Lehrstuhl für Rechnertechnik und
Rechnerorganisation
der Technischen Universität München

Grid Resource Management with Service Level Agreements

Tianchao Li

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. A. Bode
Prüfer der Dissertation:	
	1. Univ.-Prof. Dr. H. M. Gerndt
	2. Univ.-Prof. Dr. M. Bichler

Die Dissertation wurde am 21.01.2008 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 02.07.2008 angenommen.

Abstract

With Grid computing adopting Service Oriented Architecture (SOA), resources are represented by services with standardized interfaces and the management of resources converges to the more general management of services (i.e. the Service Level Management). The key to Service Level Management is the management of Service Level Agreement (SLA), a formal contract between the provider and consumer of a service which stipulates and commits the provided service to a required level of service.

We concentrate in our research on designing and establishing an infrastructure for SLA management. It consists of a site-local infrastructure for managing SLAs and local resources, and a global infrastructure for advertising and brokering of resources.

From the client's point of view, a typical session of service usage includes the following activities: the client sends job parameters and SLA parameters (such as deadline, price etc.) to the broker, the broker contacts the service sites to get available SLA templates, the broker selects the appropriate template and sends it to the client for verification and an acknowledgement is represented by the return of a signed SLA offer. The SLA offer is then forwarded by the broker to the corresponding service site to create the SLA. With the negotiated SLA, the client invokes the service and gets an id (endpoint reference) of the job. According to the negotiated SLA, the job is scheduled at appropriate time and as soon as the job finishes, a notification is sent to the client so that the result can be retrieved.

On the service site, a service has an associated management infrastructure that handles the management of SLAs and resources. The WS-Agreement services provide interfaces for requests for SLA templates, SLA creation and termination, notification of SLA status changes etc. The performance predictor estimates resource demands for specific values of job parameters, and the resource manager keeps track of resource capabilities and job schedules. Based on the information provided by both, the agreement decision maker decides on the acceptance or rejection of a specific SLA offer. The resource manager (together with the underlying scheduler) is also responsible for the scheduling of jobs with resource reservation and service invocation at scheduled time.

Targeting to the SLA-based resource management infrastructure, we focus on the fundamental issues for the establishment of such an infrastructure, including the specification of SLAs with WS-Agreement and the design and implementation of a generic and extensible support infrastructure for WS-Agreement specification, the evaluation of various techniques for application performance prediction and the establishment of a generic infrastructure for non-intrusive monitoring and adaptive prediction of the execution time of

services, discussion and solution of job scheduling issues in the local resource management infrastructure with a focus on the cases where services have exclusive access to the local resource. In addition, we also present the design and implementation of a set of development tools and support environments for the development, management, and accessing of service-oriented Grid applications that are integrated with our SLA-based resource management infrastructure.

Acknowledgments

I would like hereby to express my gratitude to all that have made suggestions and offered help towards the successful completion of this work, and to all that have supported me from different aspects during the past years.

First of all I would like to thank my advisor, professor Michael Gerndt. He not only directed me to the right way for pursuing my work but also gave me the most considerate help throughout the whole course of this research work. I would like to express my special gratitude to professor Arndt Bode, the leader of our research chair, for all his support and encouragement. I am also grateful to him for his help in overcoming all the financial and administrative issues. I would also like to thank professor Martin Bichler for taking the time to review this thesis and giving valuable comments.

I want to express my gratitude to all my colleagues and the technical and administrative staff at LRR-TUM. I would like to specially mention Dr. Jie Tao, Dr. Edmond Kereku, Dr. Josef Weidendorfer, and Houssam Haitof, who worked together with me on projects, and Dr. Daniel Stodden, who started and finished the Ph.D. study at about the same time and discussed many issues towards successful completion of the Ph.D.

My research work documented in this thesis has been supported by IBM Center of Advanced Studies with a research grant for Autonomous Resource Management for Large Scale Applications. Thanks to Dr. Toni Bollinger, Dr. Hans-Dieter Wehle, and Dr. Nikolaus Breuer from IBM Development Laboratory in Boeblingen for all the valuable discussions and help in both the application of research funding from IBM and the whole course of this project.

Finally, thanks to my wife who accompanied me and helped me taking care of many issues so that I can concentrate on the work. Thanks to my son, Tony, who has brought happiness, sorrow, hope, and a lot more to me. You have helped me to understand the true meaning of life.

Tianchao Li
Munich, Germany
December 2007

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 History and Status

Grid computing is concerned with coordinated resource sharing in a dynamic, heterogeneous, and multi-institutional environment [46]. Grid resource management refers to "the operations used to control how capabilities provided by Grid resources and services are made available to other entities, whether users, applications or services" [33]. It is the key to flexible and efficient resource sharing in Grid systems.

The vision and technology of Grid computing has been ever evolving since its emergence, and so is the resource management techniques involved. Initial generation of Grid systems known as meta-computing (e.g. I-WAY [48], and the well-known SETI@home [12]) involved proprietary solutions for sharing computing resources (high performance or commodity). Such systems usually features centralized resource management with custom job scheduling. Later systems introduced middlewares (e.g. Legion [31], Unicore [39], and Globus Toolkit [47] up to version 2) to cope with scale and heterogeneity. The Grid was viewed as a viable distributed infrastructure on a global scale that support diverse applications requiring large scale computational power and large volumes of data. Such systems have common job submission interfaces (like that of GRAM [34] and Condor-G [50] etc.) that interact with local schedulers, and usually features diverse meta-schedulers (e.g. Globus Community Scheduler Framework [1], GridWay [86], GridBus [30], GRMS [63]) enabling hierarchical resource management or custom brokers enabling resource management in a simple distributed fashion. Local information is globally shared, and the management of different types of resources is usually differentiated. Generally speaking, such Grid systems and their resource management systems mainly focus on the issue of dynamicity and heterogeneity, and do not address the issue of crossing management domains well enough.

Current Grid systems are adopting a service-oriented approach, in which a set of Web services abstract underlying resources [46]. Such services include standard services for accessing primitive resources and services providing custom functionalities and utilize arbitrary underlying resources. Service-oriented Grid systems demand resource management infrastructures featuring uniform interfaces for different services belonging to different management domains. This lead to the idea of resource management based on Service Level Agreements (SLAs) [33], where the service providers and consumers negotiate utilization of the service (hence the underlying resources) with guaranteed quality of services

(QoS). Resource management activities with SLAs provide uniform interface for managing various types of resources. They can happen in a peer-to-peer fashion, which avoid the global exposition of local information. Thus, in addition to better support of dynamicity and heterogeneity, it also addresses crossing management domain issues.

Existing research on SLA-based resource managements mostly originate from the applications in traditional Web service systems [99, 80, 100, 112] that focus on throughput and response time. On the other hand, initial work on SLA-based Grid resource management is still quite limited. Some are restricted to the presentation of basic concepts and approach (e.g. [33]), and others mostly only provides a partial solution (e.g. [76, 60]) or restricted to certain standard Grid services (e.g. [79, 105, 23]). Therefore, further investigation is demanded towards the establishment of an infrastructure for SLA-based Grid resource management that can be generally applied to potentially arbitrary custom Grid services.

1.2 Methodology and Outline

The work presented in this thesis aims at the establishment of a general infrastructure that facilitates the management of Grid resources with service level agreements. It consists of a site-local infrastructure for managing SLAs and local resources, and a global infrastructure assisting the advertising, brokering, and establishment of agreements between multiple clients and Grid resources. We focus on the fundamental issues for the establishment of such an infrastructure, including the specification of SLAs with WS-Agreement and the design and implementation of a generic and extensible support infrastructure for WS-Agreement specification, the evaluation of various techniques for application performance prediction and the establishment of a generic infrastructure for non-intrusive monitoring and adaptive prediction of the execution time of services, discussion and solution of job scheduling issues in the local resource management infrastructure with a focus on the cases where services have exclusive access to the local resource. In addition, it also presents the design and implementation of a set of development tools and support environments for the development, management, and accessing of service-oriented Grid applications and the integration of SLA-based resource management infrastructure.

The next chapter introduces our infrastructure for SLA-based resource management in general. It features a global support framework, and local SLA and resource management infrastructure. This chapter outlines the major issues in both part and serves as a rule of thumb to more detailed discussions in Chapter 3 to Chapter 5 for local management, and Chapter 6 for the global infrastructure.

Chapter 3 introduces the specification and management of SLA with WS-Agreement. It provides a brief overview of the WS-Agreement specification, together with a short overview of our contributions toward the finalization of this specification. A generic and extensible support infrastructure for WS-Agreement is presented, which utilizes a provider mechanism to provide custom processing for domain-specific part of the agreement.

Chapter 4 discusses the application of prediction techniques for the evaluation of program performances. It presents a general introduction of application performance pre-

diction techniques, including analytical modeling, statistical simulation, and analysis of historical data. The applicability and efficiency of common data mining techniques for performance prediction has been evaluated. It presents a systematic approach for performance prediction using data mining techniques, which is supported with a generic run time monitoring and prediction framework for Grid services.

Chapter 5 discusses the local job scheduling issue in local resource management. With a formal parameterization of the problem, algorithms of deadline-constraint scheduling and discussion on more advanced scenarios are discussed. These include the impact of SLA acknowledgement and asynchronous parameter submission, user specified earliest start time, lazy termination and alternative SLA offers. A comparison of different algorithms and a scheduler implementation is presented.

Chapter 6 focuses on the global infrastructure for SLA-based resource management. The design and implementation of a minimal support infrastructure formed by index service and broker service has been presented. In addition, a gateway service that enables traditional stateless Web services be managed by the SLA-based resource management infrastructure has been presented. Advanced services supporting a global infrastructure that overcomes potential issues of scalability is also briefly discussed.

In Chapter 7, development tools and support environments that help develop a Grid system utilizing the proposed SLA-based resource management infrastructure are presented. It include a Grid service development environment that helps to develop Grid services and integrating them with the SLA-based resource management infrastructure, a base client environment that can be used as the basic platform for using services in a Grid managed by the SLA-based resource management infrastructure, and a management environment that helps Grid managers to manage the Grid servers and controlling the deployment of Grid services. Based on component-based development, the development and usage of the different environments are quite efficient and flexible.

Chapter 8 presents a demonstration of the SLA-based resource management based on a real-life application scenario of distributed data mining. The scenario will be detailed, including user activities and resource management activities behind the scene. The design and implementation of this demonstration, including the service bundle and client applications for data specialists and normal data analyzers are introduced.

The thesis concludes with a short summary and some future directions in Chapter 9.

Chapter 2

An Infrastructure for SLA-based Resource Management

2.1 Introduction

With Grid computing adopting the Service Oriented Architecture (SOA), resources are represented by services with standardized interfaces and the management of resources converges to the more general management of services (i.e. the Service Level Management). The key to Service Level Management is the management of Service Level Agreement (SLA), a formal contract between the provider and consumer of a service which stipulates and commits the provided service to a required level of service.

We concentrate in our research on designing and establishing a general infrastructure for SLA-based Grid resource management. This chapter will provide a brief introduction and discuss the challenges and applications. The rest of this chapter is organized as follows. Section 2.2 presents a global overview of the SLA-based resource management infrastructure. Section 2.3 discusses the challenges towards the establishment of the infrastructure. Section 2.4 discusses the applications scenarios that the infrastructure can be typically applied. This chapter ends with a summary in Section 2.5.

2.2 An Infrastructure for SLA-based Resource Management

Figure 2.1 presents a general overview of the SLA-based resource management infrastructure. The infrastructure includes facilities for local management at resource site that handles the management of SLAs, resources and performances, and the global management infrastructure composed by supporting services such as the broker service as indicated in the figure and other services.

From the client's point of view, a typical session of service usage includes the following activities: the client sends job parameters and SLA parameters (such as deadline, price etc.) to the broker, the broker contacts the service sites to get available SLA templates, the broker selects the appropriate template and sends it to the client for verification, and an acknowledgement is represented by the return of a signed SLA offer. The SLA offer is then forwarded by the broker to the corresponding service site to create the SLA. With the

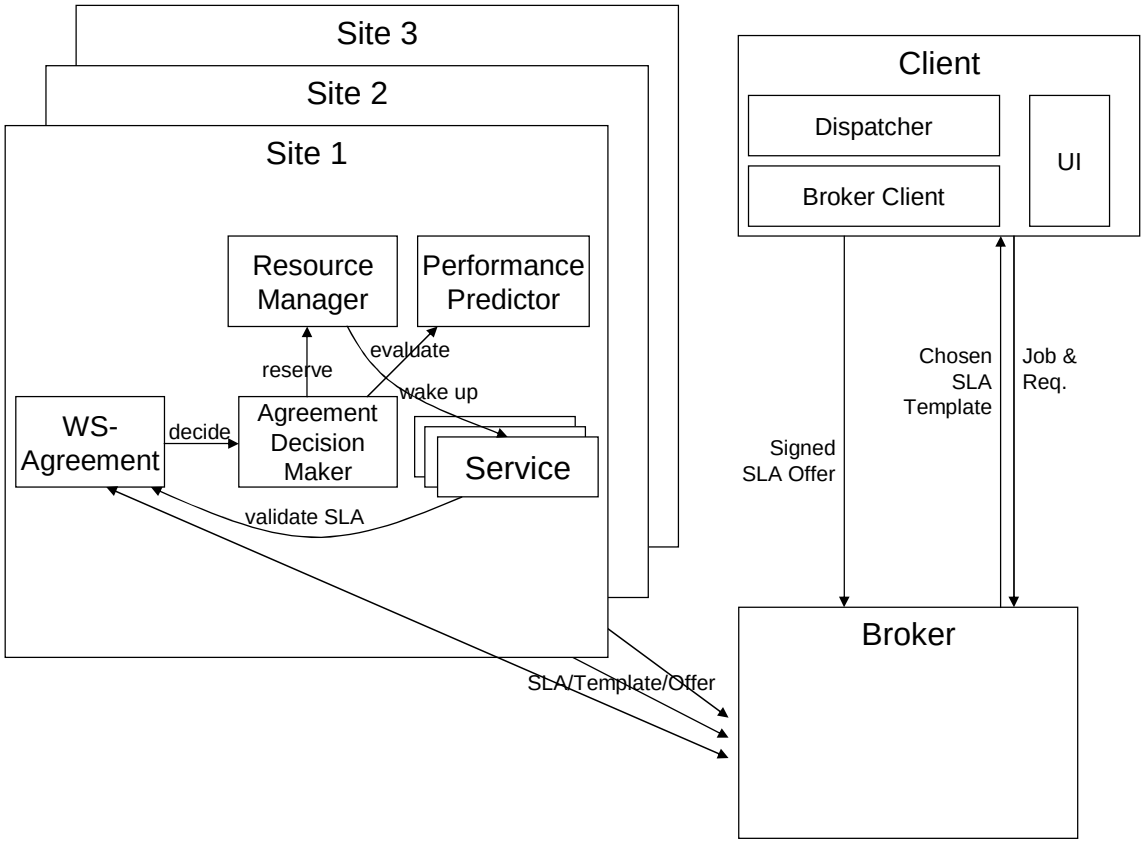

Figure 2.1 An infrastructure for SLA-based resource management.

negotiated SLA, the client invokes the service and gets an id (endpoint reference, or EPR) of the job. According to the negotiated SLA, the job is scheduled at appropriate time and as soon as the job finishes, a notification is sent to the client so that the result can be retrieved.

