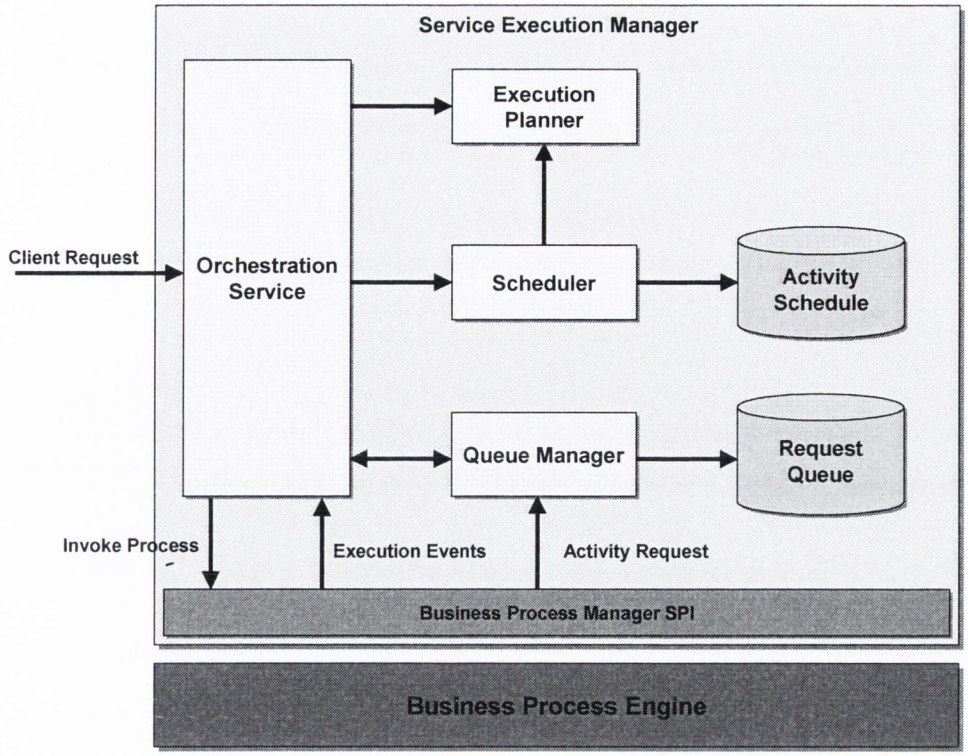


Figure 5-13 Service Execution Manager Architecture

After receiving a request, the Orchestration Service locates the Service Management Component (SMC) associated with this service request from the DSM Service Manager. The first stage of the fulfilment process is to create an *execution plan* for this service request. This is described by the UML sequence diagram in Figure 5-14. The `Planner` obtains the process definition describing the composite service from the SMC instance and navigates through the business process, represented as a directed graph [23], using the weightings defined at each conditional branch to derive an execution plan for the process. The execution plan consists of a set of *schedulable activities* that are the most likely to be executed, based on the weightings defined by the service provider. The `Scheduler` is responsible for allocating Web service resources to each of the activities and creating an execution schedule based on the list of schedulable activities for all currently executing business processes.

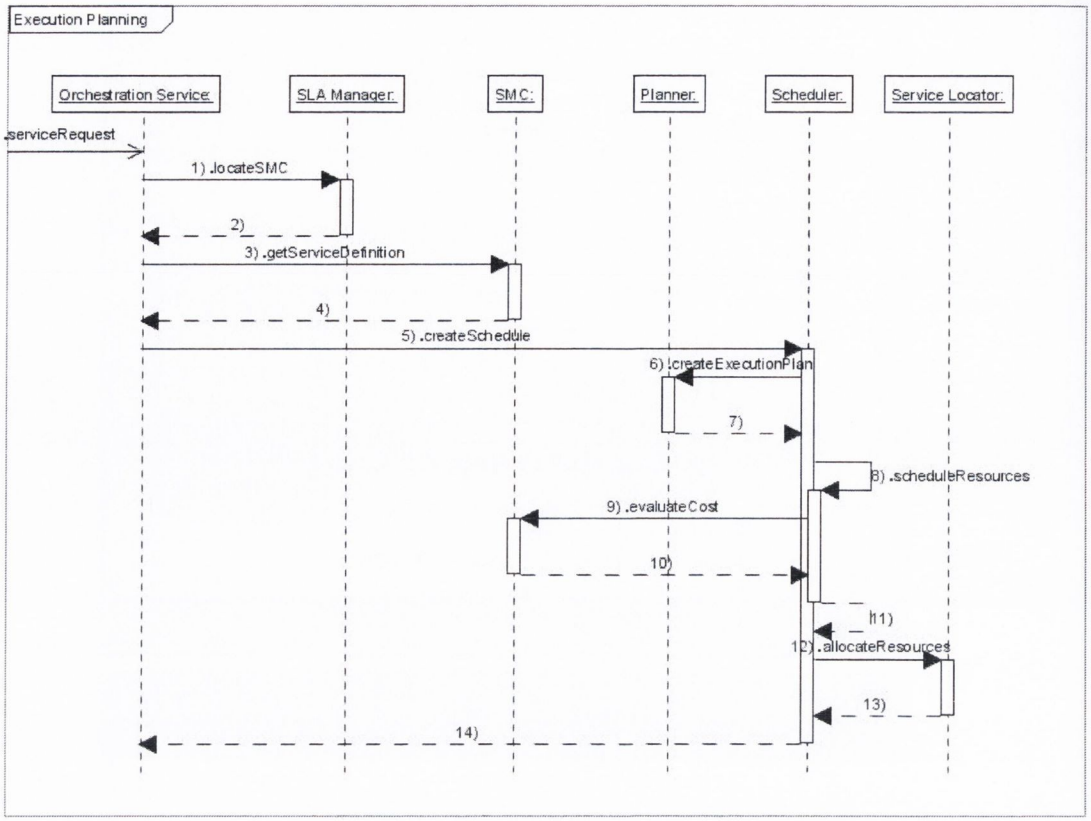


Figure 5-14 Planning Service Execution

The Scheduler is invoked with the `schedule()` method to create a new resource allocation schedule for the pending activities. The `bestSolution` variable in the Schedule class maintains a reference to the best solution that has been found. This is initialised to `null` when the scheduler is initialised, indicating that no solution has yet been found. The `traverseTree` method instantiates new nodes in the tree, level by level, as the tree is searched. The entire tree is not created in memory in advance of being searched for both speed and to decrease the memory requirements of the scheduler. As described in section 4.4.2, each level in the tree represents the assignment of Web services to a particular activity request with a particular starting time. This is described by the class `TreeNode`. The cost of each node is evaluated as a part of the whole solution cost using the method `Node.evaluateCost()`. This method uses the SMC instance associated with the resource request to evaluate the cost implications of the Web service – activity assignment. This cost of this node is evaluated in relation to the overall cost of the current partial solution. If this cost is less than the current best solution, or no solution has yet been found, this node is added to the partial solution and the tree is searched further from this node. If the cost of the partial solution including this node is greater than the current best solution

however, this node is discarded and the Scheduler backtracks to evaluate another node in the search tree.

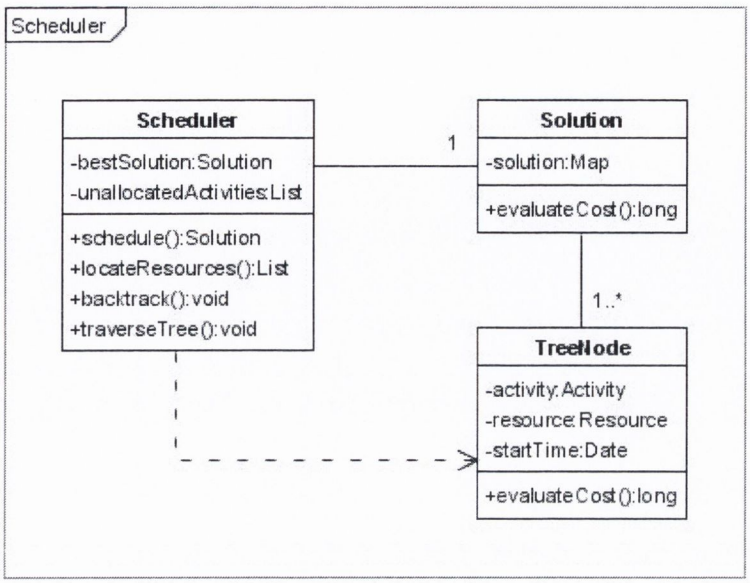


Figure 5-15 UML of Scheduler

The Orchestration Service initiates the fulfilment of the client request by invoking the business process associated with the composite service request using the business process manager Service Provider Interface (SPI). This invokes the underlying business process engine and initiates a new process instance.

As the business process engine executes the process instance, Web service resources are mapped to activity definitions as the activity is invoked. The business process manager SPI provides a lookup service to the business process engine to map the abstract service definitions to actual Web service instances. The business process engine invokes the `lookupService()` method of the SPI including the abstract service name in the activity definition. The SPI forwards this request to the `QueueManager`, which adds the request to a request queue. The outstanding requests on the queue are processed and allocated resources according to the activity schedule, and the SOAP invocation details are passed back, via the SPI, to the business process engine to invoke the Web service implementation.

During the execution of a service process, the Orchestration Service receives notification events about the execution status from the process engine, represented by

the `StatusEvent` class. This event captures information about the process flow of the process instance and the path taken in the conditional branches of the process definition. If the execution path differs from the planned execution path, the Orchestration Service informs the Execution Planner by calling the `updateSchedule()` method. This event triggers the Scheduler to re-calculate the activity schedule with the updated information. The Scheduler will first call the Planner to re-estimate the execution plan for the process instance to create a revised list of schedulable activities. It will then re-create a new resource allocation and activity schedule with the updated information.

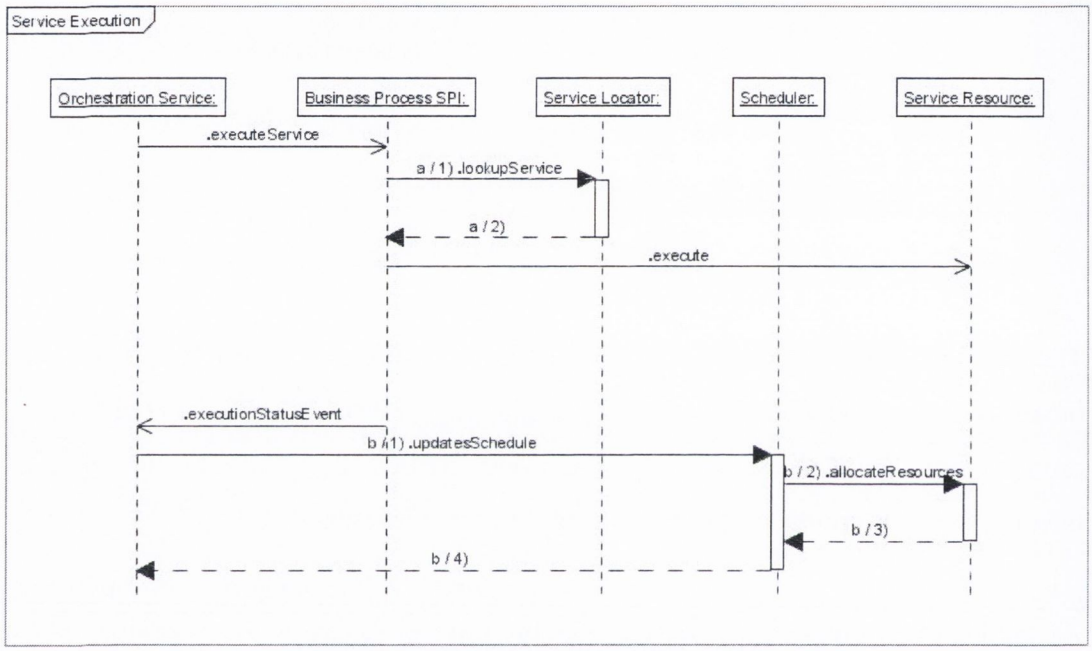


Figure 5-16 Executing a Service Instance

5.2.3 Resource Broker

This section describes the DSM Resource Broker, the component that is responsible for the interaction with the individual Web services resources that are implemented in the service provider environment. The Resource Broker provides a directory service for the Web services implemented by the service provider that are available for use by the Service Execution Manager in the fulfilment of service requests. In addition to providing a directory of resource instances, the Resource Broker also stores additional

non-functional attributes about the Web service, such as the cost of invoking the service and the estimated execution time for the Web service. This information is used by the DSM scheduler when allocating Web service instances to resource requests. The Resource Broker also provides monitoring and measurement functionality about the execution of each Web service. This information is relayed to the Service Management Component to determine compliance with the service SLA.

5.2.3.1 Resource Directory

The Resource Directory provides a directory service of Web services that are implemented in the service provider environment, and enables the DSM Execution Manager to locate Web service instances that are specified by the business process definition.

The service provider registers Web services with the Resource Directory to make them available to DSM. The Resource Directory includes the concept of an *endpoint description* to capture the non-functional characteristics of the Web service. These characteristics are used by the Scheduler component to find the "best fit" resource allocation.

```
<service name="LoanServiceImplA" type="LoanService">
 <port name="LoanServicePort" binding="tns:LoanServiceBinding">
  <soap:address location="http://localhost/services/LoanServiceImplA"/>
 </port>
 <attribute name="duration" value="100" metric="milliseconds" />
 <attribute name="cost" value="2" metric="eurocent" />
 <attribute name="availability" value="99.99" metric="percent" />
</service>
```

Figure 5-17 Web Service Endpoint Description

The business process definition includes a reference to the abstract service type, as described in section 5.2.1.0. On receipt of a service request type, the DSM Service Manager queries the Resource Directory for all Web service implementations of that type. The allocation of a Web service instance to the request from the business process execution is carried out by the Scheduler component.

5.2.3.2 Measurement Service

The Measurement Service is responsible for the measurement of quantifiable QoS-related execution data for each of the Web services, and the subsequent relay of that information to the appropriate Service Management Component.

Measurement data is received by the Measurement Service from Measurement Adapters. The `QoSMeasurementAdapter` interface provides a class interface which all measurement adapter instances implement. This allows the Measurement Service to receive measurement data through a number of different measurement mechanisms, including existing management systems such as SNMP [17] or Tivoli [17], which the service provider may utilise to manage the service resources. In DSM, the default measurement adapter implementation provides for the measurement of Web services through the use of SOAP *Message Handlers* [76]. This enables the monitoring of the Web service execution without requiring any code changes to the Web service implementation.

```
public class WSQoSMeasurementAdapter
    extends org.apache.axis.handlers.BasicHandler
    implements QoSMeasurementAdapter {

    MeasurementService service;

    /**
     * Initialise the Adapter with a reference to the Measurement Service
     */
    public void init(MeasurementService service) {
        this.service = service;
    }

    /**
     * Method called by the Apache Axis framework when the service
     * is requested.
     */
    public void invoke(org.apache.axis.MessageContext msgContext) {

        // identifier for the business process instance
        String processId = (String)msgContext.getProperty("processId");

        // if this is the service request
        if(! getPostPivot()) {
            service.setStartTime(processId, System.currentTimeMillis());

        // otherwise this is the service response
        } else {
            service.setEndTime(processId, System.currentTimeMillis());
        }
    }
}
```

Figure 5-18 Monitoring Web Service Execution

The Measurement Adapter is invoked by the web services framework prior to the request being processed by calling the `invoke()` method. The `getPostPivot()` method, implemented by the web services framework, allows the Measurement Adapter to determine whether the adapter is being invoked for the request or the response. This enables the Measurement Adapter to calculate the total execution time for the request.

Each resource Web service deployed in the service provider environment includes the Message Handler in its Web Services Deployment Descriptor [76]. In the case of the Apache Axis framework [11] used by DSM, this is achieved using the configuration described in Figure 5-19.

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
    xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

<handler name="measurement" type="WSQoSMeasurementAdapter">
</handler>
</deployment>
```

Figure 5-19 QoS Measurement Handler Deployment

The Measurement Service creates Measurement Events, as described by the Java class `MeasurementEvent` in Figure 5-20. The Measurement Service quantifies the measurement data captured by the Measurement Adapter and creates a Measurement Event object for each QoS Metric. The Measurement Service associates the both the executing process id and the activity id within that process that invoked this Web service. Using these identifiers, the Measurement Event is then received by the Service Management Component and the `QoSNode` within the SMC instance that is associated with this Web service invocation is updated with the value of the QoS Metric.

```
public class MeasurementEvent {

    String metricType;
    Long metricValue;
    java.util.Date measuredTime;
    String processId;
    String activityId;