On the service site, a service has an associated management infrastructure that handles the management of SLAs and resources. The WS-Agreement services provide interfaces for requests for SLA templates, SLA creation and termination, notification of SLA status changes etc. The performance predictor estimates resource demands for specific job parameters, and the resource manager keeps track of resource capabilities and job schedules. Based on the information provided by both, the agreement decision maker decides on the acceptance or rejection of a specific SLA offer. The resource manager (together with the underlying scheduler) is also responsible for the scheduling of jobs with resource reservation and service invocation at scheduled time.

2.3 Challenges

Several challenges have to be addressed to the establishment of the SLA-based resource management infrastructure:

Specification and Management of SLAs The specification and management of SLAs is a central task for the resource management infrastructure. It is concerned with the formal specification of service quality requirements in the form of SLAs and the management of the negotiated SLAs. In a service-oriented Grid architecture, SLAs should be specified with languages that are general and flexible enough for different services. Support infrastructure that faciliate the negotiation and management of SLA-based resource management need to be established to support arbitrary number and different types of services that are deployed.

Prediction of Resource Demand The management of agreements requires the knowledge of resource demand or application performance. In many situations, they are not explicitly specified and can only be estimated from application specific parameters. Feasible methods range from analytical modeling and statistical simulation to predictions based on historical data. In addition to the evaluation of such prediction techniques, a generic framework must be established that can be adapted to different services and support pluggable prediction techniques.

Local Resource Management and Job Scheduling As a major part of the local facilities for SLA management, the local resource manager is responsible for the management of local resources that are made available through the Grid. In addition to the examination of resource availability for a certain request, a major task of local resource management is the scheduling of service invocations or jobs. the scheduling of service invocations in a SLA-based resource management environment has specific requirements that can not be easily fulfilled by such scheduler - on-line scheduling targeting maximization of the number of jobs that can be served, jobs are normally constrained with deadline and possibly also the earliest start time, integral admission control, and inaccuracies in the predicted execution time of jobs - and thus deserves to be investigated specifically.

The Global Infrastructure In addition to the local management infrastructure that provides fundamental support for SLA-based resource management, a global infrastructure should be established that assists the establishment of agreements between multiple clients and Grid resources. Such infrastructure can, for example, help client to seek for available servers and dynamically select the appropriate server that can meet its requirement.

Programming and Run Time Environments Applications managed by the SLA-based resource management infrastructure are formed with Grid services providing standard or custom functionalities. Supporting tools and environments are demanded for the development, execution, and management of such applications and the underlying infrastructure. They are important enablers that make the SLA-based resource management infrastructure a practical approach for Grid application developers and users.

In subsequent chapters, each of the above challenges will be addressed with detailed discussions.

2.4 Application Scenarios

The SLA-based resource management approach is expected to applicable to different applications of Grid computing. Examples of typical application scenarios include:

Computational Grid for High Performance Applications Computational Grids for high performance applications represent a typical application scenario in scientific realm, where high performance computers from computing centers are made available through Grid infrastructure. Users code, compile and submit programs to be executed. In addition to the program parameters, the submissions are accompanied with job specifications of requirements or preferences for the execution platform, number of CPUs, deadline, and price etc.

Service Grid for Distributed Data Mining A typical application scenario of Grids in businesses is to utilize the Grid to facilitate distributed data mining on enterprise data. In this scenario, enterprise business data are collected in data warehouses. Analysis requirements like data mining that are transient and demands additional resource than the daily processing of data, which is to be fulfilled by the provisioning feature of Grid. Multiple services are deployed on the servers and are managed by the Grid infrastructure. Users specify the data and the data mining method together with parameters for the specific data mining method and requirements or preferences for deadline etc.

Other scenarios also exist, but the above two scenarios represent typical application scenarios in scientific and commercial realms and covers Grid applications with both standard Grid services and custom ones. Therefore, they are to be used throughout our discussions to help explain the design decisions of the SLA-based resource management infrastructure.

2.5 Summary

This chapter presents an overview of a SLA-based resource management infrastructure for applying service level agreements in Grids for autonomous resource management. This infrastructure incorporates a global infrastructure assisting SLA negotiation and a local management infrastructure for SLA enforcement which manages local resources and handles agreement negotiation and service provisioning.

The following chapters will discuss each of the major issues towards the establishment of such an infrastructure, include the specification and management of SLAs in Chapter 3, prediction of resource demand in Chapter 4, local resource management and job scheduling in Chapter 5, the establishment of a global infrastructure in Chapter 6, and the establishment of programming and run time environments in Chapter 7. In addition, Chapter 8 presents a demonstration dedicated to a detailed scenario for distributed data mining.

Chapter 3

Specification and Management of Service Level Agreement

3.1 Introduction

A Service Level Agreement (SLA) is a contract between a service provider and a service consumer about the Quality of Service (QoS) that will be guaranteed by the service provider [119]. The negotiation and management of SLAs is a central task for the resource management infrastructure.

SLAs usually take the form of SLA documents, the syntax and semantics of which are specified by SLA languages. Several different SLA languages have been specified (see Section 3.5.1 for a brief summary). The management of SLAs requires support infrastructures which in some cases are defined as part of the specification that specifies the SLA language and in other cases, like that of WS-Agreement, only the interfaces are specified and details of the infrastructure is not.

As is mentioned before, service-oriented Grids are constituted by many different standard services as well as arbitrary custom services. Each service has domain specific terms for describing the service parameters, results and QoS requirements. The generality and flexibility is a major criterion for the choice of SLA languages and the design of support infrastructures. This is also the major target of the WS-Agreement specification, which is designed to ensure that any domain specific or other standard condition expression language can be used.

WS-Agreement [13], as a standard developed by the Grid Resource Allocation Agreement Protocol working group (GRAAP-WG) [1] of Open Grid Forum (OGF)[2], is commonly recognized in both Grid computing and broader SOA. It is becoming one of the *de facto* standard languages of service level agreement.

As an active member of OGF GRAAP working group, we are deeply involved in the finalization process of the WS-Agreement standard. An implementation of the specification, on top of Globus Toolkit 4 (GT4), has also been developed as part of the Automated Resource Management for Large-Scale Applications project.

This chapter focuses on our research and development activities in the specification, de-

[1] http://forge.ogf.org/projects/graap-wg
[2] http://www.ogf.org

sign and implementation of WS-Agreement and a generic extensible support infrastructure. It is organized as follows: Section 3.2 introduces the WS-Agreement specification and our involvement and contributions. Section 3.3 describes the design of a generic support infrastructure, including the service interfaces that are responsible for interactions, management entities that are responsible for decision making and agreement monitoring, and backend provider mechanism that enables domain-specific handling of WS-Agreements. Section 3.4 explains various issues in the implementation, including the choice of GT4 as implementation platform, adaptation of WS-Agreement schemas for GT4, the type mapping and persistence issues for domain-specific terms, registration and management of providers, monitoring of agreement objectives and security issues. Related works are reviewed in Section 3.5. Performance evaluation of the implementation is presented in section 3.6. Section 3.7 concludes this chapter with a brief summary. In addition, examples of WS-Agreement agreement template, agreement offer and agreement within context of the data mining scenario described in Section 2.4 are provided in Appendix B.

3.2 WS-Agreement Specification

3.2.1 Overview

The Web Services Agreement (WS-Agreement) specification [13] is a Web service protocol for establishing agreements between a service provider and consumer, using an extensible XML language for specifying the nature of the agreement, and agreement templates to facilitate discovery of compatible agreement parties. The specification consists of three parts which may be used in a composable manner: a schema for specifying an agreement, a schema for specifying an agreement template, and a set of port types and operations for managing agreement life-cycle, including creation, expiration, and monitoring of agreement states.

The specification addresses several major issues in generality and extensibility. Such issues include the usage of arbitrary service terms, creation of agreements for existing and new services, usage of any condition specification language, symmetry of protocol, independence of negotiation model, and independent usage of different parts of the specification.

The main concepts of the WS-Agreement specification can be outlined with an example excepted from [79], which is shown in Figure 3.1. In the chosen example, the agreement responder is a service provider, the agreement initiator the service consumer. An agreement responder exposes an AgreementFactory interface, which offers an operation to retrieve a set of agreement templates proposed by the agreement provider and an operation to create an agreement. Agreement templates are agreements with fields to be filled in. Templates help an agreement initiator to create agreement offers that the agreement provider can understand and accept. The create agreement operation returns accept or reject, if a synchronous reply is expected. Otherwise, in case of a longer decision-making process, the service responder can convey the decision to an AgreementResponse interface that the agreement initiator exposes. If the create agreement operation succeeds, an agree-

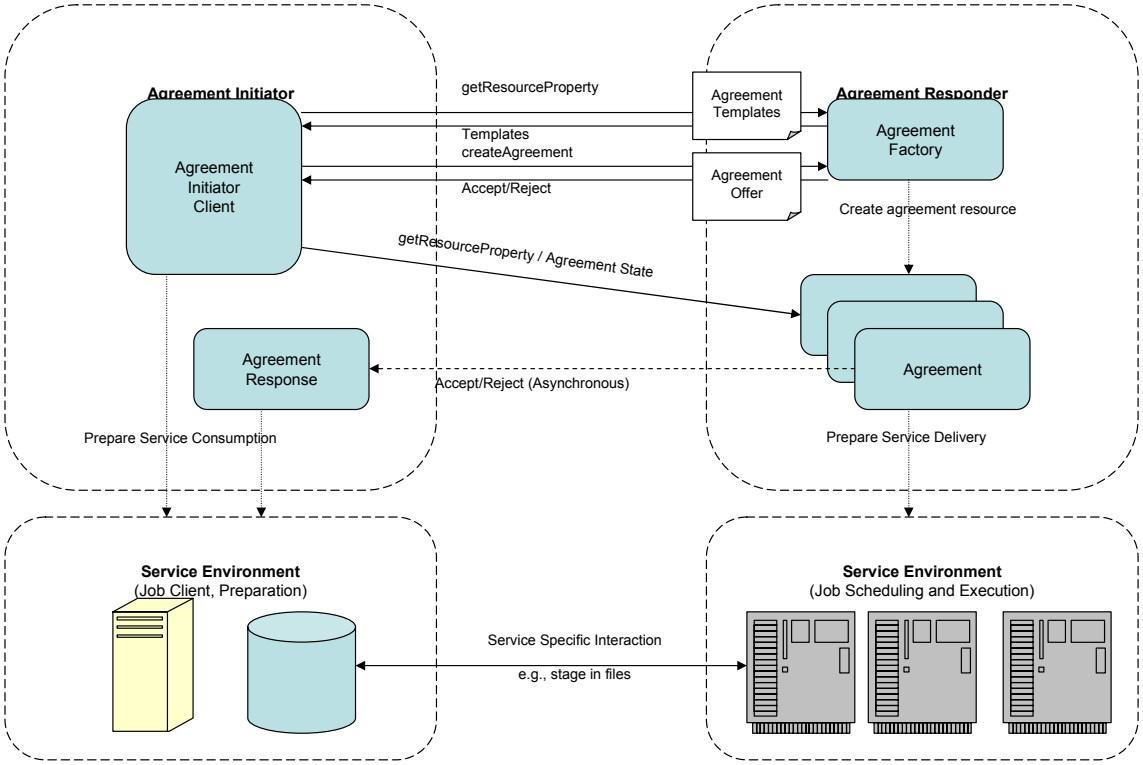

Figure 3.1 Concepts and interfaces of WS-Agreement [79].

ment instance is created. The agreement instance exposes the terms of the agreement as properties that can be queried. In addition, runtime states for the agreement as a whole and its individual terms can be inspected by the initiator. All interfaces exposed by the parties, AgreementFactory, Agreement and AgreementResponse are resources according to the Web Services Resource Framework (WSRF) [2]. Upon acceptance of an agreement, both service provider and service consumer have to prepare for the service, which typically depends on the kind of service subject to the agreement. For example, a service provider schedules a job that is defined in the agreement. A service consumer will make the input files available as defined in the agreement. Further service specific interaction may take place between the parties governed by the agreement.

The remaining part of this section will provide a brief overview of WS-Agreement specification. For further details of the specification, please refer to the original document [13].

3.2.2 Agreement Schema

As is shown in Figure 3.2, an agreement (or agreement offer) is conceptually composed of several distinct parts, including an (optional) name, meta-data for the entire agreement as context, and terms that describe the agreement itself.

Agreement context includes the identity of the initiator and responder, service provider,

Figure 3.2 Structure of a WS-Agreement [13].

and expiration time of the agreement. Optionally, it also contains the id and/or name of the template from which the agreement is created.

Agreement terms constitute the major body of an agreement offer and the consequent agreement. There are two types of agreement terms: service terms and guarantee terms. The service terms provide information needed to instantiate or otherwise identify a service to which this agreement pertains and to which guarantee terms can apply. These are further refined as service description, service reference and service property terms. The guarantee terms specify the service levels that the parties are agreeing to. Management systems may use the guarantee terms to monitor the service and enforce the agreement.

Special compositor elements can be used as logical AND/OR/XOR operators to combine terms. This enables the specification of alternative branches with potentially complex nesting within the terms of agreement.

3.2.3 Agreement Template Schema

The structure of an agreement template is the same as that of an agreement, but an agreement template may also contain a creation constraint section (see Figure 3.3). This section specify constraints on possible values of terms for creating an agreement. The constraints make it possible to specify the valid ranges or distinct values that the terms may take. The constraints refer back to individual terms they apply to using XPath [4].