}
```

Figure 5-20 Measurement Events

5.3 Business Process Engine Integration

This section describes the integration with the Business Process Engine, which is responsible for the execution of the composite service business process definition.

All interaction with the business process engine takes place using the Business Process Service Provider Interface (BPI). This provides a level of abstraction between

DSM and the business process engine implementation, allowing for different implementations of the process engine to be used. The SPI is responsible for both invoking actions on the business process engine and also receiving events from the engine, including management status events and requests for Web service implementations for each of the activities in the process definition.

5.3.1 Process Deployment

When a business process definition is created by the service provider it must be deployed in the process engine before it can be invoked. The SPI adds the process definition, created as a BPEL4WS XML document, to the process engine configuration which makes it available for runtime invocation.

5.3.2 Process Invocation

The BPI receives requests from the Orchestration Service to create a new business process execution instance. The BPI is responsible for invoking a business process engine and creating a new process instance of a process definition that has previously been deployed in the business process engine. A request identifier is passed back to the Orchestration Service for future reference.

5.3.3 Management Events

Management Events are received by the BPI from the process engine instance and are used to relay information about each of the executing process instances. The management events are passed to the SMC associated with the process instance, which uses the information to determine whether the execution path for the process instance needs to be re-estimated. This occurs when the management event indicates that the process flow has taken a different path to that predicted by the Execution

Planner. In this case, the Execution Planner is invoked again by the Orchestration Service to re-estimate the execution path based on the updated information.

5.3.4 Service Request Events

As a process instance is executed by the business process engine, an activity may require functionality that is provided by a Web service implemented by the service provider. As described in section 5.2.1.0, the process definition supports late binding of service implementations to activities. The abstract service name defined in the process definition must be mapped to an actual service implementation at runtime. The business process engine invokes the BPI to request a Web service implementation. This request is passed to the Queue Manager, which is responsible for processing the request according the resource allocation created by the Scheduler component.

5.4 DSM Implementation

The previous sections described the main components of DSM and the overall system design and architecture of the framework. This section presents a description of the technologies and platforms used during the implementation of DSM framework.

The platform on which DSM was developed and deployed consisted of a network of PCs running the Windows 2003 Server operating system. This platform was chosen due to the fact that it is recognised as an industry *de facto* standard. DSM was implemented using the Java programming language, version 5.0. Java was chosen as the programming language for DSM due to the extensive support for the web services standards and frameworks provided in the Java language. The web services frameworks used by DSM are the Axis SOAP framework [11] version 1.2 and the Web Services Invocation Framework (WSIF) [12]version 2.0, both from the Apache

Software Foundation. The web services implementations in DSM were constructed through the deployment of the Axis and WSIF frameworks in the Apache Tomcat Servlet container. For DSM, Apache Tomcat [10] version 5.5 was used. The business process engine that was used as the execution engine for the composite business processes is the Nexala Business Process Engine [83], version 4.2. The Nexala Business Process Engine was chosen as it provides direct support for BPEL4WS definition of business processes, and the late binding of Web services to Activities using abstract service definitions, as required by DSM.

5.5 Design and Implementation Issues

This section describes the issues that were encountered during the design and implementation of the DSM framework.

5.5.1 Design Issues

One of the key design issues that became clear during the implementation of the DSM framework is that the initial efficiency of the scheduler is significantly improved if the initial weightings assigned to the conditional branches of the business process definition by the service provider accurately reflect the most likely outcomes of the condition. While the weightings are automatically adjusted over time by DSM, if the weightings are not initially accurate, the re-scheduling of resources will occur as the initial schedule is incorrect. This may have an impact on the overall performance of the framework as the number of service resources and customers is increased significantly. Therefore, a detailed analysis of the business processes should be carried out by the service provider to ensure the initial accuracy of the weightings.

During the construction of an application service using the DSM framework, the service provider must also ensure that each of its service resources are both accessible via a Web service interface, and also have QoS metrics defined for that service. This is a requirement so that the QoS metrics for each service resource can be aggregated by DSM into the overall end-to-end QoS for the application service. For service resources that do not have explicit QoS metrics defined, the service provider must

establish the resource capabilities prior to using the resource as a part of an application. However, once the QoS metrics have been defined for a resource, it becomes very straightforward for the service provider to re-use that resource across multiple different application services.

5.5.2 Implementation Issues

The implementation of the DSM framework that was prototyped for this thesis uses SOAP message handlers to capture QoS measurement data for each of the service resources. However, during the implementation it was noted that the default measurement adapters only capture measurement data within the service provider environment. There may be a requirement to define QoS metrics in the SLA that uses measurement data captured outside the service provider environment for example within the customer environment so that potential network delays are also taken into account. The measurement service in DSM can be easily extended however so that measurement adapters can be placed within the customer environment, should this be a requirement.

5.6 Summary

This chapter has presented the implementation of the DSM service management architecture. The chapter began with an overview of the system architecture and the functionality of each of the components in the system was discussed. A comprehensive description of the system implementation details was then presented. This was followed by a description of the technologies that were used for the implementation of DSM and a discussion of the issues that arose during the design and implementation. The table in Figure 5-21 illustrates how each of the components in the DSM architecture contribute to the requirements for a dynamic service management framework as outlined in Chapter 3.

		SLA Manager	Service Registry	SMO Manager	QoS Agg. Service	Service Locator	Orchestration Service	Scheduler	Resource Directory	Resource Manager	Measurement Service
Dynamic Services	Composite Services	✓		✓	✓		✓				
	SLA Management	✓	✓	✓			✓				
	Dynamic Resource Assignment			✓		✓		✓		✓	
Differentiated levels of Service	Customer Specific Service Levels	✓		✓			✓	✓			✓
	Service Customisation		✓	✓		✓	✓				
Optimal Provision of Services	Dynamic Resource Provisioning				✓	✓	✓	✓	✓		✓
	SLA Driven Delivery Optimisation				✓			✓	✓	✓	

Figure 5-21 Summary of DSM Components

The framework presented in this chapter was used as the basis for a framework for the development of dynamic services and for a series of evaluation tests, which are described in the next chapter.

Chapter 6
Evaluation

This thesis describes DSM, a framework for the management of dynamic services. In particular, DSM addresses the issues of supporting services that are dynamic in nature, are defined with differentiated levels of quality to the customers of the services, and addresses the issues of resource allocation to the service requests in such a way as to minimise the penalties incurred due to service violations. In order to show that the goals of this thesis have been achieved a number of applications are built that utilise the techniques described in this thesis. By building two very different applications it is intended to demonstrate that the framework proposed in this thesis is not tied to any specific application domain.

The remainder of this chapter is organised as follows:
Section 6.1 outlines the goals of this thesis as implemented in the DSM framework. Section 6.2 describes in detail two applications that were built using DSM to fulfil both applications' requirements. Following on from this, section 6.3 presents a comparison between DSM and a number of existing frameworks for dynamic service management that were presented in chapter 3. Finally, a description of how the thesis goals and requirements have been fulfilled is presented in section 6.4. The chapter is summarised in section 6.5

6.1 Summary of Thesis Goals

A number of goals for the work described in this thesis were presented and discussed in chapter 1. In this chapter the thesis is evaluated with respect to each of these goals. For reference, the goals of the thesis are summarised as follows:

1. *Enable the dynamic outsourcing of services to service providers.* DSM automates the processes that are required to establish a service relationship between the service provider and a service customer. DSM also manages the agreed-upon SLA across the entire lifecycle of the relationship. In addition DSM minimises the effort with establishing a service relationship, as it performs dynamic allocation of resources to service requests.

2. *Enable service providers to offer differentiated levels of service to each customer.* DSM allows the service provider to offer each customer the same service at unique service levels, depending on customer requirements. By enabling the service provider provision resources dynamically (i.e., on demand), service providers can best utilise their computing resources and can dynamically redistribute resources across multiple customer workloads.

3. *Provide the service provider with mechanisms that fulfil service requests in a cost effective manner.* DSM utilises optimisation techniques to determine the allocation of resources to customer requests which minimises the cost of implementing the services.

The following sections examine these goals in more detail and discuss how the DSM framework described in this thesis fulfils each of these goals.

6.2 DSM Applications

This section describes how dynamic, differentiated services may be built using DSM. Two applications have been implemented to verify that DSM meets the requirements outlined in chapter 1. The applications chosen for this thesis evaluation are taken from different application domains with very different SLA requirements and demonstrate how the objectives of this thesis have been met. The applications described are a loan prepayment assessment application that is offered as a service to customers and a maintenance planning application for the rail industry. These applications are described in the following sections. Further in this chapter some conclusions are presented based on the experience obtained in trialling these applications. In describing these applications it can be shown that:

1. DSM enables service providers to create application services on top of existing Web service resources where the Web services are dynamically bound to the application service instances. DSM enables the service provider to associate a service level agreement with the service offering, and facilitates the calculation of the QoS metrics defined in the SLA from the measurement of the underlying resource Web services.

2. An application built using DSM can be offered at different levels of quality to each subscribing customer without imposing any restrictions on either the implementation or deployment of the underlying Web service resources.

3. The fulfilment of these applications and the allocation of Web service resources to service requests are governed by SLA contracts agreed with the customer in a cost effective way to minimise the financial penalties incurred due to service violations.

6.2.1 Loan Assessment Application Service

In order to evaluate the DSM framework in a service provider environment where service resources are dynamically outsourced to third party suppliers, a loan assessment application has been built. In this application, the service provider dynamically assigns different third party service resources which each have different QoS metrics to each customer request depending on the overall end-to-end QoS requirements of that customer.

The application provides an online service to financial institutions to evaluate a customer's mortgage loan and estimate the likelihood of loan pre-payment, including a customer moving the loan to another financial institution or selling the property and paying off the loan early. This information is used by financial institutions to minimise the risks associated with mortgage loans and also to assist in retaining existing customers. The financial institution submits a portfolio of loans to the service provider for evaluation. The application aggregates credit history, demographic, and property record information about the loan from third-party data vendor Web services in addition to the information about each loan provided by the financial institution. This enhanced data of loan information allows the service provider to create a highly customised assessment model and as such more accurately forecast pre-payment of the loan. The assessment model evaluates all of the information about each loan in the portfolio submitted by the financial institution and produces a *loan score*, indicating to the financial institution the likelihood of loan pre-payment.

The Loan Analysis application service demonstrates how, through the use of DSM, an application service can be developed by a service provider that is created as a composition of existing Web service resources, implemented by the service provider and its partners. The service provider is able to offer quality guarantees to its customers through SLA contracts agreed with the customer. DSM also allows the service provider to offer the service at different quality levels to each customer, without the requirement for dedicated resource provisioning. In order to meet these requirements the DSM framework provides the following functionality to achieve the applications goals:

1. Dynamic Services

The Loan Assessment application uses the DSM framework to create a new service offering which is constructed through the use of existing Web service resources implemented by the service provider and its partners. DSM enables the Loan Assessment application to be defined as a business process, where the activities in the process reference the abstract service type that is required. In the Loan Assessment application, the abstract Web service types that can be fulfilled by a number of Web service implementations, each with differing quality and cost attributes. DSM dynamically binds a Web service instance in order to best meet the SLA requirements of the requesting customer. The QoS metrics defined by the SLA parameters agreed with each service customer is calculated by DSM as the aggregation of the QoS metrics for each of the resource Web services that is selected during the fulfilment of a service request.

2. Differentiated Levels of Service

The Loan Assessment application uses the DSM framework to support applications customers that require different levels of service quality. Customers can subscribe to the application with different levels of service, including cost, response time, and number of loans to be processed in each request. DSM uses the QoS requirements defined by each customer when allocating Web service resources to each customer request. Each Web service resource each customer's requirements when allocating Web service resource instances to customer requests as the different Web service resources have different QoS capabilities.

3. Dynamic Resource Allocation Management

The DSM framework dynamically allocates Web service resources to fulfil customer requests for the Loan Assessment application while also minimising violations of the SLA targets that have been agreed with each customer. The Loan Assessment application includes a number of Web service resources of differing QoS capabilities that can fulfil the functional requirements of the application. DSM dynamically allocates the resources to customer requests in a cost effective manner and that minimises SLA violations.

6.2.1.1 Loan Assessment Application Implementation

The application is created through the orchestration of a number of Web service resources and is structured as a business process. This business process definition is illustrated in the state transition diagram in Figure 6-1. The application is then made available to customers via a Web service interface, defined in a WSDL document.

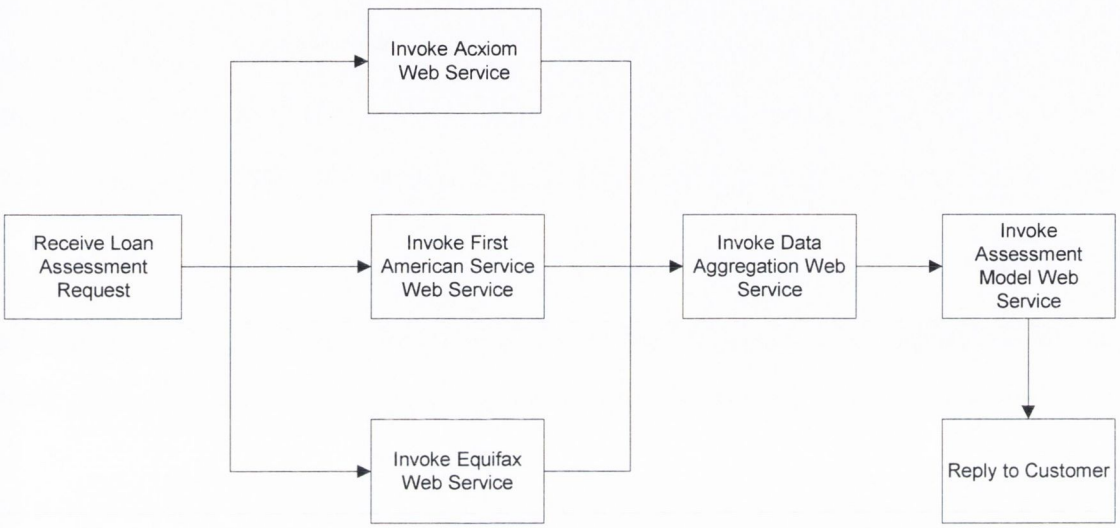


Figure 6-1: Loan Assessment Application Process Definition

A customer invokes the Web service offering via a SOAP interface and submits a portfolio of loans along with the request. This portfolio is defined in an XML document and contains information about each loan in the portfolio, including: owner name; address; loan amount and loan age.

After a request is received from a customer, extra information about the loans in the portfolio being assessed is retrieved from third-party data provider Web services as the first stage of the application business process. The data retrieved from third party data providers include demographic information, provided by, for example, Acxiom [2], property record information provided by, for example, First American [46] and credit history information provided, for example, by Equifax [42]. This information is used when evaluating each loan and provides greater assessment accuracy. Each of the third-party data providers provide a Web service to enable subscribers to submit a set of loans (including the loan owner name and address) which the data provider will use as a part of its internal process for locating the extra information about each loan.

The data providers charge on a per-loan basis and provide guarantees for the response time of the service.

These three Web services are invoked in parallel, and once all three have completed, the process continues and the aggregation Web service is invoked. This service is implemented by the service provider and is responsible for aggregating the data from both the customer provided information and the data vendor information that has been retrieved together prior to invoking the assessment model Web service. Again, this Web service is provided by the service provider environment. The assessment model Web service evaluates the combined loan information and produces a risk assessment which is subsequently returned to the customer. In this application the service provider maintains three instances of the assessment model Web service, which are located on three separate physical servers. Each Web service can accommodate a different number of loans and, due to the server performance characteristics, has different execution times. Should the load on the server increase over time, DSM enables the service provider to easily add new Web service instances for the assessment model Web service, deployed on new physical hardware, in order to meet the new demand. This has no impact on the existing the delivery of the service to customers.

The Loan Assessment application is designed to handle a number of customer requests concurrently and DSM manages the allocation of the resources across all executing requests. In the trials of the application that were undertaken, 10 concurrent loans file were sent to DSM, with each file containing one thousand individual loan records. During the allocation of resources to customer requests, DSM aggregates the QoS metrics that have been defined for each of the Web service resource to determine the impact on the application SLA that has been agreed with each customer. In this way, DSM allocates resource instances to each request that where possible will use those service resources with the lowest cost, while still minimising any penalties that may be incurred due to a service level violation. DSM manages the allocation of resources dynamically and adjusts the allocation should this be necessary. For example, should the execution time of Acxiom Web service be longer than anticipated, DSM will re-schedule that customer request to invoke an assessment model Web service with a faster response time. While this may cost more from a

service delivery perspective, it will eliminate a service level violation penalty that would have otherwise been incurred, this reducing the overall cost of the service fulfilment.

6.2.1.2 Application User Interface

Figure 6-2 presents the user interface of the Loan Assessment Application that is made available to the service provider. The application was implemented using DSM with the service provider user interface built using Java Swing [102].

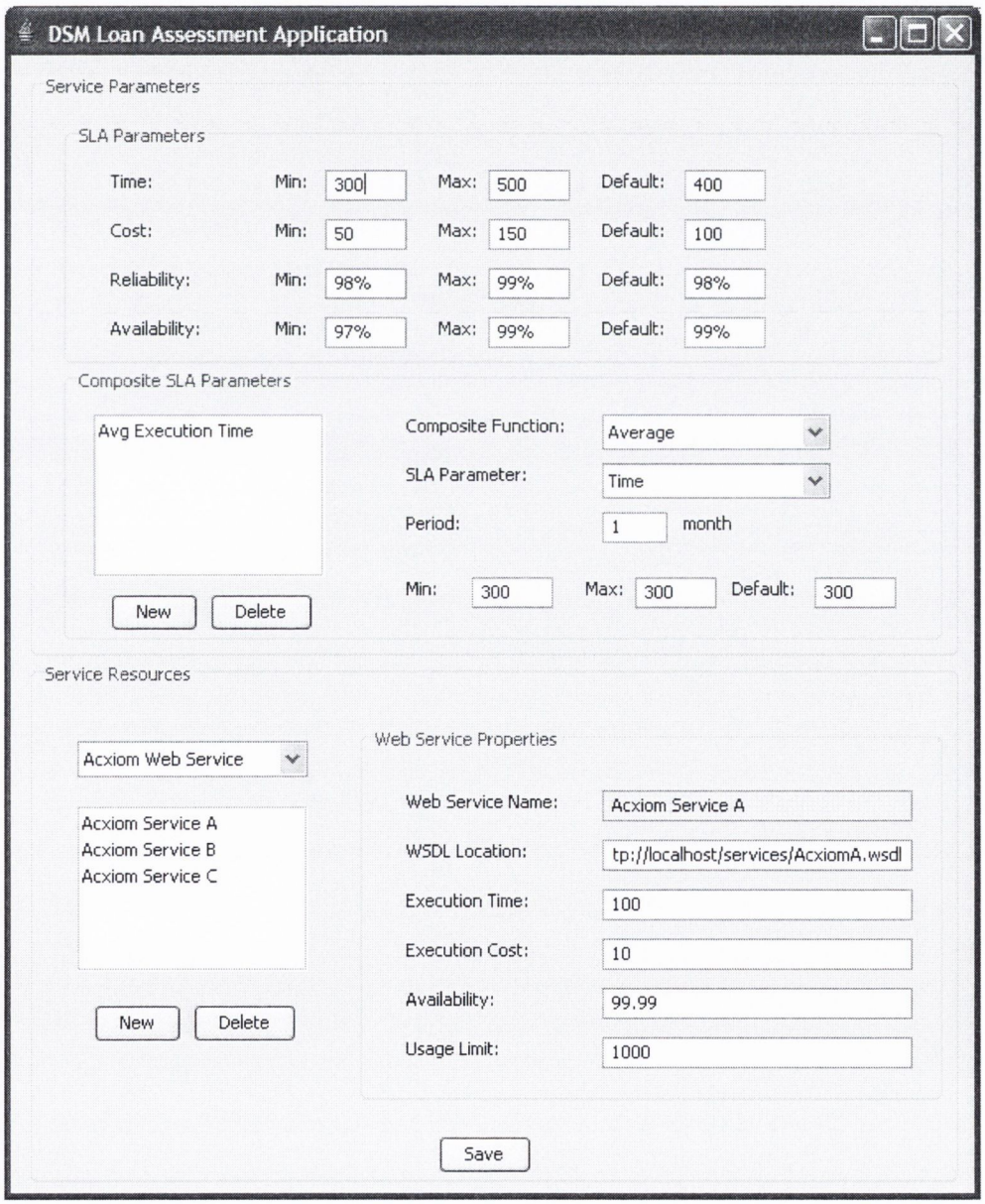


Figure 6-2: DSM Loan Assessment Application User Interface

DSM enables the specification of the service offering to be separated from the Web service resources that make up the implementation of that service. In the Loan Assessment application, the Acxiom Web service is available as three different service implementations, each with differing levels of quality and cost. These implementations are defined as `Acxiom Service A`, `Acxiom Service B` and `Axiom Service C` in Figure 6-2. The QoS details of each of these services are described in Figure 6-3. The business process definition references to the Acxiom Web service as an abstract service type. The selection of one of the Web service instances for each service type that has been defined is performed by DSM as the service executes, depending on the QoS requirements of the requesting service.

Acxiom Web Service	**Service Description**
Acxiom Service A	**Execution Time:** 100 seconds **Cost:** 10 Euro **Availability:** 99.99 % **Usage Limit:** 1,000 loans
Acxiom Service B	**Execution Time:** 200 seconds **Cost:** 7 Euro **Availability:** 99.99 % **Usage Limit:** 5,000 loans
Acxiom Service C	**Execution Time:** 500 seconds **Cost:** 5 Euro **Availability:** 99.99 % **Usage Limit:**10,000 loans

Figure 6-3 Acxiom Web Service Implementations

The application user interface facilitates the creation of an SLA template that is associated with this service offering. It enables the service provider to specify the SLA parameters that are made available to customers of the service. For the base SLA parameters *Time*, *Cost*, *Reliability*, *Availability*, the service provider is able to specify the minimum and maximum values that are acceptable, along with a default value that is presented to the customer. Through the application, the service provider can also create composite SLA parameters, which are calculated from an existing SLA Parameter using a composite function, calculated over a period of time. An example of a composite SLA parameter, as illustrated in Figure 6-2 is the *Average Execution Time*, averaged over a month.

The service provider and customer use the SLA template as the basis for agreeing an SLA contract between the two parties. The SLA contract specifies the targets for each of the SLA parameters defined in the application user interface that must be met by the service provider. The Loan Assessment application enables each customer to specify both the QoS targets that they require of the application and also the usage limits of the application. DSM provides the customer with the ability to specify the requirements for the QoS metrics Execution Time, Cost, Reliability and Availability, which DSM uses to define the allocation of Web service resources when fulfilling customer requests. The Loan Assessment application further illustrates how application specific parameters may be defined for a service.

The Loan Assessment application illustrates how DSM enables application offerings that can be easily built through the orchestration of existing Web service resources that are implemented by both the service provider and also its partners. DSM provides the following functionality in order to meet the applications goals:

1. DSM enables the service provider to offer new dynamic services, defined as a composition of existing service resources, with SLA guarantees to its customers. DSM aggregates the QoS parameters for each of the service resources that make up the loan assessment application to evaluate the QoS parameters defined in the SLAs.
2. DSM enables the service provider to offer the service offering to customers who have unique QoS requirements.
3. DSM uses the aggregation of QoS metrics to dynamically assign Web services resources across all executing customer requests.

6.2.2 Maintenance Planning Application

In order to show how DSM can be used to construct an application service that introduces differentiated quality levels on top of a common pool of application resources, a Maintenance Planning application has been implemented.

The motivation for the development of this application is to show how DSM may be used by a service provider to introduce differentiated quality levels to each of its customers while the application service is implemented on top of a common set of application resources. Each customer request for the application service is prioritised according to the QoS requirements defined in the customer's SLA.

The application provides fault resolution capabilities to railway maintenance providers. The maintenance provider is contracted to provide maintenance services to a number of train operating companies and the outsourcing of the maintenance is governed by an SLA contract between the train operating company and the maintenance provider. The maintenance provider is responsible for ensuring train availability and minimising the amount of faults that occur on the train. The SLA contract defines penalties that are incurred by the maintenance provider when faults on a train occur. The maintenance provider maintains a number of maintenance crew and maintenance depots and must find the best allocation of resources and depot bays in order to best meet the SLA contracts across all of its customers to minimise penalties. In addition, parts must be ordered prior to the scheduling of resources to resolve the fault.

The Maintenance Planning application service simulates an application that allocates maintenance personnel (resources), allocates parts, and schedules depot availability in order to resolve the faults in a cost effective way that minimises penalties incurred due to service violation. The application demonstrates that, by using DSM, the SLA parameters defined in the SLA contracts are used during the allocation of Web service resources as required by the fault maintenance business process. DSM evaluates the QoS parameters of each of the Web service resources and through the QoS aggregation techniques provided by DSM, evaluates the financial impact and SLA compliance level as the DSM Scheduler allocates Web service resources to customer requests.

The application is defined as a business process which integrates the different components of the maintenance planning system. The business process is illustrated as a state transition diagram in Figure 6-4.

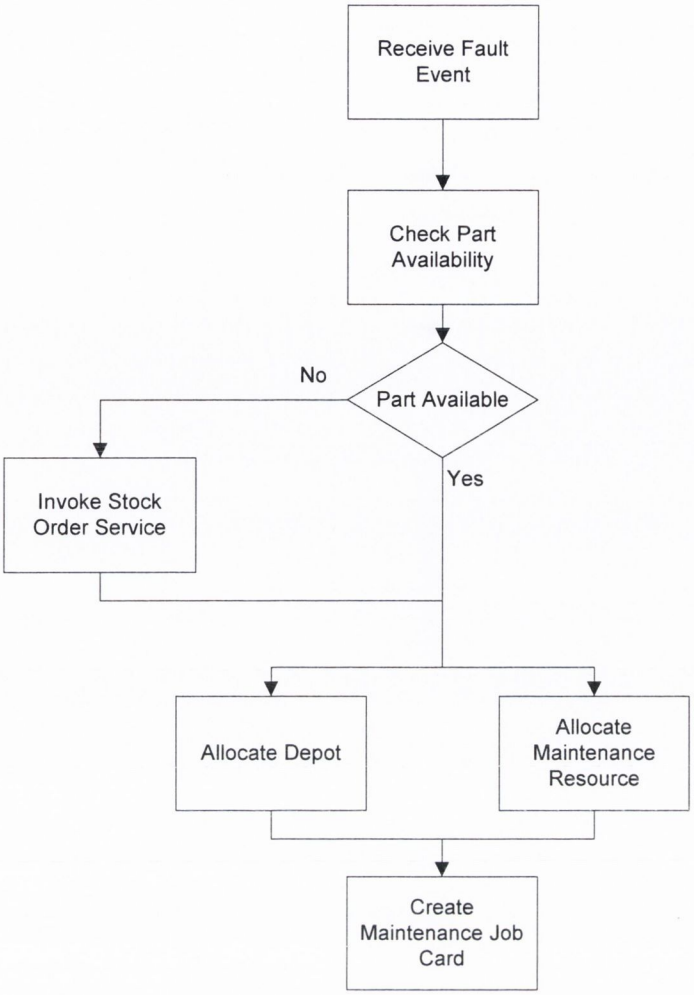


Figure 6-4 Maintenance Planning Application Process Definition

A fault is received from a customer by the maintenance service provider as a fault event. In this application prototype simulation, three different fault types are supported. These faults are described in Figure 6-5.

Fault Type	Description
`Fault_Air_Conditioning`	A fault has occurred in a train air conditioning system. Maintenance Personnel Required: Electrical Part Type Required: A/C Unit Response Time: 2 hours
`Fault_Brakes`	A fault has occurred with the train brake system. Maintenance Personnel Required: Mechanic Part Type Required: Brake Unit Response Time: 30 minutes
`Fault_Doors`	A fault has occurred with the train doors. Maintenance Personnel Required: Mechanic Part Type Required: Door Control Unit Response Time: 20 minutes

Figure 6-5 Maintenance Planning Application Fault Types

The maintenance planning application is able to a new receive maintenance event every 30 seconds. When the DSM Service Execution Manager receives the event and triggers a new business process instance in the business process manager. The service invokes the `checkPartAvailable` web service to verify that the required part for the fault type is available. This Web service typically invokes a stock management system and in this simulation has been implemented as a Web service interface to query a database for part availability. If the part is not available, the condition defined in the business process evaluates to false, and the `invokeStockOrder` is invoked by the business process. This activity invokes the Web service provided by a partner service and creates an order for the required part. The *Endpoint Description* for this Web service defines the *time* parameter for this service. The time in this instance is the time required for the part to be available in the depot. The business process defines that the part must be available before the maintenance can actually be carried out.

In this prototype simulation, it is assumed that a number of parts are maintained by the depot. Therefore, the probabilistic weighting, as defined in Section 4.4.1, is for the `Part Available` predicate to evaluate to true. This is used by the DSM Execution

Planner when creating the list of schedulable activities. The default execution path will assume that the maintenance can therefore be scheduled for maintenance immediately. Should the `Part Available` predicate evaluate to false, the DSM Orchestration Service receives this process execution event from the Business Process Interface and calls the scheduler to re-schedule the revised execution path.

Once the required part is available, the DSM Scheduler creates a schedule for the execution of `Allocate_Depot` and `Allocate_Maintenance_Resource` activities in the business process definition. These two activities will execute in parallel, as the train must be allocated a maintenance bay in the depot for the maintenance personnel to carry out the maintenance work. The `Allocate_Depot` is an abstract Web service definition for the allocation of allocation of a maintenance bay in the depot for the fault to be repaired. This application defines three separate Web service implementations for this service type, which each provide the functionality to allocate a maintenance bay to a particular maintenance fault. Each of these Web service implementations represent the allocation of a depot bay for the maintenance to be carried out. Each of the implementations are registered with the DSM Resource Broker and dynamically assigned to service requests by the DSM Execution Manager.

The `Allocate_Maintenance_Resource` abstract web service definition is executed in parallel to the `Allocate_Depot` web service and is responsible for assigning particular maintenance personnel to the fault maintenance. This application simulation defines three personnel types that are available for scheduling. Each of these personnel types have a cost, duration to fix a typical fault, and an availability, which indicates the number of personnel available in each type that can be scheduled. For the purposes of this simulation it is assumed that only one resource is required to fix each fault. The Web service implementations represent calls to a *resource management system* which result in the allocation of a particular resource type to a *maintenance job*. For this simulation, the Web service implementations represent an interface to a resource database which is used to capture resource allocation.

The Maintenance Planning application illustrates how DSM can be used for the management of applications in a domain very different to that presented in the Loan Assessment application described in section 6.2.1 of this chapter. DSM provides the following functionality in order to meet the applications goals:

1. The application uses the DSM framework to create a new application service that is constructed from the existing systems implemented in the maintenance provider environment. DSM enables the maintenance provider to offer the application service with associated QoS guarantees, which are calculated by DSM as the aggregation of the QoS metrics from each of the resource Web services that make up the planning application.
2. DSM enables the maintenance provider to offer the application to each customer with different QoS levels. Each customer is further able to specify the QoS requirements of each fault type in the planning application and DSM allocates Web service resources to meet these requirements.
3. DSM dynamically allocates Web service resources to each customer request and can re-schedule the allocation of resources according to the execution path of the application business process. In the planning application, DSM re-schedules the depot and maintenance personnel according to part availability.

6.2.3 Analysis of Trials

This section presents an analysis of the trials that were carried out during the implementation of the applications described in the previous section. The two applications were selected in order to demonstrate the fulfilment of the requirements imposed on the DSM framework during the design process, which were described in section 4.1 of this thesis.

An initial observation of the tests shows that DSM is suitable for the construction of application services where the service resources are optionally available with different quality levels. In the case of the maintenance planning application the application resources are available with one quality level whereas the loan assessment application

illustrates how DSM can use application resources with different defined levels of quality, as in the Acxiom Web services. In this case, three distinct service resources are available to DSM to allocate to customer requests, depending on the end-to-end QoS requirements. In both applications DSM enables the service provider to offer the composite application service with different levels of quality to its customers. Furthermore, it can be seen from the trials, that DSM enables the service provider to easily add new service resources should this be required, without making any changes to the application service. This is demonstrated in the loan assessment application in which the service provider can easily add (or remove) instances of the assessment model Web service, depending on the current customer load.

The maintenance planning application shows how DSM can dynamically re-schedule application resources as the business process defining the application service executes. If it is found that the stock items required to carry out the maintenance tasks are not available, the personnel resources required to carry out the task are rescheduled for a time when the materials required are received into the stores from the supplier. During the execution of the planning process, if it is found that there is a further delay in receiving the stock item from the supplier, a further re-scheduling of the maintenance personnel is carried out.

Through an analysis of the trials conducted, it can be seen that the DSM framework is ideally suited for the development of applications which can be constructed as a structured composition of service resources through a business process definition. This application type enables DSM to dynamically allocate service resource instances to each of the activities in the process definition, taking the end-to-end QoS parameters of the business process instance into account. While DSM may be utilised in the development of simpler applications that are not designed of a business process definition, but can be fulfilled as a single Web service resource, the benefits of the end-to-end aggregate QoS evaluation and resulting dynamic resource allocation provided by DSM would not be fully utilised.

The prototype applications discussed illustrate that a number of prerequisites must be provided by the application developer as a part of the development of the service provider application. These are required in order for DSM to effectively manage the

execution of the applications in the runtime. In particular, SLA template and an SLO information model must be specified at application development time, which defines the SLA QoS metrics that can be offered to the customers of the service, along with the aggregation algorithms for the quantifiable measurement data for the service resources that implement the service. Should the definition of the application change, the SLO information model must also be updated accordingly. However, once the SLA template and associated SLO information model have been specified during application development, DSM automates the subsequent runtime execution of the application.

6.3 Comparison with other Service Management Architectures

A number of different architectures for service management found in the literature were presented in chapter 3. This section presents a comparison between these architectures and DSM.

The approach adopted by a number of the frameworks, including IBM SLA Framework [63], Conformance [17], and CrossFlow [30] provide SLA management for fixed resources and, unlike DSM, do not address the separation of resources from service delivery. The approach adopted by these frameworks is to provision dedicated resources to each customer during the SLA establishment phase. This approach makes these frameworks unsuitable for use in a dynamic environment, where the service implementation is separated from the service offering and can be changed by the service provider.

The IBM WSLA Management Framework [63] adopts a utility computing [44] approach, where Web service resources are bound dynamically to each customer request according to SLA requirements. To achieve this, the framework has built a set of queues of differing priorities to which service requests are added, depending on the priority defined in the SLA. A weighted round-robin technique is used to process the queues and a Global Resource Manager adjusts the weightings periodically. This

approach provides some service differentiation, however the differentiation is restricted by the limit on the number of queues. Furthermore, this approach does not look to optimise the allocation of resources to requests in order to minimise SLA violations, as it is constrained as request are processed according to the round robin algorithm. The WSLA Management Framework also does not consider composite services and the aggregation of QoS parameters from resource level measurement. All of these issues are overcome in the DSM Scheduler which uses an optimising algorithm for resource allocation.

SAM [21] provides a service management framework that aims to minimise the penalties incurred due to SLA violations. The approach adopted by SAM is to evaluate the service management tasks that arise from SLA violations and the scheduler developed in SAM contains a number of optimising algorithms to evaluate and prioritise the events in order to minimise penalties. However, SAM, unlike DSM, does not address the issue of scheduling resource allocation to service requests in order to minimise SLA violations. It only looks to minimise penalties once an SLA violation has occurred. It does not provide service management techniques such that the services are fulfilled in a way that minimises SLA violations.

WSAF [78] supports dynamic resource allocation through the use of agent technology. The framework however does not consider service offering that are created as a composition of service provider resources and as such, the aggregation of service QoS parameters from resource level measurement is not supported. Unlike DSM, the framework does not evaluate SLA compliance over time and only allocates resources according to customer requirements pre-defined in a customer policy.

Oceano [14] also provides support for dynamic resource management assignment through an event driven architecture. However, the mechanisms available to Oceano are limited to providing additional computing power to accommodate an increased load on a service, or *throttling* the number of requests that can be processed. It does not address resource assignment to individual customer requests based on SLA targets.

The Web Service Composition Framework (MWSCF) which is a part of the METEOR-S [4] framework provides support for semantic composition of Web services. However, the MWSCF requires that the QoS requirements be specified for each activity of the business process template. It does not provide support for dynamic resource allocation based on SLA requirements for the composite service. This is overcome in the DSM Scheduler which uses an optimising algorithm for resource allocation based on the overall requirements of each SLA.

Figure 6-6 is a revised version of the table showed during the analysis of the state of the art in the area of dynamic service management, which was presented in section 3.6. This new version of the table adds DSM to the list of service management frameworks, and illustrates that DSM has a number of features that is not provided by the other architectures found in the literature.

		IBM Service Manager	Conformance	SAM	WSLA Framework	METEOR-S	CrossFlow	Oceano	WSAF	DSM
Dynamic Services	Composite Services					✓	✓			✓
	SLA Management	✓	✓	✓	✓		✓	✓		✓
	Dynamic Resource Assignment			✓	✓	✓			✓	✓
Differentiated levels of Service	Customer Specific Service Levels	✓		✓			✓		✓	✓
	Service Customisation	✓	✓				✓		✓	✓
Optimal Provision of Service	Dynamic Resource Provisioning				✓	✓		✓		✓
	SLA Driven Delivery Optimisation			✓					✓	✓

Figure 6-6 Comparison of DSM with other Service Management Architectures

DSM is unlike the above frameworks as it designed as a service management framework for managing composite services where resources are bound dynamically. It supports the aggregation of measurement data from the resource level to calculate

QoS metrics which are defined at the composite service level. Furthermore, it manages the allocation of resources to composite services such that service subscriptions with differing QoS requirements are honoured.

6.4 Summary

This chapter presented an evaluation of the DSM framework that is proposed in the thesis. The goals of this thesis were summarised and the approach to evaluating these goals was outlined. A detailed description of two applications built using DSM was given. This was followed by a detailed discussion on how the requirements outlined in chapter 1 are fulfilled by DSM. Finally, a comparison between DSM and a number of service management frameworks found in the literature was presented. This chapter has shown that DSM provides a number of improvements when compared to other frameworks found in the literature.

Chapter 7
Conclusion

This thesis presents the concepts and models provided by a framework, called DSM, for providing management services to outsourced business processes on the Internet. DSM provides management services to service providers, enabling them to rapidly create new dynamic services that are offered at differentiated levels of quality to each customer. DSM also provides novel techniques to facilitate the composition of dynamic services, enabling service providers to offer differentiated services in a cost effective manner.

This chapter summarises the contents of this thesis and presents an overview of the goals of the thesis and how each of these goals were fulfilled. An outline of the thesis contribution to the state of the art is also given. Finally the thesis is concluded with a discussion of some open research issues.

7.1 Thesis Summary

The motivation for the work described in this thesis arose from an analysis of the state of the art in service management. While a number of researchers have considered support for management of dynamic services, research in this area has primarily resulted in frameworks that that either do not address the issues of dynamically composed services that are an aggregation of services resources, and the dynamic

provisioning of resources to customer requests in order to meet the QoS requirements of each customer in a cost effective manner.

Accordingly, this thesis proposed a framework for dynamic service management that enables a service provider to create service offerings which are constructed as an orchestration of existing Web services, can be offered at different levels of quality to each customer, and manages to allocation of the Web service resources dynamically in order to meet the QoS requirements of each customer in a cost effective manner. A description of the prototype implementation of this framework, called DSM, was also given.