3.2.4 PortType Definition

Several different port types have been defined by the WS-Agreement specification. They define the interfaces to the support infrastructure in both the server side and client side. The port types include AgreementFactory, PendingAgreementFactory, Agreement, AgreementAcceptance and AgreementState.

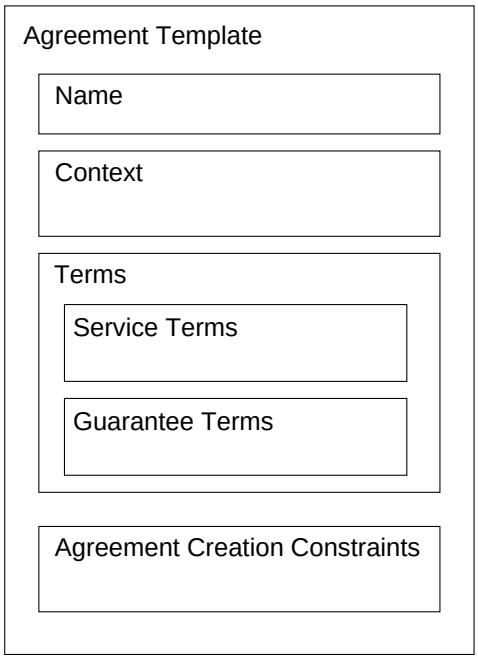

Figure 3.3 Structure of a WS-Agreement template [13].

Based on top of WSRF, every port type exposes a wsrp:GetResourceProperty operation as defined in WS-ResourceProperties. This operation enables the port types to expose read-only resource properties. Other operations defined by the WS-ResourceProperties like wsrf-rp:GetMultipleResourceProperties and wsrf-rp:QueryResourceProperties may also be composed into domain-specific agreement and agreement factory types. The wsrl:Destroy operation as defined in the WS-ResourceLifetime may be used by the initiator to explicitly destroy no longer used resources.

AgreementFactory port type provides template resource property that represents a sequence of 0 or more templates for agreement offers. It also defines wsag:CreateAgreement operation for generating an agreement from agreement offer, which returns either an EPR of the created agreement or a fault response indicating that the offer was rejected and may also indicate domain-specific reasons.

PendingAgreementFactory port type also has the template resource property, like the AgreementFactory port type. It defines a wsag:CreatePendingAgreement operation for generating an agreement when the decision process is deferred. In combination with port type wsag:AgreementAcceptance, which allows a deferred decision to be communicated as to whether the offer is accepted or rejected, it allows the creating of agreement in an asynchronous way.

Agreement port type has resource properties that represent an agreement, which is created either by AgreementFactory or PendingAgreementFactory port type. For separation of concern, resource properties that represent the status of an agreement are defined in a separate port type wsag:AgreementState. However, it is intended to be composed with the

Agreement port type. wsag:Agreement port type also provides a wsag:Terminate operation, which terminates an agreement, if possible.

3.2.5 Involvement and Contributions

As an active member of OGF GRAAP working group, we are deeply involved in the development process of WS-Agreement specification. By participating in the discussions, improving schema and port type definitions, as well as renewing the specification document, we have played an active role in finalizing the specification.

Our contributions help to improve the specification in different aspects, including better conformance to WS-ResourceFramework standard, improved type definition of agreement states and service term states, more consistent naming conventions and so on. Most of these contributions are accepted by the working group and become part of the final version of the specification. In addition, we have also defined extensions to the specification that we have found to be necessary in practical application of the specification. For more details of our contributions, please refer to [74].

3.3 WS-Agreement Support Infrastructure

3.3.1 Generic Design

Unlike many other SLA languages that are accompanied with a detailed design of the support infrastructure, the WS-Agreement specification only describes the interface. The nature of WS-Agreement specification requires the design of the support infrastructure to be generic as well.

Figure 3.4 shows a schematic diagram of the WS-Agreement support infrastructure. It has a layered design, with the following three layers in its architecture: interface layer that provides service interfaces as defined by the WS-Agreement specification, management layer containing components for managing and monitoring the agreement, and extension layer with managed sets of backend providers for custom processing of domain-specific terms.

3.3.2 WS-Agreement Services

The services follow the port type definition of the WS-Agreement specification and provide external interfaces for the whole support infrastructure.

WS-Agreement specification defines the interfaces that an AgreementAcceptance service should implement. Practically, each client might maintain multiple agreements at the same time and thus need to manage multiple “instances” of AgreementAcceptance services. In the term of WS-ResourceFramework, we need a Web Service - Resource pair, i.e. we need to introduce state to AgreementAcceptance service and provide a factory that manages the creation of the service “instance”s. This introduces the AgreementAcceptanceFacory service that dynamically creates and manages AgreementAcceptance resources, which enable the client to trace the status of multiple agreements dynamically.

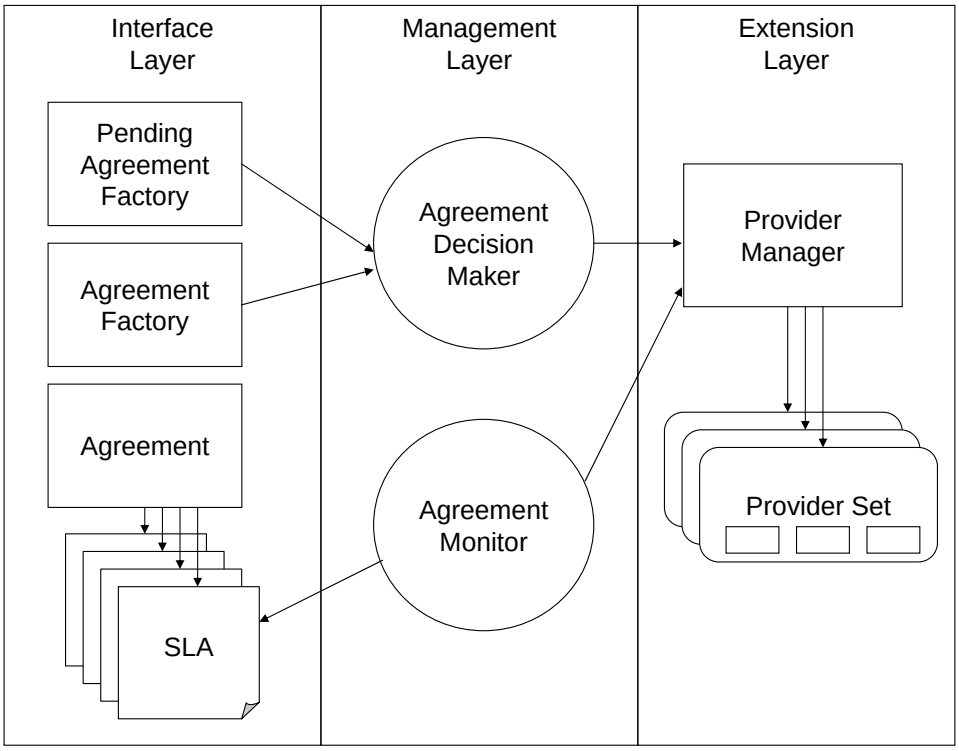

Figure 3.4 WS-Agreement support infrastructure.

They collaborately serve as a necessary extension to the WS-Agreement specification so that the AgreementAcceptance service can be practically used to support the acceptance notification of multiple agreements.

A schematic presentation of the implemented services and resources on both server side and client side are illustrated in Figure 3.5. The port types (PendingAgreementFactory, AgreementFactory, Agreement, and AgreementAcceptance) and resources (AgreementFactoryProperties and AgreementProperties) in bold are those defined by the WS-Agreement specification.

3.3.3 Agreement Management and Monitoring

While the SLA services implement interfaces for the interaction between agreement initiators and responders, the actual management of the agreements is conducted by backend management components. The decision of agreement creation is delegated to agreement decision maker, and the responsibility for the monitoring of agreement status is taken over by the agreement monitor.

The agreement decision maker cooperates with the performance predictor and local resource manager to estimate the resource demand and check the actual availability of resource as requested. If possible, the agreement decision maker also checks or determines the price of the service according to predefined strategies.

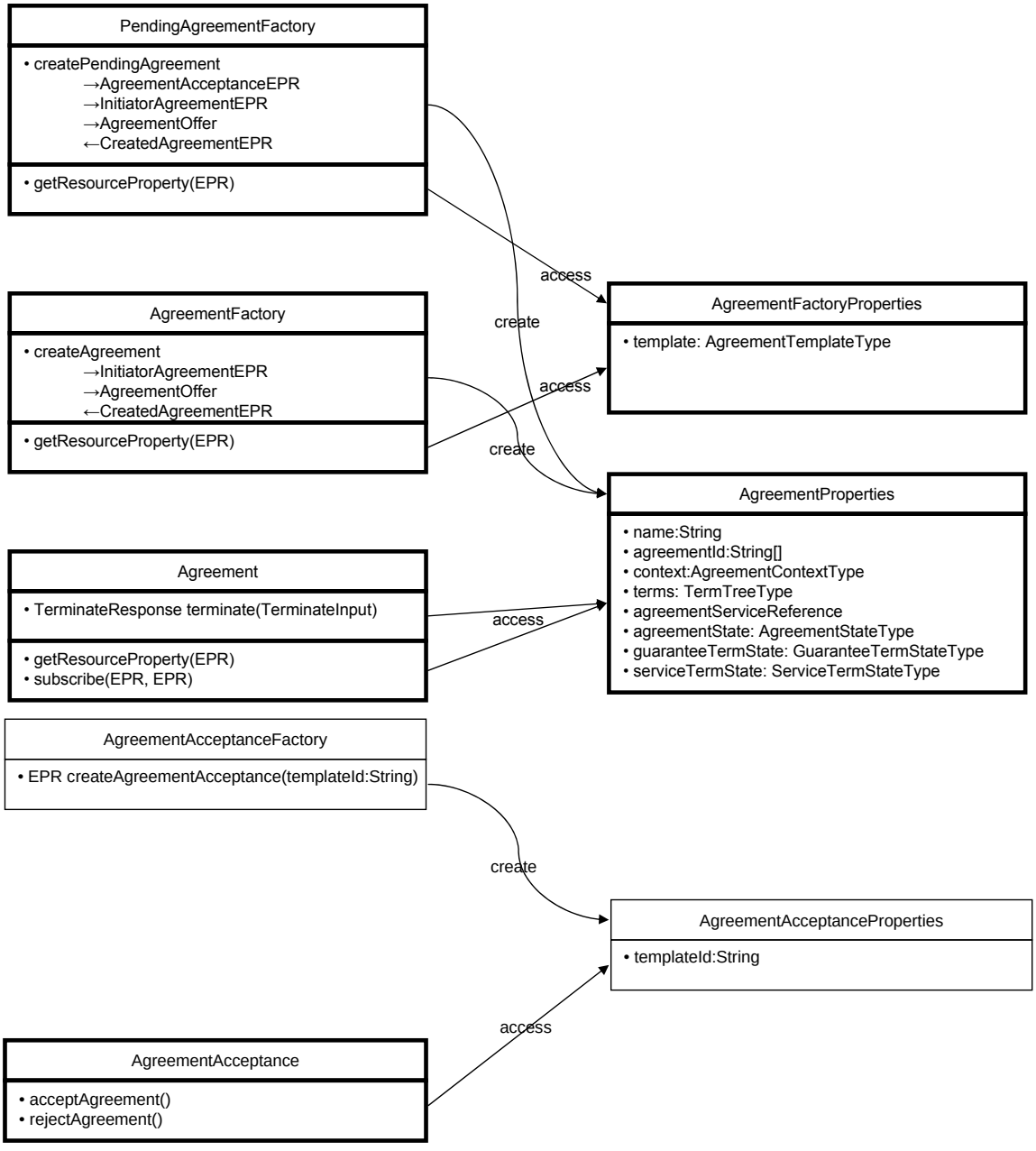

Figure 3.5 Agreement management services on server-side (upper) and client-side (lower).

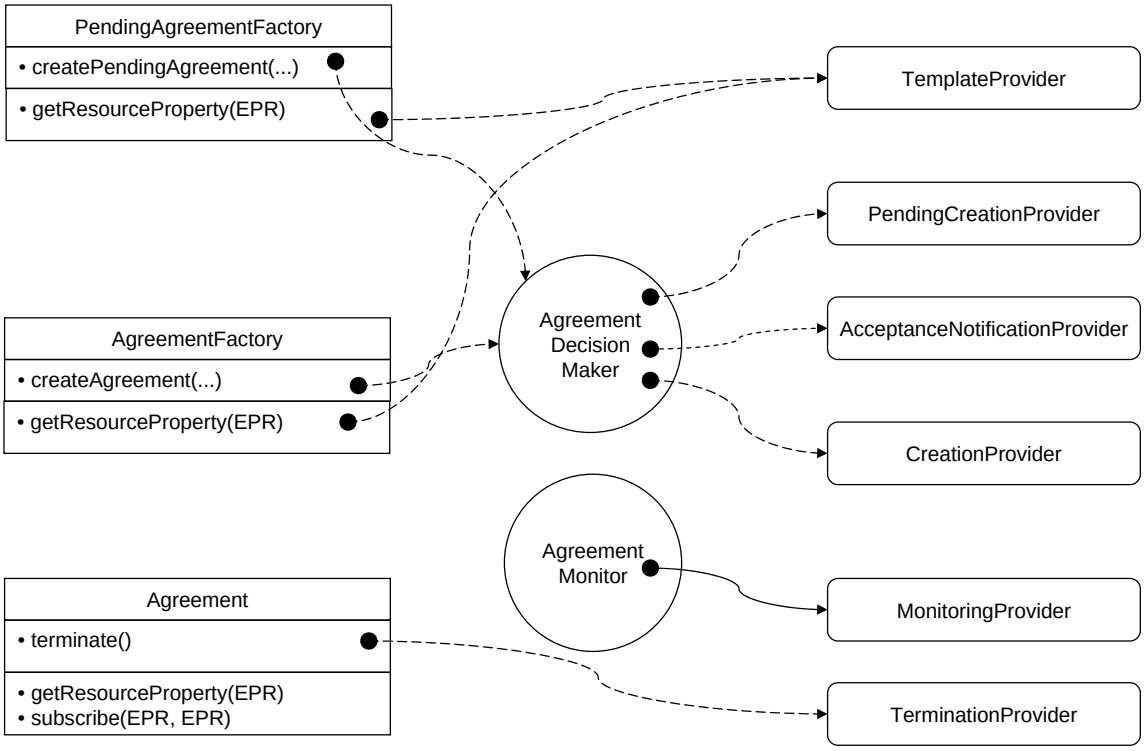

Figure 3.6 Providers handle domain specific or customized processing of agreement.

Each time a new agreement is created, a new agreement monitor is started that monitors the objectives specified in the agreement. The agreement monitor utilizes resource monitoring information to check the fulfillment of agreement objectives and updates the status of agreement as appropriate. The change of status can be passively accessed by or proactively notified to the corresponding party of the agreement. The agreement monitor checks the status periodically, with an interval specified in the agreement.

3.3.4 Domain Specific Handling with Backend Providers

Provider and Provider Set

The WS-Agreement specification is defined to be generic, and allows the usage of domain specific terms by using xs:any elements. Although a dedicated WS-Agreement implementation can be implemented that works only for specific domain, it is desirable to maintain the generality of the WS-Agreement services and allowing configurable extensions. For Grid systems, it should also provide concurrent support for multiple services that are dynamically deployed. We have designed a provider-handler infrastructure to achieve such goal of extensibility and configurability.

While the service front-ends provide generic interfaces for the interaction between agreement initiator and responder, providers are invoked at critical life-cycle points to perform domain specific or custom processing (ref. Figure 3.6). All providers must implement

appropriate interfaces with multiple classes or a single class. Such interfaces include:

1. IAgreementTemplateProvider: interface for agreement template providers. Each factory or pending factory service can have multiple providers associated.

2. IAgreementCreationProvider: interface for agreement creation providers. It defines methods to be invoked before and after agreement creation. With different return values of the former method, it also allows custom control of the agreement creation.

3. IPendingAgreementCreationProvider: interface for pending agreement creation providers. It defines methods to be invoked before and after pending agreement creation.

4. IAgreementTerminateProvider: interface for agreement termination providers. It defines methods to be invoked before and after agreement termination. With different return values of the former method, it also controls whether the agreement termination is allowed.

5. IAgreementAcceptanceNotificationProvider: interface for agreement acceptance notification providers. It defines methods to be invoked before and after agreement acceptance notification is delivered. This allows customized actions to be taken, for example, resend the notification or simply ignore it.

6. IAgreementMonitoringProvider: interface for custom agreement monitoring. It defines methods to be invoked by the agreement monitor in each monitoring cycle.

Each custom service implementation provides a set of above mentioned providers on a per-template basis. Different providers for the same template can be separately implemented by multiple classes each implements one provider or one single class that implements all providers.

Provider Management

Figure 3.7 schematically illustrates major activities of provider management, including the registration, instantiation and retrieval of agreement providers.

Each custom service comes provides a set of providers on a per-template basis. Programmers of the custom service are required to proactively register their set of providers, using provided APIs for provider registration.

The providers are managed by ProviderManager, which maintains a registry of available providers. The registry is a mapping between the agreement template ID and the corresponding set of providers. The providers are instantiated at the first time when they are accessed and are cached for future retrieval. The advantage of this approach is that the same agreement management infrastructure can provide differentiated processing for each template, or serve different types of services at the same time.

In case providers can not be found in the registry for a specific template ID, dummy providers will be used by default. The dummy providers perform nothing more than logging, which can be used for debugging and performance studies.

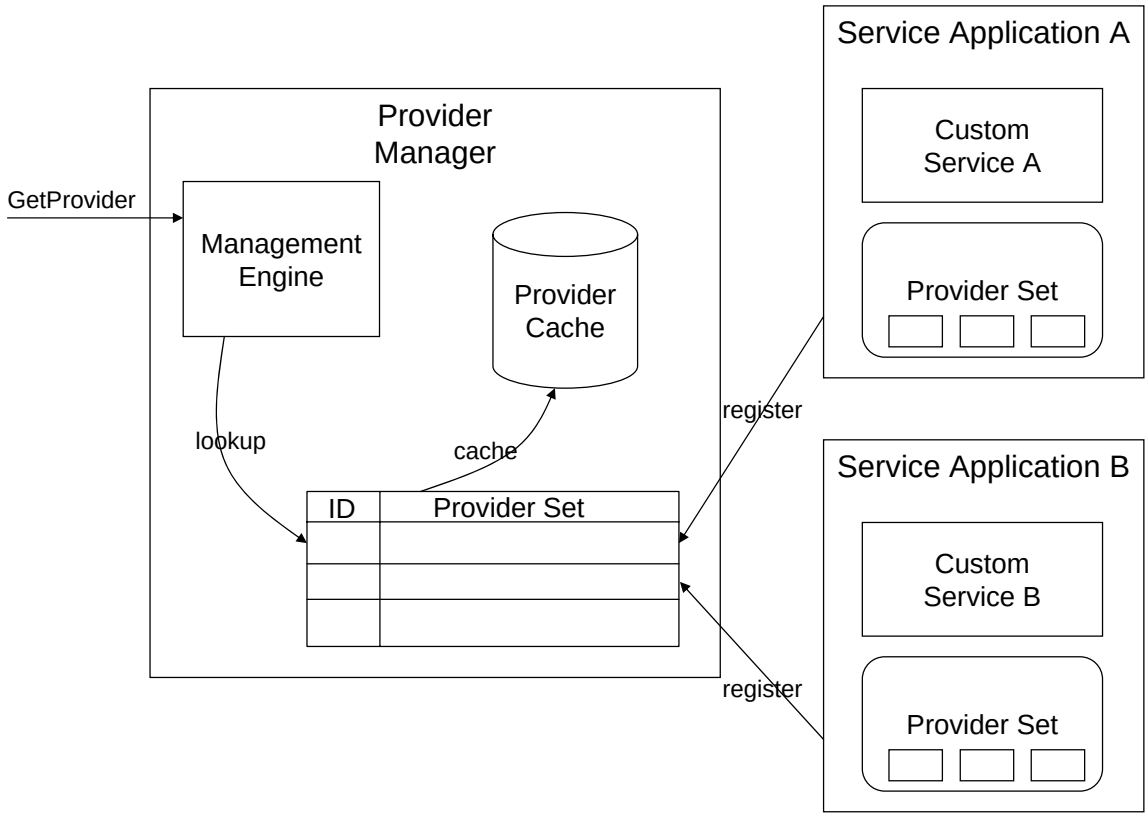

Figure 3.7 Provider registration and management.

3.4 Implementation Issues

3.4.1 GT4 Platform

We have chosen Globus Toolkit 4 (GT4) [3] as the platform for our WS-Agreement implementation. As the major component and basic platform of Globus Toolkit, GT4 WS Core provides a Web service platform featuring support for WS-ResourceFramework (WSRF) and WS-Notification family of standards, as well as security technology for Web services, and the Servicegroup implementation.

While WSRF support is the prerequisite of WS-Agreement, the Grid Security Infrastructure that provides extensive support for authentication and authorization is needed for both the identification of participating parties in the agreements and the security of agreement services themselves.

As an open source software toolkit for building Grids with services, GT4 is the *de facto* standard Grid platform. Having our implementation based on GT4 provides better opportunity for our implementation to be integrated and utilized in various Grid systems.

[3] http://www.globus.org/toolkit

3.4.2 Interface Schema Adaptation

Due to specialties and limitations of GT4, the schema definitions of WS-Agreement specification need specific adaptations. This includes the replacement of namespaces and fault types that are not supported or differently defined by GT4, specification of imported WSDLs and XSDs, simplifying the WSDLs by using compact schema, and modifications to the definition of xs:simpleRestrictionModel and xs:typeDefParticle elements to overcome the limitation of GT4 on xs:choice elements. More details are provided in Appendix A.

3.4.3 Domain Specific Terms

Wildcards in WS-Agreement

WS-Agreement is designed to be generic to cope with different usage scenarios and application domains. As a result, the specification use XML wildcards like xs:any and xs:anyAttribute extensively in the schema definition, to allow the occurrence of elements and attributes from specified namespaces. The extensibility of the content model is thus enabled, while maintaining a degree of control over the occurrence of elements and attributes.

Definition and Processing

Domain specific terms are defined within separate XML schemas as part of custom services. Those XML schemas need to be imported into WS-Agreement WSDLs to be processed by the Web service platform.

The XML schemas contain only the syntax of custom elements, and do not provide semantic information. The contributed handlers, as part of the custom service, are responsible for understanding and custom processing of the elements that are specific to a concrete domain and usage scenario.

Type Mapping

GT4 relies on the type mapping mechanism of Axis [4] for custom mapping between XML elements and Java objects. As an extensible and configurable type mapping mechanism, it allows the user to define custom type mapping.

Although type mapping configuration of common WS-Agreement elements can be directly inserted into client-config.wsdd, this is not convenient for domain specific elements and attributes of custom Grid services. This can be solved by defining custom type mapping definitions in client-deploy.wsdd as part of the custom service, which will automatically update the client-config.wsdd when the service is deployed.

Persistence of Resource Properties

The properties of the agreement WS-Resource need to be persisted to survive container restarts. Such persistence in GT4 is done by serializing each resource property object.

[4] http://ws.apache.org/axis/

GT4 guarantees that the message elements are serializable, except for elements of type `xs:anyType`, e.g.:

```
<xs:element name="QualifyingCondition" type="xs:anyType" />
```

Such elements that are not serializable demand a custom serializer/deserializer to assist the persistence. The custom serializer and deserializers are provided together with the custom services to handle domain specific elements.

3.4.4 Provider Management

Provider Registration

The provider registry is implemented as a Hash table. The key of a table item is the agreement template ID, and the value is an instance of ProviderSet, a class that contains each provider for that template. The ProviderSet also contains links to existing provider instances, which is utilized by provider manager to manage cached instances.

Registering Providers Proactively

The registrations of providers need to be done by the custom services proactively, which is contradictory to the passive invocation pattern of Web services. Normally, a Web service is not instantiated until it is invoked for the first time. By modifying the deployment descriptor of the Web service, it is possible to tell the Web service engine to load and instantiate the Web service class on startup.

The code that actually performs provider registration can be provided as part of static code in the service class, which will be automatically executed on class loading. A better alternative is to use the life cycle support of Web services by having the Web service class implementing javax.xml.rpc.server.ServiceLifecycle interface. This allows not only the registration of providers in the init() method, but also de-registration in the destroy() method. We chose this approach in our implementation.

Having Web service classes which normally have a large memory footprint loaded on startup is not memory efficient. Besides, we would like to separate the implementation of custom services and the code for provider registration. Therefore, in place of the custom service, we have a dedicated service to provider registration which is configured to be loaded on startup. Except for the code that handles the registration and clear up, the dummy service does not implement other methods and thus keeps a very small memory footprint.

3.4.5 Agreement Monitoring

The agreement monitor checks agreement objectives periodically. It is implemented with the help of the Timer framework supported by GT4 WS Core. The key components of the Timer framework are TimerManager and TimerListener. TimerManager is responsible for the periodical invocation of TimerListener. The TimerListener is where the timer work is implemented. The following code snippet is an example of retrieving TimerManager from

the Web service context and the registration of a new timer that is started every 100 seconds from now:

```
InitialContext ctx = new InitialContext();
TimerManager timerManager = (TimerManager)
    ctx.lookup("java:comp/env/timer/ContainerTimer");
timerManager.schedule(new SLAMonitorTimerListener(),
    1000 * 100);
```

In the actual implementation, the interval for timer expiry is configured as is specified in the agreement. Upon the creation of each new agreement, a new timer is added that handles the periodical invocation of agreement monitor.

3.4.6 Security Issues

The WS-Agreement specification does not regulate the exact presentation of AgreementInitiator and AgreementResponder in an agreement. The approach we adopted is to use credentials of the participating parties (service provider and service consumer) to specify the initiator and responder of agreement. Besides providing identification of the negotiation parties, authentication is also necessary for authorization. GT4 WS Core provides extensive support for Web service security, which greatly simplifies our implementation for authentication and authorization.

3.5 Related Work

3.5.1 SLA Languages

Prior to WS-Agreement, several SLA languages exist. Well-known ones include the SLAng [106] [70] developed by University College London, Web Service Level Agreement (WSLA) [77] [78] developed by IBM T.J. Watson Research Center, Web Service Management Language (WSML) [99] [100] developed by Hewlett-Packard Laboratories, and Web Service Offering Language [114] developed by Ottawa-Carlton Institute of Electrical and Computer Engineering.

Overviews and comparisons of most languages can be found in [21] [91] [85]. Most of the SLA languages are formal languages in XML format. WSLA and SLAng have metrics as part of the language, which restricts their generality. Like WS-Agreement, WSLA supports the concept of agreement templates, which is not supported by SLAng and WSOL. Both WSLA and WSOL have practical applications in the industry, while SLAng is mainly used in academia. None of these languages are widely accepted and their usages are limited to specific field.

3.5.2 SLA Infrastructures

There have been a number of attempts at defining SLA support infrastructure for both Web and Grid services. Well-known ones include Web Services Offering Infrastructure

(WSOI) [112] [113] for WSOL, Web Services Management Network (WSMN) [100][80] for WSML, TrustCoM [6][120] for extended WSLA. An overview and comparison of those systems can be found in [91].

Except for the differences in the concrete design and certain excessive functions, those support infrastructures are essentially quite similar, with interface components providing operations that handle SLA language protocol and provide other externally-accessible operations, different management components that handles decision making, and SLA monitoring/evaluation, together with supporting data structures or stores. For monitoring purposes, many of those infrastructures instrument Web services by inserting handler or proxy into the SOAP processing route. Despite the fact that most of them are developed together with a specific SLA language, the same designs are applicable to other languages as well.

With a focus on coordinated management of SLAs for workflows, WSMN also features components for interaction with workflow engines and exchanging measurement information among WSMN members. WSMN also provides a backend graphical user interface for system administrators to examine SLA violation logs on specific machine. TrustCoM supports the separation of SLA monitoring/evaluation and notification functions from the local support infrastructure into third party services. WSOI design includes an extension mechanism to support other SLA languages with language specific handlers, however the actual implementation stays with WSOL only.