The thesis is structured as follows:

Chapter 1 provides some background on dynamic service management and moves on to describe and motivate the goals of this thesis. The important goals identified are: that the framework proposed in this thesis should enable the dynamic outsourcing of services to service providers; should enable service providers to offer differentiated levels of service to each customer; and provide the service provider with mechanisms that fulfil service requests in a cost effective manner.

In chapter 2 an overview of the research area is presented. The chapter is divided into four main sections. The first section presents an overview of virtual enterprises and details the issues that arise in the outsourcing of services in a virtual enterprise environment; the second section presents a description of service level agreements; the third section presents an overview of service management, and the describes the issues associated with dynamic service management. Finally a description of web services and web service composition is presented.

In chapter 3 a survey of dynamic service management frameworks is presented. Each of these systems is reviewed according to their software architecture and support for dynamic services.

In chapter 4, a framework designed to meet the goals of the thesis is presented. The chapter firstly presents a description of the goals of the thesis and the requirements of

the framework presented in this thesis. A description of the framework architecture is then presented. The Service Management Component (SMC) for the management of each SLA contract is discussed. Through the use of the SMC, the QoS metrics that are measured at the resource level are aggregated to evaluate the SLA commitments of the application service. The scheduling algorithm which enables the dynamic assignment of resources is presented. Finally resource management is discussed.

The design and implementation details of DSM are presented in chapter 5. The chapter is divided into four main sections. The first section outlines the design of the Service Manager and the Service Management Component (SMC); the second section describes the Service Execution Manager; the third section describes the Resource Broker component. Finally a detailed description of the implementation is given.

Chapter 6 evaluates the thesis. The chapter firstly presents a summary of the goals of the thesis. Following on from this, a description of two applications built using DSM is presented. The first of these applications, a Loan Assessment application, is an online application service to financial institutions to evaluate mortgage loans and predict pre-payment of the loan. Following on from this a description of a second application, a Maintenance Planning application, built using DSM is presented. This application provides fault resolution capabilities to railway maintenance providers. The chapter then presents a comparison between DSM and other service management frameworks that are found in the literature and discussed in chapter 3. Finally, a discussion on how the DSM framework meets the thesis goals and requirements is given.

7.2 Achievements

A set of goals for this thesis were identified in chapter 1. In chapter 6 a study of these goals and how each of these goals was achieved was presented. The three principal achievements documented in this thesis along with the evaluation approach are presented here.

The first goal was that the framework should enable the dynamic outsourcing of services to service providers. DSM enables service providers to easily create new service offerings through the use of existing resources that are implemented by the service provider. DSM enables the service provider to offer the service at defined levels of quality in order to meet customer expectations. This was demonstrated by describing the construction of two applications using DSM, a Loan Assessment application and a Maintenance Planning application.

The second goal was that that framework should enable service providers to offer differentiated levels of service to each customer. DSM enables the service provider to customise SLA contracts according to specific customer requirements while still adopting a dynamic provisioning approach to resource allocation, where resources are dynamically assigned to customer requests from a common pool.

The third goal was that the framework should provide the service provider with mechanisms that enable the fulfilment of service requests in a cost effective manner. DSM adopts a dynamic resource allocation approach for service fulfilment and manages the allocation of Web service resources and the prioritisation of service requests in order to meet the obligations defined in the customer SLA contracts. DSM incorporates a scheduling algorithm to allocate resources to customer requests in a cost effective manner.

This thesis makes a number of additional contributions to the state of the art in dynamic service management. By adopting an object representation of each SLA contract in the Service Management Component, DSM is able to aggregate measurement data and continually monitors the compliance of each SLA. This approach is then used by the scheduling algorithm adopted by DSM to dynamically allocate resources to best fit the needs of customer requests.

7.3 Thesis Contributions

This section discusses how the thesis goals and requirements that were presented during the design phase in chapter 4 and outlined in chapter 1 are fulfilled by the DSM framework. These goals and requirements include the ability to support dynamic services, provide those services at different levels of quality to each customer as required, and manage the fulfilment of those services in the most cost effective manner according to business requirements. Each of these requirements is discussed in the following sections.

1. **Enable the dynamic outsourcing of services to service providers.**

DSM enables the service provider to easily create new applications as customer requirements change. It leverages the service oriented computing [65] paradigm, enabling the service provider to separate the service offering from the implementation of the underlying service components. The DSM Service Manager enables the service provider to define services as an orchestration of component Web services. This orchestration, expressed as a business process, references the functionality required by each of the business process activities in abstract terms. The DSM Execution Manager supports late-binding of resources to executing business process instances. The service provider therefore does not have to provide dedicated resources to each customer subscription, which adds significant cost and management requirements. DSM enables the service provider to operate in a more flexible, where resources are allocated to customers dynamically, depending on the current requirements. The Loan Assessment application described in section 6.2.1.1 of the previous chapter illustrates how the service provider can change the allocation of resources that are allocated to the assessment model Web service resource in order to best meet the current demand.

DSM enables the service provider to offer the services at defined levels of quality in order to meet customer expectations. The DSM Service Manager associates an SLA template with each service offering. The SLA defines the service level targets that are required to be met when fulfilling service requests. DSM manages the life-cycle of each customer SLA in order to minimise the overhead to the service provider. The

DSM Service Management Component (SMC) provides runtime management of each SLA that has been agreed upon. The SMC tracks the current status of each SLA and the fulfilment of the service defined by the SLA. The DSM Service Execution Manager uses the information captured by the SMC when allocating resources to service requests in order that the service level targets are met as best as possible. The SMC further enables the service provider to easily report on the status of each SLA as the SMC captures all of the information required for customer reporting.

2. Enable service providers to offer differentiated levels of service to each customer

As discussed in section 4.1.2, a key requirement for SLA management is the ability to support non-trivial customer-specific customisation of the service provider's offerings in order to accommodate a customer's unique data or business process management needs. This adds significant complexity for a service provider to provide the service in a cost-effective manner as the number of SLA contracts grow. Application examples described in section 6.2 of the previous chapter show that DSM enables the service provider to customise SLA contracts according to customer requirements. SLA templates are associated with each service offering, which allow both the specific service level targets and also the penalties to be imposed for service violations to be specified for each customer subscription.

The DSM Service Execution Manager uses a dynamic resource provisioning approach to service fulfilment, whereby resources allocated to service requests are taken from a resource pool common to all customers. DSM has adopted this approach in order to minimise cost and management overhead which are typically associated with dedicated per-customer resource provisioning. The DSM Service Execution Manager allocates Web service resource instances to service requests dynamically on order to meet the SLA obligations for that request.

Evaluation of the QoS metrics defined in each SLA contract is carried out automatically by the Service Management Component corresponding to each SLA. The SMC receives measurement data from the execution of each of the resources used

during the execution of a service request. The applications discussed previously in chapter 6 used Web service filters to capture execution data for each of the Web services. DSM also supports the capture of measurement data from management software that has been deployed by the service provider, or from third-party *management service providers* [63] that may be specified in the SLA contract. The measurement data captured at the resource level is aggregated by the SMC to calculate the QoS metrics that have been defined in the SLA.

3. Provide the service provider with mechanisms that fulfil service requests in a cost effective manner

DSM adopts a dynamic resource allocation approach for service fulfilment. This leverages a Web service resource pool common to all customers in order to reduce management costs. In order to provide differentiated levels of service to each customer, DSM manages the allocation of Web service resources and the prioritisation of service requests in order to meet the obligations defined in the customer SLA contracts. For each service request, the Execution Planner component derives a list of the activities that are most likely to execute in the business process. This drives the Scheduler component, which allocates resources and schedules execution of each of the activities across all currently executing service requests. The Scheduler uses the SMC of each service request to evaluate the QoS metrics at the application level for each resource allocation. Using this approach, the Scheduler allocates resources to each service request which best fits the quality requirements of each of the SLA contracts, whilst minimising the cost to the service provider for fulfilling the service requests.

The Loan Assessment application discussed earlier in this chapter also illustrates how DSM supports the implementation of Web service resources with different quality characteristics. In this example, the Acxiom Web service is provided in three different quality levels, with each level having a different associated cost. The service definition need only reference the abstract resource Web service type and not an actual Web service instance. DSM automatically allocates the resource instances that meet the QoS targets at the lowest cost to the service provider.

7.4 Open Research Issues

In the course of the research that led to this thesis, a number of issues were uncovered that were outside the scope of this thesis but could lead to possible new and exciting research. One such area is the integration of the DSM Service Execution Manager as a part of the underlying business process engine. The current design of DSM is such that it uses a business process manager to execute the Web service business processes. The business process engine is invoked by DSM through a Business Process Interface. This provides some value in terms not tying DSM to a particular business process engine instance, and allows different business process engine implementations to be plugged into DSM. However, typically the business process engine also contains a scheduler component which executes the rules defined in the business process definition. Some further improvement of the scheduling of Web services could potentially be achieved through the integration of the two separate scheduler components.

A second area worthy of further investigation is the introduction of contract negotiation techniques during the SLA establishment phase with a customer. DSM facilitates the agreement of an SLA contract between the service provider and service customer through the use of SLA templates, but the process still requires both parties to agree the terms of the SLA parameters. By introducing negotiation techniques, this process could potentially be automated, further enabling the dynamic outsourcing of services.

7.5 Summary

This chapter summarised the motivation for and the most significant achievements of this thesis, followed by a discussion of the contributions of the thesis. The chapter was concluded with some suggestions for possible future work arising from the research undertaken as part of this thesis.

Appendix A

Related Publications

The following is a list of previous related papers written by the author. These papers are available from the Computer Science Department Website (www.cs.tcd.ie).

O'Connell, M. and P. Nixon. Next Generation Business-to Business E-Commerce. in *First International Conference on Electronic Commerce and Web Technologies*. 2000. London, UK.

Nixon, P., V. Wade, S Terzis, M. O'Connell and S. Dobson. The VIRTUES Architecture: A Software Infrastructure for Business-to-Business E-Commerce. in *2nd International Conference on Enterprise Information Systems*. 2000. Stafford, UK.

O'Connell, M. and P. Nixon. JFS: A Secure Distributed File System for Network Computers. in *the IEEE Euromicro Conference on Network Computing*. 1999. Milan, Italy.

O'Connell and P. Nixon. A synthesis of concepts for the representation of business processes in Virtual Enterprises. Technical Report, Trinity College Dublin. TCD-CS-2000

Appendix B

Glossary of Abbreviations

B&B	Branch & Bound Algorithm
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPI	Business Process Service Provider Interface
CIM	DMTF Common Information Model
DeB	Dynamic e-Business
DMTF	Distributed Management Task Force
DSM	Dynamic Service Manager Framework
eTOM	Telemanagement Forum Enhanced Telecom Operations Map
HTTP	Hypertext Transfer Protocol
MSP	Management Service Provider
OS	Operating System
QoS	Quality of Service
SLA	Service Level Agreement
SLO	SLA Object

SMC	Service Management Component
SNMP	Simple Network Management Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPI	Service Provider Interface
TMF	Telemanagement Forum
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
VE	Virtual Enterprise
WSDL	Web Service Definition Language
WSIF	Web Service Invocation Framework
WSLA	Web Service Level Agreement

Appendix C

Resource Optimisation Model

Optimisation Algorithm

The Resource Optimisation Model provides a scheduling algorithm to dynamically allocate service provider resources to customer requests in such manner that the costs of service implementation are minimised for the service provider. The scheduler uses the branch and bound (B&B) algorithm [93], which is the main technique for solving integer and discrete programming problems. The B&B method is based on the observation that the enumeration of integer solutions can be organised into a tree structure. B&B is an efficient algorithm in that it is not necessary to construct the entire tree in order to find an optimal solution. Instead, B&B expands the tree in stages, expanding the most promising node at each stage and discarding nodes permanently when it can be evaluated that neither the node, nor any of its descendents, will ever be either feasible or optimal.

The original problem, referred to as the *root problem* is bounded from above and below. If the bounds match, the optimal solution has been found. Otherwise, the feasible region, i.e., the space in which in which the problem in confined by explicit constraints, is partitioned into sub-regions. The sub-regions constitute feasible regions for sub-problems, which become children of the root problem in the search tree. The principle behind creating relaxed sub-problems of the original problem (known as branching) is that unlike the original problem, the relaxed sub-problems can be solved within a reasonable amount of time. If a sub-problem can be optimally solved, its solution is a feasible, though not necessarily optimal, solution to the original problem.

Therefore, it provides a new upper bound for the original problem. Any node of the search tree with a solution that exceeds the global upper bound can be removed from consideration, i.e., the branching procedure will not be applied to that node.

The Branch & Bound algorithm is structured as follows [57]:

Let P denote the abstract problem. Let $P_{i.value}$ denote the optimal solution to the instance P_i of problem P. Further, let R be the relaxation of P such that $R_{i.value}$ bounds $P_{i.value}$ from below. It is important that $R_{i.value}$ may be computed at all stages. Further, a branching rule for breaking up the feasible set of the currently analysed instance of the relaxation R_i is defined. Finally, a rule for choosing the next relaxed problem to be processed (selection rule) is defined. The Branch & Bound algorithm is then given by:

1. Put P_i on the active list of problems. Initialise the upper bound $u=\infty$. Set the current best solution to be null.

2. If the list of active problems is empty, then stop: the solution associated with u is optimal (or if $u=\infty$, P_i has no solution). Otherwise choose a sub-problem of P_i according to the sub-problem selection rule and remove P_i from the list.

3. Solve the relaxation R_i of P_i and let $l_i = R_{i.value}$. If $l_i > u$ then return to step 2. If $l_i < u$ and the solution of R_i is also a valid solution of P_i then set the current best solution to be $R_{i.solution}$ and $u = R_{i.value}$ and go to step 4.

4. Apply the branching rule to P_i, i.e, generate new sub-problems $P_{i1}, P_{i2} \ldots P_{iq}$, place them on the list of active problems and go to step 2.

In DSM, the branching rule allocates a resource (Web service) to a service task (activity in a business process definition) for a particular node in the search tree. The search of the tree is terminated when there are no further nodes to be expanded – i.e., all tasks have been assigned to a resource instance and no other nodes in the search tree can produce a solution that is better than the complete feasible solution that has already been found.

Optimisation Metric

The Resource Optimisation Model developed in DSM allocates resources to jobs in such a way as to minimise the overall cost of the service delivery for the service provider. The cost of delivery as measured by DSM includes both the usage cost of the resources and also penalty costs for breaching service level guarantees in the SLAs. The Resource Optimisation Model evaluates each node in the search tree in terms of both the feasibility of the solution with the particular task / resource assignment and also the cost of service delivery of that particular assignment. The upper bound to the problem is defined as the overall financial cost of fulfilling each customer request.

Model Constraints

The Resource Optimisation Model imposes a number of constraints on the problem solution which are used when evaluating the feasibility of a node in the search tree.

The constraints imposed on the model solution are described as follows.

Task Precedence

For each customer request, the service implementation is composed of a number of tasks, whose precedence ordering is defined by the business process definition for the service. The Task Precedence constraint ensures that the tasks are scheduled in the correct order according to the business process definition and the starting time of a task must be after the completed time of the preceding task.

QoS Metrics

A number of QoS parameters are defined for the service that is requested by the customer. The individual QoS metrics of the resource is evaluated to ensure that it meets the end-to-end QoS requirements of the application service. If it does not meet

the requirements, this particular task / resource assignment does not form a valid solution and this particular branch of the search tree is not explored further.

Resource Capacity

Each of the service resources that are used to compose the application service being delivered to the customer can support only a limited number of concurrent requests, the number of which is specific to each service resource instance. This can be defined as either a total number in a given period, or a number of concurrent tasks that can be fulfilled. When allocating service resources to specific tasks, the Resource Optimisation Model must ensure that the concurrent capacity limitations of each service resource is not exceeded.

Dynamic Runtime Environment

One of the key requirements of the Resource Optimisation Model is that it must support the allocation of resources to customer requests in a dynamic runtime environment, where new customer requests are continually being received. In this environment, the set tasks to be scheduled are continually changing. The Resource Optimisation Model is an event driven scheduler that responds to either new customer requests that are received, or new activities that need to be scheduled when the execution path of a particular request has changed due to the evaluation of the predicates at conditional branches in the process flow. When an event is received, the scheduler *backtracks* along the search tree to a node where a partial viable solution to the problem is available. The scheduler then proceeds to evaluate the outstanding unallocated tasks, which now include the new or changed request tasks. In this way, a complete re-evaluation of the entire problem is not required each time a new request is received by DSM.

Appendix D

Sample WSLA Template

```
<?xml version="1.0"?>
<!--
Simple SLA Contract Template
-->
<SLA xmlns="http://www.ibm.com/wsla"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.ibm.com/wsla c:\thesis\wsla.xsd"
        name="TemplateSLA" >
        <Parties>
                <ServiceProvider name="provider">
                        <Contact>
                                <POBox>XXX</POBox>
                                <City>XXX</City>
                        </Contact>
                        <Action xsi:type="WSDLSOAPActionDescriptionType"
                                name="Notification" partyName="provider">
                                <WSDLFile>Notification.wsdl</WSDLFile>

        <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
                                <SOAPOperationName>Notify</SOAPOperationName>
                        </Action>
                </ServiceProvider>
                <ServiceCustomer name="customer">
                        <Contact>
                                <Street>XXX</Street>
                                <City>XXX</City>
                        </Contact>
                        <Action xsi:type="WSDLSOAPActionDescriptionType"
                                name="Notification" partyName="customer">
                                <WSDLFile>Notification.wsdl</WSDLFile>
```

```
<SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
                        <SOAPOperationName>Notify</SOAPOperationName>
                </Action>
        </ServiceCustomer>
        <SupportingParty name="ms" role="MeasurementServiceProvider"
                sponsor="provider">
                <Contact>
                        <Street>XXX</Street>
                        <City>XXX</City>
                </Contact>
        </SupportingParty>
</Parties>
<ServiceDefinition name="StockQuoteservice">
        <Operation xsi:type="wsla:WSDLSOAPOperationDescriptionType"
                name="WSDLSOAPGetQuote">
                <Schedule name="MainSchedule">
                        <Period>
                                <Start>2001-11-30T14:00:00.000-05:00</Start>
                                <End>2001-12-31T14:00:00.000-05:00</End>
                        </Period>
                        <Interval>
                                <Minutes>2</Minutes>
                                <Seconds>30</Seconds>
                        </Interval>
                </Schedule>
                <SLAParameter name="AverageResponseTime" type="float"
                        unit="seconds">
                        <Metric>averageResponseTime</Metric>
                </SLAParameter>
                <Metric name="averageResponseTime" type="double" unit="seconds">
                        <Source>ms</Source>
                        <Function xsi:type="wsla:Divide" resultType="double">
                                <Operand>
                                        <Function xsi:type="wsla:Plus"
                                                resultType="double">
                                                <Operand>