3.5.3 WS-Agreement Implementations

Cremona (Creation, Monitoring and Management of WS-Agreement) [76] is a WS-Agreement implementation by IBM T. J. Watson Research Center. As the first WS-Agreement implementation, and the only one which was available before the start of our work, it is based on an early draft of WS-Agreement which is very different from the final 1.0 version. Since the early draft does not rely on WS-ResourceFramework, it was able to be implemented on Axis. PendingAgreement and AgreementAcceptance port types are not supported. Because of intellectual property restrictions by IBM, the source code of Cremona is not available, although a binary distribution is included in IBM Emerging Technologies Toolkit (ETTK) [5] with an IBM AlphaWorks 90 Day Trial license. Cremona is designed to separate domain-independent from system-specific and domain-specific components. To customize it for specific domain, implementations of DecisionMaker, AgreementImplementer and StatusMonitor that follow specific interface should be provided and configured through configuration script.

The GridARM (Grid Askalon Resource Management) System [105] of Askalon [6] [40] contains a WS-Agreement implementation. The source code is accessible with a custom restrictive Askalon Software License[7]. As part of a resource management system, it is dedicated for advance reservation of Grid resources and both the design and implementation are tightly coupled with proprietary terms for resource reservation. It is based on a non-final

[5] http://www.alphaworks.ibm.com/ettk

[6] http://www.dps.uibk.ac.at/projects/askalon/

[7] http://www.dps.uibk.ac.at/projects/gridarm/software/license.html

	Platform	Spec Ver.	Complete	Source	License	generality
Cremona	Axis	early draft	no	close	custom	extend & config
ASKALON	GT4	pre-1.0	no	open	custom	no
AssessGrid	GT4	1.0	no	open	Apache 2.0	extend & static register
VIOLA	Axis2	1.0	no	close	custom	extend & config
WSAG4GT4	GT4	1.0	yes	-	-	dynamic

Table 3.1 Comparison of existing implementations of WS-Agreement.

version of the WS-Agreement specification, and does not implement PendingAgreement and Acceptance port types. It is implemented on top of GT4.

The Negotiation Manager [8] [20] is a component of the AssessGrid [9] [23] software stack that implements the WS-Agreement specification on GT4. It is an open source implementation, licensed under Apache 2.0 license. It complies with the WS-Agreement specification version 1.0, but currently PendingAgreement and AgreementAcceptance are not supported. The implementation is fixed to JSDL, JSDL-POSIX [15] and some domain specific pieces. Since it does not include an extension mechanism, customization has to be done by manual modifications and recompilation of the source code. The implementation separates generic and domain dependent code to ease the customization work.

The WS-Agreement Framework [10] [79] is implemented by the Fraunhofer-Institute for Algorithms and Scientific Computing (SCAI) as part of the VIOLA (Vertically Integrated Optical Testbed for Large Applications in DFN) project [11]. It is fully compliant with the WS-Agreement specification version 1.0, but currently PendingAgreement and AgreementAcceptance are not supported. It is implemented in Java and takes advantage of the innovative Axis2 platform. Besides separating generic and domain dependent code, it also provides a simple customization mechanism based on class inheritance and configuration script similar to Cremona.

Our implementation (denoted as WSAG4GT4 in Table 3.1) is by far the only implementation that fully implements the latest WS-Agreement specification. Compared to other existing implementations, it exceeds in its generality and extensibility. Domain dependent processings are dynamically contributed by the individual services as handlers. The handler manager manages multiple contributions so that they can coexist, thus multiple services of different domains can be supported concurrently. This is exceptionally important for Grid systems.

[8] https://cit-server.cit.tu-berlin.de/trac/negmgr/wiki

[9] http://www.assessgrid.eu/

[10] http://packcs-e0.scai.fraunhofer.de/mss-project/wsag4j/index.html

[11] http://www.viola-testbed.de/

operation	1st invocation	2nd and later
get template	10ms	1ms
create agreement	16ms	6ms
get agreement	1ms	1ms
terminate agreement	10ms	2ms

Table 3.2 Performances of WS-Agreement implementation.

3.6 Experimental Results

3.6.1 Performances

An experiment is performed to test the performance of the WS-Agreement implementation. The underlying platform is a machine with dual Intel Xeon 2.8GHz processors with Linux system kernel 2.4.20.

As we concentrate on the performance of WS-Agreement support infrastructure, we would like to avoid interactions with the resource monitor, performance predictor and the scheduler. Therefore, dummy providers are used, which also provides logging functionality.

The performances are measured for four different tasks: agreement template retrieval, agreement creation, agreement query, agreement termination, with clock time recorded for each invocation. Table 3.2 presents the result of the experiment. We distinguish between the clock time for the first invocation and an average of the following ones.

Agreement creation is a more complex operation; therefore it takes the longest time to complete. The first invocation for retrieving template, creating agreement and terminating agreement always takes longer than the following invocations. This is because of class loading, resolving and instantiating the agreement providers. The first retrieval of agreement is almost as fast as the later invocations, because the relevant classes are already instantiated and no providers are involved.

3.6.2 Examples

In order to examine the functionalities of the implementation, we have extended the WS-Agreement support infrastructure to support example Grid services for distributed data mining (more details of the services as well as the configuration of agreement providers will be provided later in Section 8.5). Different operations of the infrastructure are performed, and the resulting agreement template, agreement offer and the established agreement are listed in Appendix B.

3.7 Conclusion

Resource management for service-oriented Grid requires flexible and general languages for SLA specification. As an active member of OGF GRAAP working group, we have actively

participated in the finalization of WS-Agreement specification.

An implementation of the specification on top of Globus Toolkit 4 (GT4) has also been developed as part of the Automated Resource Management for Large-Scale Applications project. It follows a layered design that separates the service interface, agreement management and monitoring, as well as backend providers for extension and customization. Our implementation exceeds in its generality and extensibility. For Grid systems, its concurrently support for multiple services of different domains that are dynamically registered is important.

As part of the resource management infrastructure, the agreement manager has to cooperate with other components, including those for performance estimation and job scheduling. In addition, it also needs a global infrastructure that assists the establishment of SLAs among multiple clients and services. They will be discussed in the following chapters.

Chapter 4

Prediction-based Application Performance Evaluation

4.1 Introduction

The resource provisioning capability of Grid depends on dynamic choice of resource for a specific demand. For such a dynamic scheduling to be possible, detailed specification of resource requirements is needed, including the type and amount of resource to be consumed. However, in many situations, the resource demand is indirectly specified with application specific parameters which have to be mapped to the type and amount of resource demands.

Many techniques can be used to predict the performance of applications, like analytical modeling, statistical simulation, and historical data analysis. The applicability of each technique differs. Among the many techniques, performance prediction using historical data is a more general technical that can be applied to a broad range of applications. The prediction of program performances from historical data falls into a more generic category of problems called "numeric prediction". It takes as input a set of (numerical and nominal) variables and outputs a numerical value as prediction. As this is already an intensively investigated field in statistics and artificial intelligence under the name of data mining, our focus is not on the invention of new techniques. Instead, we aim to follow a systematic approach and establish a generic infrastructure for performance prediction, with a focus on its application in a service-oriented Grid. The infrastructure should incorporate predefined performance predictors using standard data mining techniques or custom predictors utilizing other performance prediction techniques.

The rest of this Chapter is organized as follows. Section 4.2 explains the need for performance prediction in our SLA-based resource management infrastructure. Section 4.3 presents a general introduction of application prediction techniques that are feasible for practical usage, including analytical model, statistical simulation, and analysis of historical data, together with a detailed evaluation of the applicability and efficiency of common data mining techniques in performance prediction, including statistical regression, decision tree, nearest neighbour, and artificial neural networks. Section 4.4 presents a systematic approach for performance prediction using data mining techniques, which is supported with a generic run time monitoring and prediction framework for service-oriented Grid in Section 4.5. Section 4.6 discusses implementation issues. Section 4.7 summarizes the related work. Section 4.8 provides experimental result that evaluates the efficiency of application param-

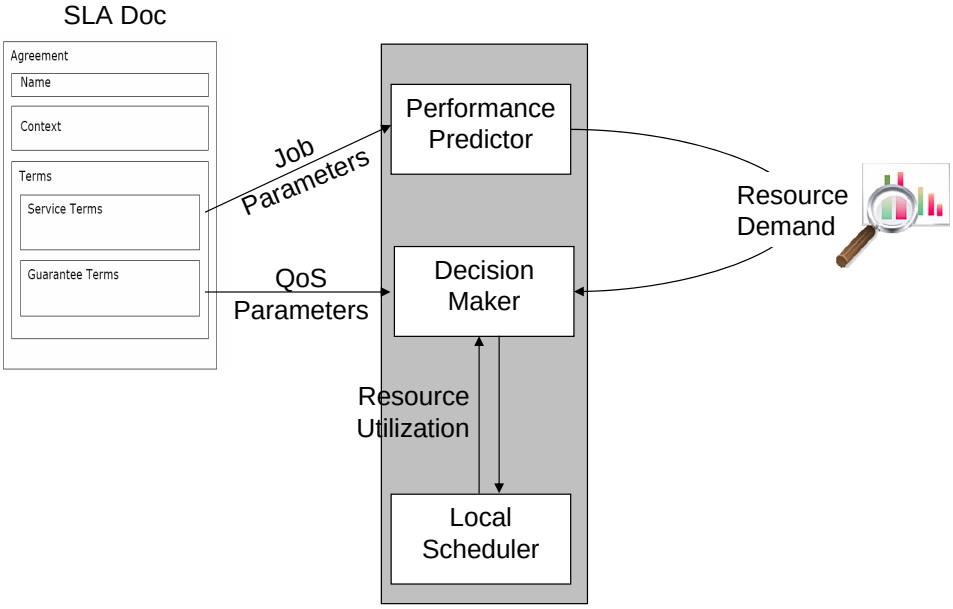

Figure 4.1 The need for performance prediction in SLA decision.

eter and run time recording, and performance modeling and prediction. Finally, Section 4.9 presents a brief summary.

4.2 The Need for Performance Prediction

The WS-Agreement language and the corresponding support infrastructure presented in Chapter 3 allow service consumers and providers to establish a formal contract for service usage. During this process, the service provider must evaluate the QoS requirements specified in the contract and decide whether it will accept it. A major criterion for this decision is whether it will be able to meet the requirements based on availability of resource and an estimation of resource demand or application performance with application specific parameters. While resource availability can be retrieved from the local resource manager, the application performance must be estimated using performance prediction techniques (see Figure 4.1).

The two application scenarios discussed in 2.4 also represent two typical scenarios for performance prediction. In the distributed data mining services scenario, performance prediction focuses on the pre-deployed service or its backend application. In the high performance computing center scenario, performances of arbitrary applications submitted by the user have to be predicted. While the former represents custom application services that are common in commercial usage of the Grid, the latter is more common in Grids for scientific purpose.

For a pre-deployed service or application, we can model its performance with techniques that require insight of the code, apply different performance analysis tools, or per-

	Generality	Feasibility	Prediction Type		Overall Accuracy
			Application	System	
Analytical Model	very poor	poor	yes	partially	varies
Statistical Simulation	poor	acceptable	no	yes	moderate
Historical Data Analysis	good	good	yes	yes	good

Table 4.1 A brief summary of application performance prediction techniques.

form offline analysis that takes long time. Despite their difference in model building, performance prediction in both scenarios has to be done in real-time.

4.3 Feasible Application Performance Prediction Techniques

Many techniques can be used to predict the performance of applications, however only a few of them are feasible for practical usage. A brief summary of such techniques are given in Table 4.1, which gives comparisons depending on the following factors: generality - technique can be applied to different cases, feasibility - how easily the technique can be used, accuracy of the prediction in general, prediction type - “application” means the performance of application with different input parameters and “system” means the performance of an application with different system configurations.

4.3.1 Analytical Model

Analytical models (ref. Section 4.7.1 for examples) attempt to derive an analytical equation that describes application performance using variables from application parameters and system characteristics.

Analytical is probably the most direct approach to developing a predictive performance model. It can be applied to predict the performance of an application with different input parameters and sometimes it is also possible to partially predict the application performance with different system configurations (e.g. [11]).

A major restriction of analytical model is the requirement of in-depth knowledge of the program or minimally the algorithm employed. For programs that present complicated/subtle interaction behaviors, deriving an analytical equation can also be very hard (if not impossible at all). It is infeasible to take architectural details and detailed interaction between the program and architecture into the model. In addition, analytical models are limited in scope to the concrete application and the specific system architecture being modeled.

4.3.2 Statistical Simulation

Due to the complexity of modern computer architecture and the immense number of instructions executed in average applications, detailed simulations are not applicable for per-

formance prediction due to the huge slowdown. A common approach for simplifying the models of parallel applications and systems relies on statistical properties. The application is modeled as code blocks separated by communication events, with statistics of memory access and floating-point operations for the code blocks derived from profiling. The system model is simplified with benchmark measurements that represent the speed of accessing different levels of memories, floating-point calculation, and communication latency and bandwidth.

Statistical simulation provides a feasible approach for predicting the performance of certain types of parallel applications (mainly those for numerical simulation) under different system configurations. It is not applicable, however, to predict the performance of an application with different input parameters. Another difficulty comes from the fact that existing tools (ref. Section 4.7.2) are mostly experimental and still under development.

4.3.3 Historical Data Analysis

Techniques have been proposed for predicting application performance using historical information. They rely on techniques originated from statistics, data mining, machine learning, fuzzy logic etc., to (semi-)automatically discover the hidden relationship between application performance and application parameters and/or system characteristics.

Compared with other methods, techniques based on historical data are more general because no direct knowledge about applications and the execution environments is required. This makes it exceptionally attractive in a heterogeneous environment with various types of applications and systems with different characteristics like the Grid. Depending on the concrete scenario where this technique is applied, historical data contains application parameters, system characteristics, or both. Correspondingly, it can be applied to both application performance prediction and architecture performance prediction. As it is able to capture unexpected details of the interactions between application and systems, the accuracy of historical data analysis can be usually quite good.

Common data mining techniques and algorithms can be evaluated according to their applicability and efficiency. Two types of data mining techniques - clustering and association rule mining can be immediately excluded from our discussion as they are not for predictions. A brief summary of the applicability and efficiency of other common data mining techniques is given in Table 4.2, and more detailed discussions are provided in the following subsections.

Statistical Regression

Statistical regression tries to describe the relationship between the input attributes and the target attribute with a linear or non-linear formula. This technique, though feasible at certain circumstances, is not generally applicable due to difficulties of finding an appropriate type of regression function and performing appropriate transformations.

Prediction Technique	Incremental	Input		Continuous Output
		Discrete	Continuous	
Regression	no	yes †	yes	yes
Decision Tree	generally no	yes	yes	yes ⊥
Nearest Neighbor	yes	yes †	yes ‡	yes
Neural Network	yes	yes †	yes ‡	no ⊣

Table 4.2 A brief summary of data mining algorithms for application performance prediction (†: need transformation, ‡: need normalization, ⊥: requires specific algorithm, ⊣: with workarounds).

Decision Tree

A specific type of decision tree, the regression tree, can be applied to numerical prediction. Regression tree (ref. [27]) is a kind of predictive model based on recursively splitting data into a tree of partitions. Each tree node is a super set of its children, and each child node contains a sub set of data with better consistency in the prediction value. A simple but common approach is to divide the data by applying a threshold on a single column.

A regression tree is a special type of decision tree that permits continuous target values. The advantage of regression tree is that it requires little preprocessing. With respect to a variety of predictor types (e.g. number, categorical etc.), the algorithm is fairly robust [9].

Although regression tree can be run relatively quick, it is expensive for continuous inputs [42]. This constitutes a major drawback in performance prediction.

Nearest Neighbour

The k-nearest neighbour algorithms (kNN, ref. [8]) predict the output value(s) for a given query point by applying a weighting function to the output values of the k nearest instances as defined by the instance metric.

kNN is a kind of instance-based learning [37], which requires no explicit training phase and the collection of data directly acts as knowledge base. Another advantage of kNN is its capability to capture *ad hoc* features of the application performance [68].

The major problem of such algorithms is the somehow arbitrary selection of the weight function which does not necessarily reflect the relative importance of each parameter correctly. Other issues of this technique include performing appropriate transformations, difficulty of avoiding outlier (especially for simple nearest neighbours with k=1). Performance is a potential issue and caution has to be taken to choose the correct algorithm. Simple algorithms do not perform well if the number of attributes is over 10, while sophisticated algorithms like kD-Tree or M-Tree can create metric trees that deal successfully with thousands of dimensions [123].

Artificial Neural Networks

Artificial neural network (ANN, ref. [98]), often just called "neural network", is an analogy of biological neural networks. It consists of interconnected artificial neurons as processing units, which respond in parallel to a set of input signals given to each using its small sphere of knowledge and local memory of data. ANN usually adapts its structure based on external or internal information that flows through the network during the learning phase.

The greatest advantage of ANNs is their ability to learn unknown, complex and dynamic relationships directly from the data. Another major advantage is their robustness towards noisy data [32], which makes them well suited for application performance predictions with usually a presence of system noise in the performance data.

A minor complexity of applying neural networks in performance prediction is that preprocessing is usually necessary [9]. Continuous input attributes need to be normalized and categorical predictors are broken up into virtual predictors that are 0 or 1 for each value of the original categorical predictor. However, many software packages that implements data mining algorithms also support certain automation of those preprocessing.

In the following discussions, we will concentrate on the application of data mining techniques in application performance prediction.

4.4 A Systematic Approach for Performance Prediction with Data Mining Techniques

4.4.1 Overview

Some preliminary research on performance prediction using historical data analysis exist (ref. 4.7.3). However, all existing studies are limited to the evaluation of a specific analysis method, and many are restricted to data of a specific format that are collected for certain application or public workload archives of computing centers. Such studies are all right in proofing the applicability of data mining techniques for performance prediction, but the approach can not be followed in general. On the one hand, mature software tools or libraries are available that support more advanced data mining techniques. Therefore, there is no need to invent the wheel another time. On the other hand, it is a known fact that different data mining techniques perform differently with different data. The selection of a data mining technique and the concrete algorithm is also a major factor that affects prediction accuracy.

Different Grid services have different application parameters and performance characteristics. To support performance prediction and consequently SLA-based resource management, we need to follow a systematic approach that can be generally applied to different types of services. The essence of this approach is to apply standard data mining process and techniques on performance data.

Figure 4.2 is a simplified schematic figure that shows a typical data mining process. The collected data are separated into test data, training data, and production data. The training data is used by the modeler to build models. The quality of the model is estimated

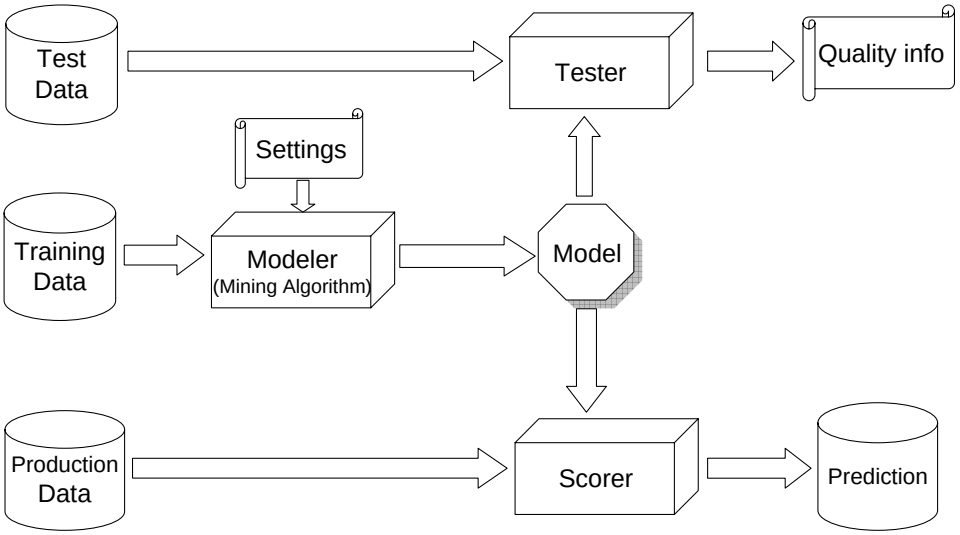

Figure 4.2 A simplified data mining process.

by the tester against test data. The quality information can be used to tune parameters of the model. The model can be applied on production data by the scorer to get the prediction.

4.4.2 Performance Data Collection

For performance prediction purposes, two types of data need to be collected - application parameters and performance data.

In the service-oriented Grid, application parameters can be collected in a straightforward way, since they are often specified in the SLA document and/or the service invocation message. The performance data, on the other hand, has to be collected by monitoring. While most existing Grid monitoring systems focus on the performance statistics of the system, several are capable of monitoring application performance and can be applied to computational services (for example, GRM [89], NetLogger [59], and Autopilot [116]).

The collected data has to be transformed into standard data types, including continuous numerical values (e.g. 1.2), discrete numerical values (e.g. 5), categorical nominal values (e.g. red), and nominal values (e.g. large). For example, a fixed set of variables of the same type can be transformed into several independent inputs, and a variable set of variables of the same type can be transformed into a variable matrix.

4.4.3 Performance Data Mining

Once the application parameters and performance data are collected, standard data mining techniques can be applied. This process can be roughly divided into two phases - modeling and predicting. Modeling, or training, is the process of creating a model using historical data. And predicting, or scoring, is the process of applying the model to unseen data to make new predictions. For instance-based mining techniques like kNN, no explicit model-

ing process is necessary.

Two parameters play key roles in performance prediction: the predicted performance, and the prediction accuracy. While the predicted performance can be derived by applying the model to the data (program parameters), the prediction accuracy can only be derived by evaluation, which is also part of the performance modeling step.

As is discussed in Sub-section 4.3.3, common data mining techniques that are feasible for application performance prediction include regression tree, nearest neighbour, and artificial neural networks. Each technique is supported by different algorithms, and many are implemented by existing software or libraries. Some advanced ones like Weka [122] are capable of automated parameter tuning. A major advantage of using the standard data mining techniques is the possibility to reuse those mature algorithms and software. Optionally, the performance of different data mining techniques can be compared for selection.

4.4.4 Data and Model Management

With new records of application parameters and performance data collected, the existing model need to be updated. For instance-based learning techniques where the data is the model, each update of data also updates the model.

The frequency of model update depends on the characteristics and performance of selected data mining algorithm. As is mentioned in Section 4.3.3, both kNN and ANNs are incremental which can be updated as each new record comes. For regression trees and other techniques that are slow and non-incremental, the model can only be periodically updated with a background process.

Depending on the concrete scenario, a two layer organization of data and models similar to [68] can be optionally applied. In addition to the normal data and model, a local layer is added that contains only recent data and model built out of the recent data and represents the short-term performance of the system. This helps to cope with short term performance perturbations.

4.5 A Run Time Monitoring and Prediction Framework for Grid Services

4.5.1 Overview

While the systematic approach described in Section 4.4 provides a general guideline, actual application of that approach demands a lot of effort. A uniform framework that provides infrastructure support for the collection of performance data and performances prediction is needed. The framework, as is shown in Figure 4.3, features generic interfaces for modeling and prediction, data collection with snoop in the SOAP handler chain, and automatic management of performance models. This framework has the following advantages:

1. Non-intrusive: it can be plugged in to the Grid environment without modification of the service implementation. It also means that the process of performance data

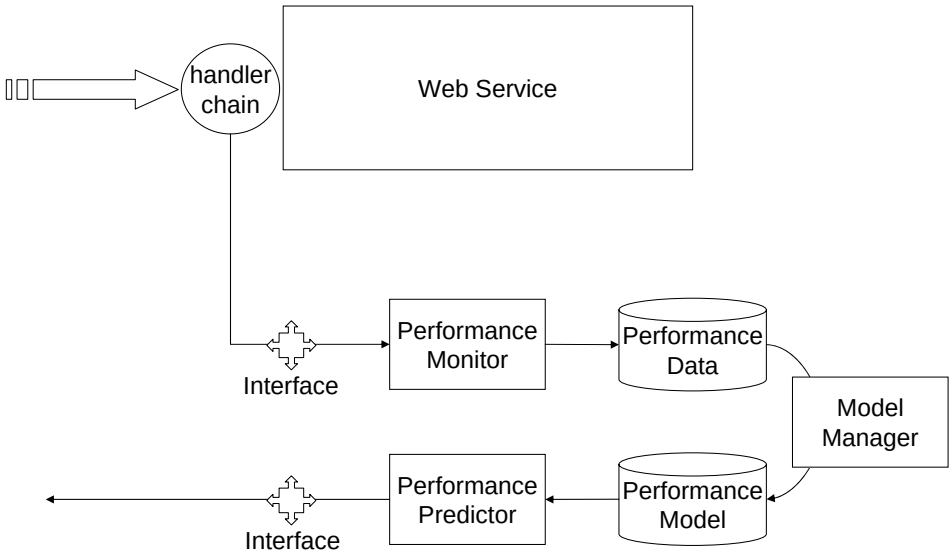

Figure 4.3 A run time monitoring and prediction framework for service-oriented Grid.

collection and performance prediction will not have much influence on the service execution performance.

2. Adaptable: it can be adapted to different types of services, including standardized services with well-known interfaces like the Globus GRAM etc, or customized services with specific port type definitions.

3. Generic: it allows different performance prediction techniques employed under the same hood, including different data mining techniques for historical data analysis as well as custom models with analytical modeling or statistical simulation to be applied.

4. Architecture neutral: the framework does not depend on a specific resource monitoring, neither is its usage restricted to a specific SLA management infrastructure.

4.5.2 Non-intrusive Data Collection

The collection of application parameters and performance data should not require the modification of existing services to be monitored. This is achieved by inserting a handler into the SOAP processing chain of the SOAP engine. Upon service invocation, this handler is invoked by the SOAP engine and deserializes the message into values. These values are recorded with a timestamp. Once the service invocation finishes, the previous record is updated with the job finish time and total length of job execution.

While the timestamp can be retrieved directly from the system, the parameters are derived by deserializing the SOAP message. In addition, other performance or utility data can be collected by monitoring tools. Depending on the exact environment and data to

be collected, different tools can be used, for example the Java Management Extensions (JMX) [95] to monitor application services themselves, Windows Management Interface (WMI) [71] for reading Windows performance counters, and GANGLIA [82] monitoring tool available for both Linux and Windows for unified systems monitoring.

4.5.3 Generic Modeling and Prediction Interface

The framework should be able to incorporate different prediction techniques. This is enabled by the definition of general interfaces and pluggable performance recorders and predictors.

The general interface isolates the performance recorder and predictor with other parts of the resource management infrastructure. It also regulates the different implementations of performance recorders and predictors that can be plugged into the framework. A manual configuration of performance recorder and predictor can be specified in a local configuration script.

4.5.4 Automatic Management of Performance Model

Model manager manages the periodical update of models under predefined or user supplied policies. It also takes care of the actual performance recorders and predictors that are applied, which means the instantiation of the corresponding class and delegation of the actual work to that instance, in case that a manual configuration of the performance recorder and predictor exist.

Because of the vastness in possible algorithms and parameters for different mining methods, manual choice of appropriate models is a challenging task, which includes the application of different mining methods on the performance data to build performance models, evaluate the models in different ways, adjust model parameters accordingly, and re-evaluate the model. Semi-automatic configuration and evaluation of different performance modeling techniques against the specific application can be established by managing a pool of pre-configured models. The model manager periodically updates and evaluates the models in the pool, and applies the model that produces the most accurate predictions within relatively shorter period of time for the specific service.

4.6 Implementation Issues

4.6.1 The Globus SOAP Handler

In a typical SOAP engine, SOAP messages go through a sequence of handlers each doing specific processing. Axis SOAP engine distinguish three types of handlers (see Figure 4.5) - transport handlers for protocol-specific processing, global handlers that are applied to all services and service handlers that are applied to a specific service.

Globus WS Core is based on Axis SOAP engine and provides several global handlers for addressing, authentication, authorization and fault handling. Our handler should be placed behind all these handlers, with three possibilities shown in Figure 4.5: (A) add a

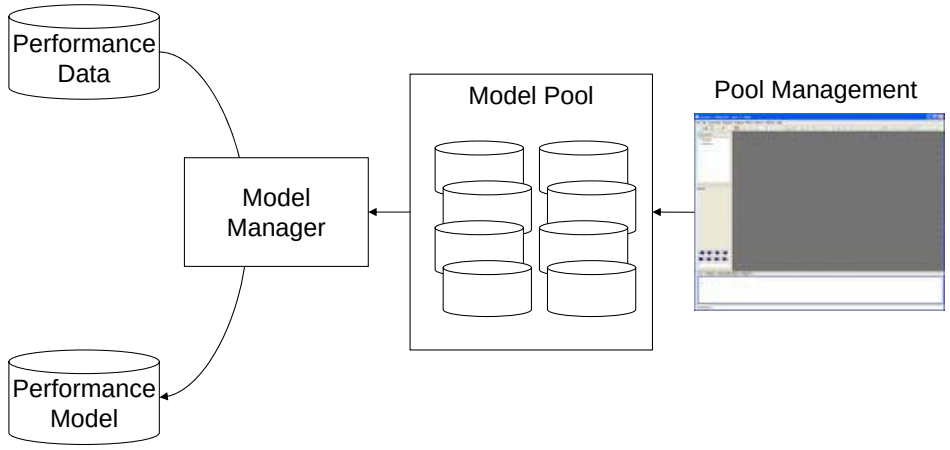

Figure 4.4 Automatic management of performance model pool.