                                                <Metric>averageResponseTimeHost1
                                                </Metric>
                                                </Operand>
                                                <Operand>

                                                <Metric>averageResponseTimeHost2
                                                </Metric>
                                                </Operand>
                                        </Function>
                                </Operand>
                                <Operand>
                                        <LongScalar>2</LongScalar>
                                </Operand>
```

```
                </Function>
        </Metric>
        <Metric name="averageResponseTimeHost1" type="double"
                unit="seconds">
                <Source>ms</Source>
                <Function xsi:type="wsla:Mean" resultType="double">
                        <Metric>responseTimesHost1</Metric>
                </Function>
        </Metric>
        <Metric name="responseTimesHost1" type="TS" unit="seconds">
                <Source>ms</Source>
                <Function xsi:type="wsla:TSConstructor" resultType="TS">
                        <Schedule>MainSchedule</Schedule>
                        <Metric>responseTimeHost1</Metric>
                </Function>
        </Metric>
        <Metric name="responseTimeHost1" type="double" unit="seconds">
                <Source>ms</Source>
                <MeasurementDirective xsi:type="wsla:ResponseTime"
                        resultType="double">

                <MeasurementURI>http://ms.com/testResponse
                </MeasurementURI>
                </MeasurementDirective>
        </Metric>
        <Metric name="averageResponseTimeHost2" type="double"
                unit="seconds">
                <Source>ms</Source>
                <Function xsi:type="wsla:Mean" resultType="double">
                        <Metric>responseTimesHost2</Metric>
                </Function>
        </Metric>
        <Metric name="responseTimesHost2" type="TS" unit="seconds">
                <Source>ms</Source>
                <Function xsi:type="wsla:TSConstructor" resultType="TS">
                        <Schedule>MainSchedule</Schedule>
                        <Metric>ResponseTimeHost2</Metric>
                </Function>
        </Metric>
        <Metric name="responseTimeHost2" type="double" unit="seconds">
                <Source>ms</Source>
                <MeasurementDirective xsi:type="wsla:ResponseTime"
                        resultType="double">
                        <MeasurementURI>http://ms.com/testResponse
                        </MeasurementURI>
                </MeasurementDirective>
        </Metric>
        <WSDLFile>thesis.wsdl</WSDLFile>
        <SOAPBindingName>SOAPNotificationBinding</SOAPBindingName>
        <SOAPOperationName>getQuote</SOAPOperationName>
</Operation>
```