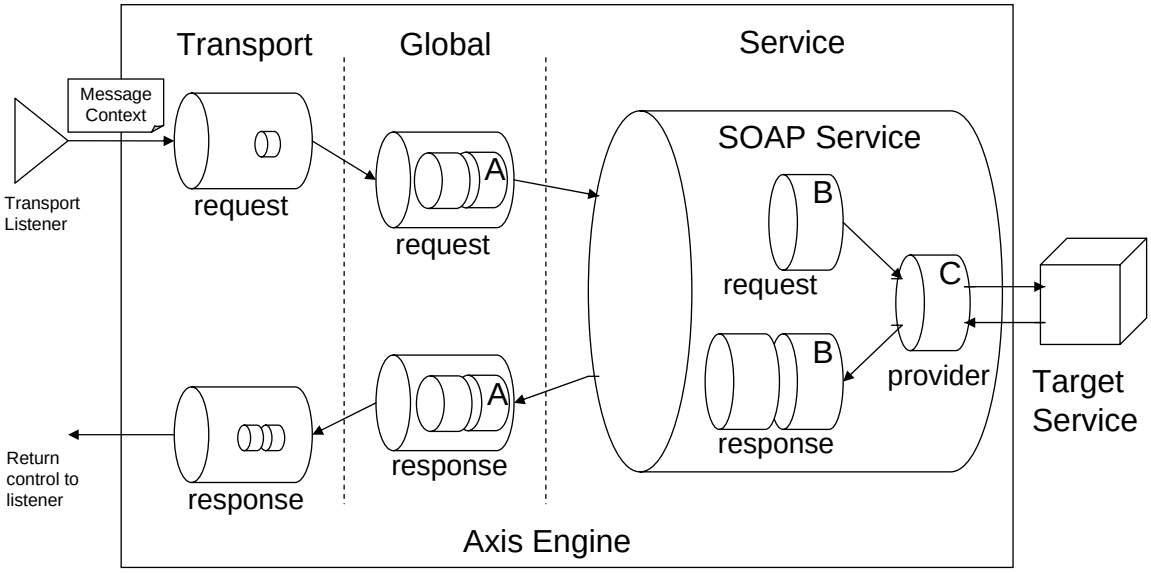

Figure 4.5 Possible locations (A, B, C) for data collection handler.

pair of global handlers as the last one in global request handler chain and the first one in global response handler chain, (B) add a pair of service handlers each for request and response, (C) replace the pivot handler (provider) that invokes the actual service.

In our implementation, we chose the last option and implement a RecordingRPCProvider that extends the existing RPCProvider class. The major benefit of this approach is that we can reuse the existing code in RPCProvider class that parses the service arguments. By modifying the service deployment descriptor, our provider can be automatically inserted into the handler chain when the service is deployed:

```
<service name= ...>
  ...
```

```
    <parameter name="handlerClass"
      value="de.tum.in.lrr.sog.RecordingRPCProvider"/>
    ...
  </service>
```

Before invoking the service method, the RecordingRPCProvider calls argument recorder to record the arguments. And after the service method returns, it calls time recorder to record the run time. In order to allow some flexibility in data storage methods (e.g. files or data base), the argument recorder and time recorder can be customized by implementing predefined interface.

4.6.2 Performance Data Storage

We chose Derby[1] in our implementation for data storage. Derby is a pure Java relational database that can be embedded into Java applications. This makes it possible to automatically start the database when starting the Grid server, and by automatically managing the database and data tables, the administrative effort is minimized. With embedded JDBC driver, database accesses are directly in memory, which can be much faster than through socket.

Derby is embedded into Globus with the help of a dummy Web service. In the deployment descriptor of that Web service, it is configured to load on server startup. The Web service class implements javax.xml.rpc.server.ServiceLifecycle interface, and implements init() and destroy() method that will separately be invoked at server start up or shutdown. The embedded driver is loaded and the database is started in the init() method. In the destroy() method, all connection are closed and the database is shutdown.

The collected data are stored in the database with a table following specific schema. Given the fact that different services have different number of parameters, we define a pattern for the database schema:

```
ID INTEGER NOT NULL GENERATED ALWAYS AS IDENTITY,
START TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
FINISH TIMESTAMP,
LENGTH BIGINT,
VALUE_1 <type1>,
...
VALUE_n <typen>
```

The table is created dynamically when the data recorder is invoked for the first time, and the number of values and the type of value are automatically determined for the database. The mapping between Java types and SQL types is performed with the help of a mapping table.

There is a one to one mapping between the services and the database tables that contain performance data. The mapping between service and table is achieved by enforcing certain rules of the table names, which contains a prefix and the service ID.

[1] http://db.apache.org/derby

4.6.3 Performance Prediction

Our data modeling and prediction implementation is based on Weka [122] from the University of Waikato in New Zealand. Weka is a large collection of state-of-the-art machine learning algorithms for data mining tasks. It contains tools for pre-processing, classification, regression, clustering, association rules, and visualization. It is written in Java and can be easily extended with new machine learning schemes and algorithms.

The current version of Weka has a limitation of database access - parameters for database access including table name must be provided with system properties. However, there might be multiple services deployed on the same Grid server, the performance data of which are stored in different tables or even different databases. Therefore, we have reimplemented some part of Weka to enable the usage of data from existing database connections.

As is mentioned in Section 4.6.2, although all the performance database follow the same pattern, the exact number of data fields in the database schema differs from service to service. We use database metadata to dynamically determine the name and type of each data fields that represent application parameters.

Weka data mining algorithms are Java classes and can be directly invoked from our program. First, an instance of the corresponding miner is created. Then, it is applied on the data retrieved from the database to build the model. With cross validation, its prediction accuracy can be estimated. The miner, with its model inside, can then be applied on a data instance. And the predicted values are returned together with the estimated accuracy.

4.6.4 Configuration and Management of Recorders and Predictors

The performance recorders are configured on a per-service basis through the service deployment descriptor:

```
<service name= ...>
  ...
  <parameter name="argumentRecorder"
    value = "de.tum.in.lrr.sog.DatabaseArgumentRecorder"/>
  <parameter name="timeRecorder"
    value = "de.tum.in.lrr.sog.DatabaseTimeRecorder"/>
  ...
</service>
```

Performance predictors are accessed by the agreement decision maker, which is out of the scope of the specific service and a configuration script like that of the performance recorders can not be directly accessed. Therefore, a registry of available providers is implemented to globally publish the configuration in the Grid platform. The registry is a mapping between the service parameters and the corresponding performance predictor. The registrations of performance predictors are performed proactively by the custom services in a way similar to the approach described in Section 3.4.4.

4.7 Existing Work on Application Performance Prediction

4.7.1 Analytical Model

Most work on analytical modeling is done against smaller application or kernels. [117] analyzed sparse matrix-vector multiply (SpMV, $y = y + Ax$, where A is a sparse matrix and x, y are dense column vectors). [72] analyzed symmetric sparse matrix-vector multiply (A is a symmetric, sparse matrix, i.e. $A = A^T$) and symmetric sparse matrix-matrix multiply (SpMM, $Y = Y + AX$, where X, Y are dense matrices).

In many cases, the analytical models are only able to model the upper and lower bounds. For example, [118] analyzed the upper and lower bounds of sparse matrix operation SpA^TA, i.e. $y = A^TAx$, where A is a sparse matrix and x, y are dense vectors.

For large scale scientific applications, [97] analyzed the sequential and parallel performance of an application that simulates the collision of two black holes from the Cactus [2] software package [11]. [19] presents an analytic performance model of Krak [29], a large-scale parallel hydrodynamics code irregular mesh partitioning. A series of research by the Los Alamos National Laboratory model the performance of several applications for the simulation of deterministic particle transport on structured meshes [61], unstructured meshes [83], non-deterministic particle transport using Monte-Carlo simulation [84], and adaptive mesh refinement [69].

Analytical modeling other than scientific simulation also exists. Examples include the performance modeling of 3D isosurface visualization software [24], and that of a specific type of databases that uses indexed file as the physical storage mechanism [43].

A methodology of composing analytical models of individual components into a general model of whole application is discussed by [101], and [52] discussed the same issue with "kernel coupling". The impact of stochastic values on model parameters is discussed in [102, 104].

4.7.2 Statistical Simulation

There are many research activities in this area. While detailed summary and comparisons among existing toolsets can be found in [115] and [36], we review some typical ones here to provide hints about the current status and applicability of current tools.

PERC/PMaC framework

The PERC framework [108] is a collection of tools for gathering machine profiles and application signatures and provides automated convolutions. The tools are separately developed and collaborately integrated by several labs and universities from USA and Europe.

It provides a benchmark application MAPS (Memory Access Pattern Signature) [110] to measure single processor signatures, with emphasis on memory access patterns. The result can be used in combination with the third party benchmarks that measure network

[2] http://www.cactuscode.org/

(MPI) performances like PMB (Pallas MPI Benchmarks) [3], or NPB (NAS Parallel Benchmarks) [18].

For application profiling, it uses MetaSim Tracer [110] to trace memory accesses and floating-point operations, and uses MPIDtrace [17] to record communication (MPI calls).

For statistical simulation, DIMEMAS simulates the execution of the application on a parallel system, based on the MPI event trace produced by MPIDtrace to simulate the execution of the application on the target system. It uses latency and bandwidth to model the communication time. The run time between MPI events is estimated using MetaSim Convolver [110], which can be used to estimate the run time for the whole application or parts of the application on a single processor. The specification of the target system is via configuration file, which can be user parameterized or measured with benchmarks.

The PMaC Prediction Framework[3] is an upgrade of the PERC framework. The MetaSim Tracer, as a binary instrumenter tool on top of the ATOM toolkit for Tru64 Unix on Alpha AXP processors, has been replaced by PMaCinst, an instrumentation tool for XCOFF binaries on AIX for PowerPC processors. The MetaSim Convolver becomes the PMaC Convolver, and MAPS becomes MultiMAPS. A migration of the MetaSim Tracer based on DyninstAPI is also reported [109].

PACE

PACE (Performance Analysis and Characterisation Environment) [90] is a modeling and performance analysis toolset for high performance and distributed applications. It is based on a layered infrastructure formed by extensible objects describing each of layer: application, subtask, parallel template (computation-communication interactions), and hardware.

The system model is formed by hardware objects, which are organized in a hierarchical way to describe the hardware system in general or individual parts like memory, CPU, and communication system. The hardware objects are custom developed in C language and collaboratively form a hardware object library.

The application model is described with CHIP^{3}S (Characterisation Instrumentation for Performance Prediction of Parallel Systems) language [94], and can be semi-automatically generated using ACT (Application Characterisation Tool). ACT performs static analysis of the source code to produce the control flow of the application, operation counts in terms of SUIF language [121] operations, and the communication structure. Dynamic performance related aspects of the application, such as data dependent parameters, are obtained either by manual analysis or profiling using tool AddSensors [10].

A compiler translates CHIP^{3}S scripts to C code which are linked with an evaluation engine and the hardware models. The final output is a binary file which can be executed rapidly. The user determines the system/application configuration and the type of output that is required as command line arguments. The model binary performs all the necessary model evaluations and produces the requested results. PACE includes an option to generate predicted traces (PICL [54, 53], SDDF [16]) that can then further analyzed by visualization tools (e.g. PABLO [96]).

[3] http://www.sdsc.edu/pmac/projects/

	first invocation	2nd and later (average)
parameters recording	106ms	6ms
run time monitoring	3ms	3ms

Table 4.3 Overhead of application parameter and run time recording.

4.7.3 Historical Data Analysis

There are relatively few researches on the application of historical data analysis in application performance prediction. Most work focus on case studies for a specific program. For example, a group of researchers has [64] applied multilayer neural networks to predict the performance of SMG2000, a semi-coarsening multigrid solver based on the *hypre* library [41]. It uses a "multilayer fully connected feed-forward neural network" with one hidden layer and sigmoid activation function.

The k-nearest neighbour method with three different weighting functions - unweighted average, weighted average, and locally weighted polynomial regression - is used in [68] to predict resource usage for the PUNCH (the Purdue University Network-Computing Hubs) [67] system, a Web portal for executing tools on servers at Purdue University. [73] improved the above technique by using a genetic algorithm to automatically search for optimal parameters and applied it on workload traces.

Very few researches also attempt to apply other algorithms that are not very popular. [14] presented an approach based on a genetic algorithm and fuzzy logic which allows for creation of robust prediction models even with scarce training data. [38] presented an approach based on a state-transition model.

4.8 Experimental Results

Experiments are performed to test the performance of run time monitoring and prediction infrastructure. The experiments are made using a custom service for ray tracing that invokes a famous open source ray tracing program Povray [4] in the backend. The experiment platform is a machine with dual Intel Xeon 2.8GHz processors with Linux operating system. The Linux kernel version is 2.4.20 and the exact version of the Povray program is 2.6.

4.8.1 Efficiency of Data Recording

To evaluate the performance of the run time and application parameter monitor, we manually instrumented the service source code to measure the amount of time spent on the recording of application parameters and runtime. A test client is implemented that continuously invokes the service with random parameters. The result of the experiment is shown in Table 4.3.