```
        </ServiceDefinition>
        <Obligations> <ServiceLevelObjective name="g1"
                serviceObject="WSDLSOAPGetQuote"> <Obliged>provider</Obliged>
                <Validity> <StartDate>2001-08-15:1400</StartDate>
                <EndDate>2001-09-15:1400</EndDate> </Validity> <Expression> <Predicate
                xsi:type="wsla:Less"> <SLAParameter>AverageResponseTime</SLAParameter>
                <Value>5</Value> </Predicate> </Expression>
                <EvaluationEvent>NewValue</EvaluationEvent> </ServiceLevelObjective>
                <ActionGuarantee name="g2"> <Obliged>ms</Obliged> <Expression>
                <Predicate xsi:type="wsla:Violation">
                <ServiceLevelObjective>g1</ServiceLevelObjective> </Predicate>
                </Expression> <EvaluationEvent>NewValue</EvaluationEvent>
                <QualifiedAction> <Party>customer</Party> <Action
                actionName="notification" xsi:type="Notification">
                <NotificationType>Violation</NotificationType>
                <CausingGuarantee>g1</CausingGuarantee>
                <SLAParameter>AverageResponseTime</SLAParameter> </Action>
                </QualifiedAction> <ExecutionModality>Always</ExecutionModality>
                </ActionGuarantee> WSLA Language Specification Version 1.0 92
                <ActionGuarantee name="g3"> <Obliged>ms</Obliged> <Expression>
                <Predicate xsi:type="wsla:Less">
                <SLAParameter>AverageResponseTime</SLAParameter> <Value>4</Value>
                </Predicate> </Expression> <EvaluationEvent>NewValue</EvaluationEvent>
                <QualifiedAction> <Party>customer</Party> <Action
                actionName="notification" xsi:type="Notification">
                <NotificationType>Violation</NotificationType>
                <CausingGuarantee>g1</CausingGuarantee>
                <SLAParameter>AverageResponseTime</SLAParameter> </Action>
                </QualifiedAction> <QualifiedAction> <Party>provider</Party> <Action
                actionName="notification" xsi:type="Notification">
                <NotificationType>Violation</NotificationType>
                <CausingGuarantee>g1</CausingGuarantee>
                <SLAParameter>AverageResponseTime</SLAParameter> </Action>
                </QualifiedAction>
                <ExecutionModality>OnEnteringAndOnLeavingCondition</ExecutionModality>
                </ActionGuarantee>
                </Obligations>
</SLA>
```

Bibliography

[1] Aalst, W.M.P.v.d. Modeling and Analysing Interorganisational Workflows. in *1998 •International Conference on Application of Concurrency to System Design (CSD'98)*. 1998. Fukushima, Japan.

[2] Acxiom Corpororation, http://www.acxiom.com,

[3] Agarwal, V., et al. A Service Creation Environment Based on End to End Composition of Web Services. in *WWW 2005*. 2005. Chiba, Japan.

[4] Aggarwal, R., et al. Constraint Driven Web Service Composition in METEOR-S. in *IEEE International Conference on Services Computing*. 2004.

[5] Akkiraju, R., et al. A Framework for Facilitating Dynamic e-Business Via Web Services. in *OOPSLA 2001*. 2001.

[6] Akram, M.S., B. Medjahed, and A. Bouguettaya. Supporting Dynamic Changes in Web Service Environments. in *ISOC 2003*. 2003: Springer-Verlag Berlin Heidelberg.

[7] Alonso, G., et al., Web Services. Concepts, Architectures and Applications. 2004: Springer-Verlag Berlin Heidelberg.

[8] Anderson, M., Workflow Interoperability - Enabling E-Commerce. White Paper. 1999, The Workflow Management Coalition

[9] Ankolenkar, A., et al. DAML-S: Web Service Description for the Semantic Web. in *First International Semantic Web Conference (ISWC)*. 2002. Sardinia, Italy.

[10] Apache Software Foundation. Apache Tomcat version 5.5.

[11] Apache Software Foundation, Axis SOAP Framework version 1.2. http://ws.apache.org/axis/

[12] Apache Software Foundation, Web Services Invocation Framework (WSIF) version 2.0. http://ws.apache.org/axis

[13] Appel, W., Towards the theory of Virtual Organisations: A description of their formation and figure., in *virtual-organization.net*. 1999

[14] Appleby, K., et al. Oceano - SLA Based Management of a Computing Utility. in *IEEE/IFIP International Symposium on Integrated Network Management Proceedings*. 2001.

[15] ASP Industry Consortium. Service Level Agreements. 2000. http://www.allaboutasp.org/builder.asp?cname=pr-14nov00a as of 06/21/2002.

[16] Bagchi, S., G. Kar, and J. Hellerstein. Dependency Analysis in Distributed Systems using Fault Injection: Application to Problem Determination in an e-commerce Environment. in *12th International Workshop on Distributed Systems: Operations and Management DSOM'2001*. 2001. Nancy France.

[17] Bhoj, P., S. Singhal, and S. Chutani, SLA Management in Federated Environments. ACM Computer Networks, 2001. **35**(1): p. 5-24.

[18] Booth, D., et al., Web Services Architecture. W3C Working Group Note, in *W3C Technical Reports and Publications*. 2004. http://www.w3.org/TR/ws-arch/

[19] Bouman, J., J. Trienekens, and M.v.d. Zwan. Specification of Service Level Agreements, Clarifying Concepts on the Basis of Practical Research. in *IEEE Conference Software Technology and Engineering Practice*. 1999. Putsburgh, Pennsylvania.

[20] Brown, A., G. Kar, and A. Kellar. An Active Approach to Characterising Dynamic Dependencies for Problem Determination in a Distributed Environment. in *7th International IFIP/IEEE Symposium on Integrated Management (IM 2001)*. 2001. Seattle, Washington, USA.

[21] Buco, M., et al. Managing eBusiness on Demand SLA Contracts in Business Terms Using the Cross-SLA Execution Manager *SAM*. in *Sixth International Symposium on Autonomous Decentralized Systems (ISADS'03)*. 2003: IEEE Computer Society.

[22] Buco, M., et al., Utility Computing SLA Management Based Upon Business Objectives. IBM Systems Journal, 2004. **43**(1): p. 159-188.

[23] Cardoso, J. and A. Sheth, Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS), 2003. **21**(3): p. 191-225.

[24] Casati, F., S. Ilnicki, and L.-J. Jin, eFlow: a Platform for Developing and Managing Composite e-Services. 2000, Hewlett Packard. HPL-2000-36

[25] Caswell, D. and S. Ramanathan, Using Service Models for Management of Internet Services. 1999, Hewlett Packard Laboratories. HPL-1999-43

[26] Cervantes, H. and R.S. Hall. Automating Service Dependency Management in a Service-Oriented Component Model. in *6th ICSE Workshop on Component-*

Based Software Engineering: Automated Reasoning and Prediction. 2003. Portland, Oregon, USA

[27] Chan, M.C., Y.-J. Lin, and X. Wang. A Scalable Monitoring Approach for Service Level Agreements Validation. in *2000 International Conference on Network Protocols (ICNP'00)*. 2000.

[28] Chen, M.Y., et al. Pinpoint: Problem Determination in Large, Dynamic Internet Services. in *International Conference on Dependable Systems and Networks (DSN'02)*. 2002.

[29] Crawford, C.H., et al., Toward an on demand service-oriented architecture. IBM Systems Journal, 2005. **44**(1): p. 81-107.

[30] CrossFlow Consortium, Crossflow - Architecture Description, in *Cross Organisational Workflow*, Y.H.I. Research), Editor. 1999. ESPRIT E/28635

[31] CrossFlow Consortium, Crossflow - Contract Requirements, U.o. Twente, Editor. 1999. ESPRIT E/28635

[32] Curbera, F., W.A. Nagy, and S. Weerawarana. Web Services: Why and How. in *OOPSLA 2001 Workshop on Object-Oriented Web Services*. 2001. Tampa, Florida, USA.

[33] Dan, A., et al., Web Services on Demand: WSLA-driven Automated Management. IBM Systems Journal, 2004. **43**(1): p. 136-158.

[34] DARPA Agent Markup Language Program (DAML), Ontology Web Language (OWL) version 1.1. 2004 http://www.daml.org/services/owl-s/1.1/

[35] Debusmann, M. and A. Keller. SLA-Driven Management of Distributed Systems Using the Common Information Model. in *8th IFIP/IEEE International Symposium on Integrated Network Management (IM 2003)*. 2003. Colorado Springs, Co, USA.

[36] Diao, Y., J.L. Hellerstein, and S. Parekh, Using Fuzzy Control to Maximize Profits in Service Level Management. IBM Systems Journal, 2002. **41**(3): p. 403-420.

[37] Distributed Management Task Force, Common Information Model (CIM) Core Model. White Paper. 2000, Distributed Management Task Force. DSP0111

[38] Distributed Management Task Force, Common Information Model (CIM) Specification Version 2.2. June 1999, Distributed Management Task Force

[39] Distributed Management Task Force, Understanding the Application Management Model CIM Version 2.7. 2003. DSP0140

[40] Drews, F., et al. Utility-Function based Resource Allocation for Adaptable Application in Dynamic, Distributed Real-Time Systems. in *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*. 2004: IEEE Computer Society.

[41] Ensel, C. and A. Keller. Managing Application Service Dependencies with XML and the Resource Description Framework. in *7th IEEE/IFIP International Symposium on Integration Network Management (IM 2001)*. 2001. Seattle, WA, USA.

[42] Equifax Inc, http://www.equifax.com,

[43] Falkowski, T. and S. Voss. Application Service Providing as Part of Intelligent Decision Support for Supply Chain Management. in *36th Hawaii International Conference on System Sciences (HICSS 03)*. 2002. Hawaii: IEEE Press.

[44] Farrel, A.D.H., et al. Performance Monitoring of Service-Level Agreements for Utility Computing Using Event Calculus. in *First International Workshop on Electronic Contracting (WEC-04)*. 2004: IEEE Computer Society.

[45] Fiedler, U., B. Plattner, and G. Alonso. Quality of Service in Business-to-Business E-Commerce Applications. in *Symposium on Reliable Distributed Systems*. 1999.

[46] First American Real Estate Solutions, www.firstam.com,

[47] Fung, C.K., et al. Extending Business Process Execution Language for Web Services with Service Level Agreements Expressed in Computational Quality Attributes. in *38th Hawaii International Conference on System Sciences*. 2005: IEEE.

[48] Garey, M.R. and D.S. Johnson, Computers and Intractability. A Guide to the Theory of NP-Completeness. 1979: W H Freeman and Company, New York.

[49] Geise, H. Contract-based Component System Design. in *33rd Hawaii International Conference on System Sciences*. 2000. Hawaii.

[50] Hasselmeyer, P. Managing Dynamic Service Dependencies. in *12th International Workshop on Distributed Systems: Operations and Management DSOM'2001*. 2001. Nancy France, October.

[51] Hegering, H.G., S. Abeck, and B. Neumair, Integrated Management of Networked Systems - Concepts, Architectures and their Operational Application. 1999: Morgan Kaufmann.

[52] Hoffner, Y. Supporting Contract Match-Making. in *Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises*. 1999.

[53] Hung, P.C.K., Specifying Conflict of Interest in Web Services Endpoint Language (WSEL). ACM SIGecom Exchange, 2002. **3**(3): p. 1-8.

[54] IBM, et al., Business Process Execution Language for Web Services version 1.1. 2005. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/

[55] ITAA. ITAA's Application Service Provider (ASP) Service Level Agreement (SLA) Guidelines. 2000. http://www.itaa.org/asp/itaasla.pdf as of 06/21/2002.

[56] Jensen, W., W. Steenbakkers, and H. Jagers. Electronic Commerce and Virtual Organisations. in *2nd International VoNet Workshop. Organisational Virtualness and Electronic Commerce*. September 23-24 1999. Zurich.

[57] Kalczynski, P., A Java Implementation of the Branch and Bound algorithm: The Asymmetric Traveling Salesman Problem. Journal of Object Technology, 2005. **4**(1): p. 155-163.

[58] Kar, G. and A. Keller. An Architecture for Managing Application Services over Global Networks. in *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies INFOCOM 2001* 2001. Anchorage, AK, USA.

[59] Kar, G., A. Keller, and S. Calo. Managing Application Services over Service Provider Networks: Architecture and Dependency Analysis. in *Seventh IEEE/IFIP Network Operations Management Symposium (NOMS 2000)*. 2000. Honolulu, HI.

[60] Kearney, R., et al., Electronic Service Level Agreement (eSLA) for Application Hosting, in *IBM Research Report*. 2001, IBM Research. 99002

[61] Keller, A., An Approach for Managing Service Dependencies with XML and the Resource Description Framework, in *IBM Research Report*. 2002, IBM. RC 22307

[62] Keller, A. and G. Kar. Dynamic Dependencies in Application Service Management. in *2000 International Conference on Parallel and Distributed*

Processing Techniques and Applications (PDPTA 2000). 2000. Las Vegas, NV, USA: IEEE Press.

[63] Keller, A., et al., Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture. 2001, IBM Research. RC22162 (W0109-001)

[64] Keller, A. and H. Ludwig, The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, Special Issue on E-Business Management, 2003. **11**(1).

[65] Khalaf, R., N. Mukhi, and S. Weerawarana. Service-Oriented Composition in BPEL4WS. in *World Wide Web 2003 Conference, Web Services Track*. 2003. Budapest, Hungary.

[66] Klingemann, J., J. Wasch, and K. Aberer. Adaptive Outsourcing in Cross-Organisational Workflows. in *11th International Conference on Advanced Information Systems Engineering (CAiSE'99)*. June 1999. Heidelberg, Germany.

[67] Klingemann, J., J. Wasch, and K. Aberer. Deriving Service Models in Cross-Organisational Workflows. in *Ninth International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises*. 1999. Sydney, Australia.

[68] Leymann, F., Web Services Flow Language (WSFL 1.0) May 2001, IBM Software Group

[69] Leymann, F., D. Roller, and M.T. Schmidt, Web Services and Business Process Management. IBM Systems Journal, 2002. **42**(2): p. 198-211.

[70] Liu, Z., M.S. Squillante, and J.L. Wolf. On Maximizing Service-Level-Agreement Profits. in *EC'01*. 2001. Tampa, Florida: ACM.

[71] Ludwig, H. and Y. Hoffner. Contract-based Cross-Organisational Workflows. The CrossFlow Project. in *Workshop on Cross-Organisational Workflow Management and Co-ordination*. 1999. San Francisco, USA.

[72] Ludwig, H., et al., A Service Level Agreement Language for Dynamic Electronic Services. 2002, IBM Research. W0201-112

[73] Ludwig, H., et al., Web Service Level Agreement (WSLA) Language Specification, version 1.0. 2003, IBM Research. wsla-2003/01/28

[74] Ludwig, H. and K. Whittingham. Virtual Enterprise Co-ordinator - Agreement-Driven Gateways for Cross-Organisational Workflow Management. in *International Joint Conference on Work Activities Co-Ordination and Collaboration (WACC'99)*. 1999. New York: ACM.

[75] Lumpu, E., Z. Milosevic, and M. Sloman. Use of Roles and Policies for Specifying and Managing a Virtual Enterprise. in *9th International Workshop on Research Issues on Data Engineering: Information Technology for Virtual Enterprises - RIDE - VE '99*. 1999. Sydney, Australia.

[76] Ma, W., et al. Extending Apache Axis for monitoring Web Service Offerings. in *IEEE International Workshop on Business Services Networks*. 2005. Hong Kong.

[77] Mandell, D. and S. McIlraith. Adapting BPEL4WS for the Semantic Web: the Bottom-Up Approach to Web Service Interoperation. in *Proceedings of the Second International Semantic Web Conference (ISWC2003)*. 2003. Sanibel Island, Florida.

[78] Maximilien, E.M. and M.P. Singh, A Framework and Ontology for Dynamic Web Services Selection, in *IEEE Internet Computing*. September/October 2004. p. 84-93

[79] Maximilien, E.M. and M.P. Singh. Towards Autonomic Web Services Trust and Selection. in *ISOC 2004*. 2004. New York, USA: ACM.

[80] Microsoft Corporation, XLANG: Web Services for Business Process Design, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm

[81] Mowshowitz, A., Virtual Organization, in *Communications of the ACM*. 1997. p. 30-37

[82] Naik, V.K., S. Sivasubramanian, and S. Krishnan, Adaptive Resource Sharing in a Web Services Environment, in *Middleware 2004*, H.-A. Jacobsen, Editor. 2004, LNCS 3231. p. 311-330.

[83] Nexala Technologies, Nexala Business Process Engine version 4.2 Specification: http://www.nexala.com

[84] Ni, Q. and M. Sloman. An ontology-enabled Service Oriented Architecture for pervasive computing. in *International Conference on Information Technology: Coding and Computing (ITCC 2005)*. April 2005. Las Vegas, Nevada: IEEE Computer Society.

[85] Nierstrasz, O. and T.D. Meijler, Research Directions in Software Composition. ACM Computing Surveys (CSUR) 1995. **27**(2): p. 262-264.

[86] Nixon, P., et al. The VIRTUES Architecture: A Software Infrastructure for Business-to-Business E-Commerce. in *2nd International Conference on Enterprise Information Systems*. 2000. Stafford, UK.

[87] O'Connell, M. and P. Nixon. JFS: A Secure Distributed File System for Network Computers. in *the IEEE Euromicro Conference on Network Computing*. 1999. Milan, Italy.

[88] O'Connell, M. and P. Nixon. Next Generation Business-to Business E-Commerce. in *First International Conference on Electronic Commerce and Web Technologies*. 2000. London, UK.

[89] O'Connell, M. and P. Nixon, A synthesis of concepts for the representation of business processes in Virtual Enterprises. . 2000. Technical Report, Trinity College Dublin, TCD-CS-2000

[90] Ortner, W. and C. Stary. Virtualization of Organizations: Consequences for Workflow Modeling. in *32nd Hawaii International Conference on System Sciences*. 1999.

[91] Ouzounis, V. and V. Tschammer. A Framework for Virtual Enterprise Support Services. in *32nd Hawaii International Conference on System Sciences*. 1999.

[92] Ouzounis, V. and V. Tschammer. Integration of Electronic Commerce Business Processes in Virtual Enterprises. in *European Multimedia, Microprocessor Systems and Electronic Commerce, EMMSEC'98*. 1998.

[93] Papadimitriou, C.H. and K. Steiglitz, Combinatorial Optimization. Algorithms and Complexity. 1998: Dover Publications.

[94] Rodosek, G.D. and L. Lewis. Dynamic Service Provisioning: A User-Centric Approach. in *12th International Workshop on Distributed Systems: Operations and Management DSOM'2001*. 2001. Nancy France.

[95] Sahai, A., et al. Automated SLA Monitoring for Web Services. in *13th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management: Management Technologies for E-Commerce and E-Business Applications*. 2002: Springer-Verlag.

[96] Schmietendorf, A., R. Dumke, and D. Reitz. SLA Management - Challenges in the Context of Web-Service-Based Infrastructures. in *IEEE International Conference on Web Services (ICWS'04)*. 2004: IEEE Computer Society.

[97] Sheth, A. and K. Kochnut. Workflow Applications to Research Agenda: Scalable and Dynamic Work Coordination and Collaboration Systems. in *Workflow Management and Interoperability*. 1999: Springer Verlag.

[98] Sivashanmugam, K., et al., Framework for Semantic Web Process Composition. 2003, LSDIS Lab, Dept of Computer Science, UGA. Technical Report 03-008

[99] Sturm, R. and W. Bumpus, Foundations of Application Management. 1998: J. Wiley.

[100] Sun, H., et al. Research and Implementation of Dynamic Web Services Composition. in *5th International Workshop on Advanced Parallel Programming Technologies (APPT 2003)*. 2003. Xiamen, China: Springer-Verlag Berlin Heidelberg.

[101] Sun Microsystems, Java Servlet Specification. http://java.sun.com/

[102] Sun Microsystems, Java Swing Library. http://java.sun.com/

[103] TeleManagement Forum, Enhanced Telecom Operations Map (eTOM) The Business Process Framework. Addendum D: Process Decompositions and Descriptions. Release 5.0. April 2005. GB 921 D

[104] TeleManagement Forum, Enhanced Telecom Operations Map (eTOM). Addendum P: An eTOM Primer. Release 4.5. November 2004. GB 921 P

[105] TeleManagement Forum, Enhanced Telecom Operations Map (eTOM). The Business Process Framework Version 5.0. 2005. GB921

[106] TeleManagement Forum, SLA Management Handbook, Volume 1, Executive Overview. July 2004. GB 917-1

[107] TeleManagement Forum, SLA Management Handbook, Volume 2, Concepts and Principles. April 2004. GB 917-2

[108] Tolksdorf, R. A Dependency Markup Language for Web Services. in *Web, Web-Services, and Database Systems: NODe 2002, Web- and Database-Related Workshops*. 2002. Erfurt, Germany: Springer-Verlag

[109] UDDI Project, UDDI Technical White Paper 2000. http://www.uddi.org

[110] W3C, Web Services Architecture. W3C Working Group Note. 2004. http://www.w3.org/TR/ws-arch/

[111] WfMC, Workflow standard - Terminology and glossary. Technical Report EFMC-TC-1011. Version 2.0. 1996

[112] World Wide Web Consortium, Web Services Addressing (WS-Addressing). 2004. http://www.w3.org/Submission/ws-addressing/

[113] World Wide Web Consortium, Web Services Description Language (WSDL) version 1.1. 2001. http://www.w3.org/TR/wsdl