[4] http://www.povray.org

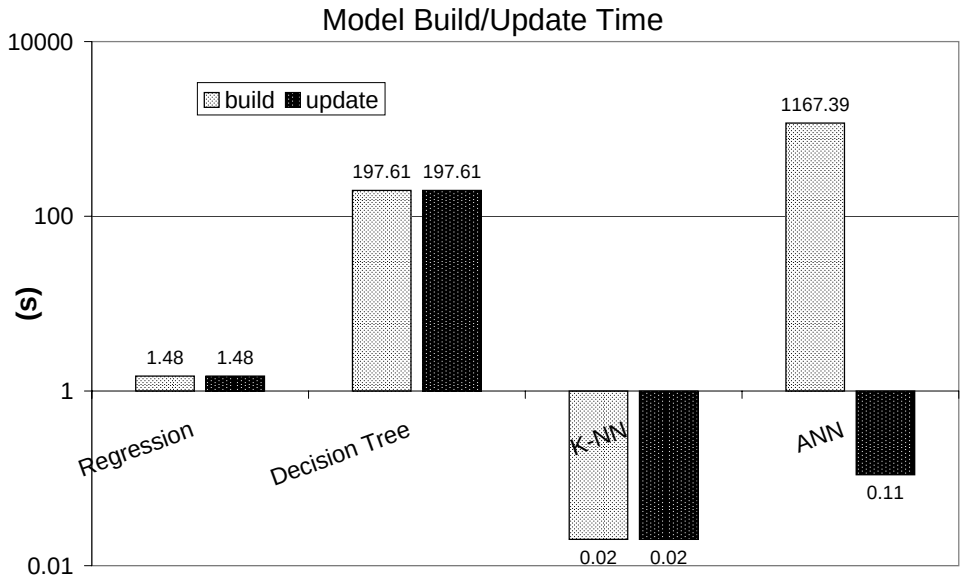

Figure 4.6 Run time for build and update performance prediction model.

The recording of application parameters for the first time takes longer than the others. The excessive time is spent on the creation of database table. The recording of service parameters takes longer than that of the run time. This is because of the additional processing of the recorder to dynamically adapt to multiple and variable parameters, including the determination of the parameter type and the construction of corresponding SQL statement.

4.8.2 Efficiency of Model Building and Prediction

The application parameters collected are the most important and typical ones that Povray supports, including scene file, image width, image height, and image quality. The parameters cover several typical types of data, including nominal, continuous and discrete numerical. We choose 12 scenes out of the advanced sample scenes as part of the povray distribution. For each scene, we sampled image width from 200 to 2000 with a gap of 200 and image quality from 1 to 0 with a gap of 0.1. This means a total of 9x10x10 samples for each scene.

The performances of the learning algorithms are evaluated mainly interms of prediction time. Prediction accuracy, which is know to be completely application specific, is not of our interest here. The result is shown in Figure 4.6 (note that y axis is in logarithm). The time for building performance model with all the samples and the time for updating the model with the last sample are separated. Regression and kNN method performs very fast. And decision tree is a bit slower. ANN might give an initial impression that seems to be relatively slow. However, taken the fact that ANN is incremental, the run time for each update is greatly reduced.

To give a little bit more insight, Figure 4.7 provides an example of the learning curve measured using ANN. It shows the improvement of the evaluation accuracy with increasing

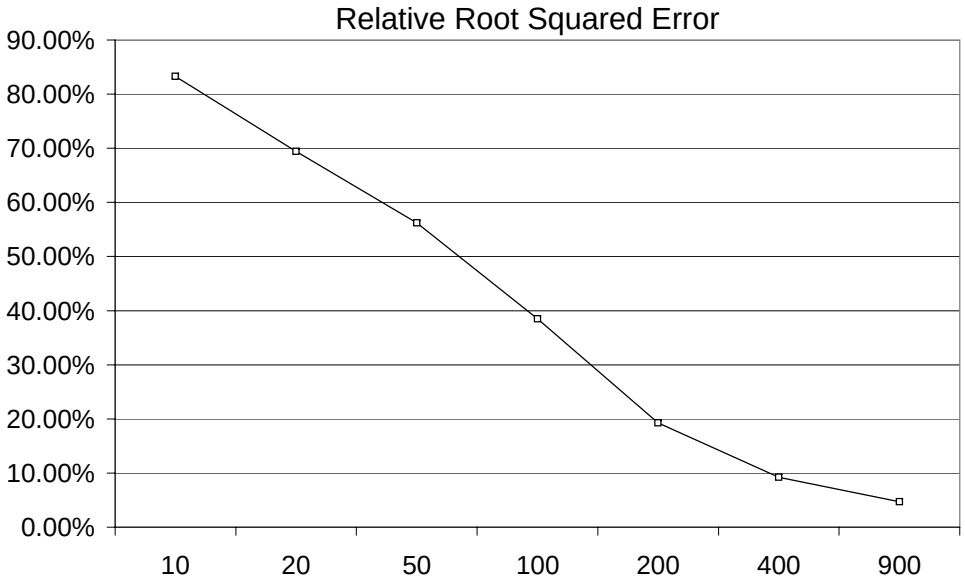

Figure 4.7 Learning curve show how average error improves with training set size.

number of samples.

It should be noted that such prediction techniques can work equally for scientific applications. For example, the same algorithms are applied to workload logs of super computer centers from the Parallel Workloads Archive [44]. Although the resulting prediction is not very accurate due to the fact that application specific parameters are missing from those logs, it clearly shows the applicability of such techniques to predict the run time and memory usage of parallel applications on various high performance computers.

4.9 Conclusion

Agreement decisions require accurate estimation of resource demand or application performance with application specific parameters or different system configurations. Compared with other methods like analytical modeling and statistical simulation, techniques based on historical data are more general. It requires no direct knowledge about applications and the execution environments.

While existing works focus on the application of a specific algorithm, we aim to follow a systematic approach that applies standard data mining process and techniques for performance prediction. To assist the application of such an approach in service-oriented Grid, a generic, adaptable, non-intrusive run time monitoring and prediction framework has been designed and implemented.

Based on application performance prediction, the local resource manager can schedule Grid jobs according to the availability of resources and QoS requirements of the job. While the availability of performance prediction provides an additional opportunity for the scheduler, the stochastic nature of the predicted performance implies however difficulties, all of

which will be discussed in the following Chapter 5.

It should be noted that the prediction-based approach can be applied to a broader range of resource types other than computational. For example, the Network Weather Service [124] utilizes the prediction-based approach to predict network performances. This enables the uniform applicability of the SLA-based resource management approach on different types of resources.

Chapter 5

Job Scheduling for Local SLA Management

5.1 Introduction

As a major part of the local facilities for SLA management, the local resource manager is responsible for the management of local resources that are made available through the Grid. Major functions of the local resource manager include the examination of resource availability for a certain request and the scheduling of service invocations or jobs, which are supplied by the local scheduler.

While normal schedulers focus on minimizing the total execution time (i.e. make-span) of a set of scheduled jobs, the scheduling of service invocations in a SLA-based resource management environment has specific requirements that can not be easily fulfilled by such schedulers. The scheduling is made on-line, where instead of minimizing make-span, the objective is to maximize the number of jobs that can be served. Admission control, i.e. the judgment of accepting or rejecting a scheduling request, is an integrated part of the scheduler. The jobs are normally constrained with deadline and possibly also the earliest start time. In addition, the scheduler has to cope with inaccuracies in the predicted execution time of jobs. For example, the scheduler can decide if the current job execution should be extended if the actual execution exceeds the scheduled finish time.

This chapter focuses on job scheduling problems for local SLA management. More specifically, we deal with a case where the service in execution has exclusive access to the local resource. We start with a basic scenario in 5.2.1 where all the jobs are constraint with individual deadlines. Based on a formal parameterization of this basic scenario, discussions on dealing with probabilistic run time and algorithm for scheduling deadline-constraint jobs are presented. Section 5.3 discussed several more complicated scenarios, including situations where SLA acknowledgement and asynchronous parameter submission are necessary, the user specifies the earliest start time, the job can opportunistically run over its scheduled finish time, and alternative SLA offers are provided. Section 5.4 overviews job scheduling phases before and after a SLA is established. Section 5.5 discusses the design and implementation issues. Section 5.6 presents experimental results that compare the performance of different scheduling algorithms, in terms of run time and acceptance rate. Related work is summarized in Section 5.7. Section 5.8 concludes this chapter with a brief summary.

5.2 Basic Problem Statement

5.2.1 Problem Parameterization

A simplistic description of job scheduling in the SLA-based resource management infrastructure can be accomplished with the following parameters:

1. The job has predefined parameters which can be used to predict the estimated run time $\hat{t}_{job}$ and the variance σ^2.
2. The job has associated SLA parameters, including QoS parameters like deadline $t_{deadline}$ and business values like the price p and penalty (or fine) f.

Based on these parameters, scheduling determines if the required QoS can be fulfilled so that the job is admitted (admission control) and decides when the job should be executed. The objective here is to maximize the total number of jobs that are scheduled. The important lifecycle times of the job include:

1. t_{start}: the start time of the job [1].
2. t_{finish}, T_{finish}: the actual and scheduled finish time of the job.
3. t_{job}, T_{job}: the actual and scheduled execution time of the job.

In our discussion, for simplicity, we assume that once a job is admitted and scheduled, it will not be canceled by the scheduler to move space for later jobs.

5.2.2 Probabilistic Run Time

The possibility of a job finishing before deadline is defined as P:

$$\begin{aligned} P &= P(t_{finish} \leq t_{deadline}) \\ &= P(t_{start} + t_{job} \leq t_{deadline}) \\ &= P(t_{job} \leq t_{deadline} - t_{start}) \end{aligned}$$

.

Since $t_{finish} = t_{start} + t_{job}$, and t_{job} has a truncated distribution ($t_{job} >= 0$) centering on its expected value $\hat{t}_{job}$, the distribution of t_{finish} centers around its expected value $t_{start} + \hat{t}_{job}$ which is schematically illustrated in Figure 5.1. And in a practical schedule where the job is scheduled before T_{finish}, the probability P is calculated as $P = P(t_{job} \leq T_{finish} - t_{start})$.

The utility U of a new job that is scheduled on the server, i.e., the expected benefit of the job, can be evaluated as:

[1] At any time, the actual and scheduled time are always the same. If the job is delayed, a re-scheduling must be performed before the job can be executed, which updates the scheduled time.

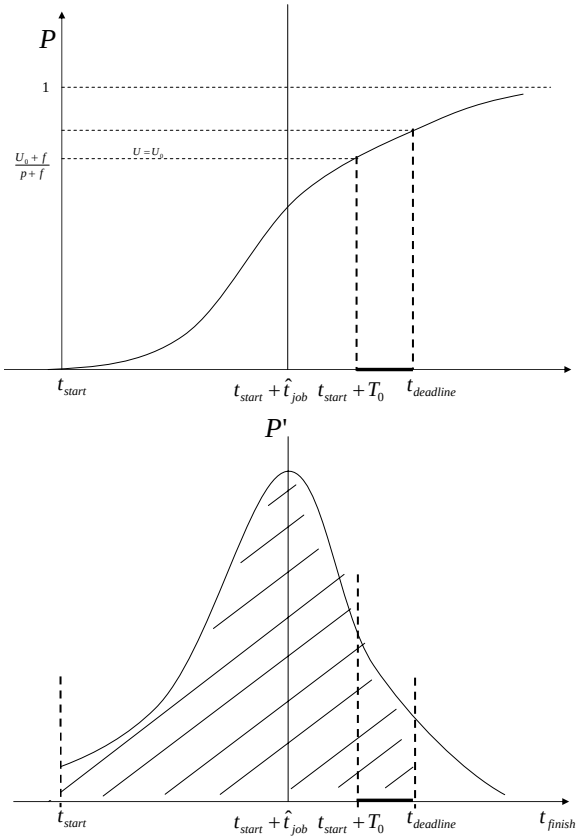

Figure 5.1 Cumulative distribution function (upper) and probability distribution function (lower) of job finish time.

$$\begin{aligned} U &= p \cdot P - f \cdot (1 - P) \\ &= (p + f) \cdot P - f \end{aligned}$$

A minimal utility U_{min} can be used to regulate the execution time of job t_{job} during scheduling. If $U \geq U_{min}$, then $P \geq (U_{min} + f)/(p + f)$. That means, the job must be scheduled with a length greater than T_{min} ($t_{finish} \geq t_{start} + T_{min}$, ref. Figure 5.1), so that the probability of the job finishing before the deadline is greater than $(U_{min}+f)/(p+f)$. In case a minimal utility is not specified, there is also an implicit minimal value of $U_{min} = 0$ because the utility should not be negative.

In practice, the scheduled job execution time T_{job} is determined based on $\hat{t}_{job}$ and its distribution so that the probability is reasonably higher than a certain threshold (*accuracy threshold*). This assures that the job will not occupy too much excessive time without obvious improvements to the utility. This is exceptionally important when the deadline is not very restrictive.

5.2.3 Scheduling Deadline-Constrained Jobs

For the simplest case where the jobs are constraint with deadlines, the scheduling can be based on Earliest Deadline First (EDF) algorithm in which the job with the earliest absolute deadline is executed first. For preemptive scheduling, EDF has been proven to be optimal with respect to minimizing the maximum lateness of n independent tasks with arbitrary arrival times (aporadic task set) [56, 75]. For non-preemptive scheduling, EDF is also proven to be optimal for sporadic task sets among all the non-idle algorithms (the processor is not allowed to be idle when there are active jobs) [55].

To apply EDF algorithm in our infrastructure, it has to be twisted with admission control. Upon the arrival of a new job, all jobs are sorted according to their deadline. A schedule is attempted by continuously selecting and scheduling the job with the earliest deadline until there are no more jobs. If the new schedule can not hold all jobs, the new job will be rejected and the original schedule will be kept. Otherwise, the new job is accepted and the new schedule replaces the original one.

5.3 More Complicated Scenarios

5.3.1 SLA Acknowledgement and Asynchronous Parameter Submission

Two situations have similar impacts on job scheduling:

1. If the agreement offer is initialized by the service provider instead of service consumer, the agreement has to be acknowledged by the service consumer. The service specifies a deadline for the SLA acknowledgement, namely T_{SLA}. The actual SLA acknowledge time t_{SLA} must be before T_{SLA}.

2. For security reasons, the SLA might only contain the job parameters that are critical for SLA negotiation and performance evaluation. The full set of parameters can be encrypted and sent to the service asynchronously, before certain deadline T_{ready} specified by the scheduler. The actual ready time t_{ready} must be before T_{ready}.

The impacts of SLA acknowledge and asynchronous parameter submission on scheduling include:

1. The initial schedule is only temporary, which has to be consolidated by SLA acknowledgement or parameter submission. If the temporary schedule expires and thus has to be canceled, the scheduler can perform a rescheduling to optimize the schedule.

2. If the actual ready time t_{ready} is earlier than T_{ready}, scheduler can determine to keep the current schedule or modify the existing schedule between current to deadline $t_{deadline}$.

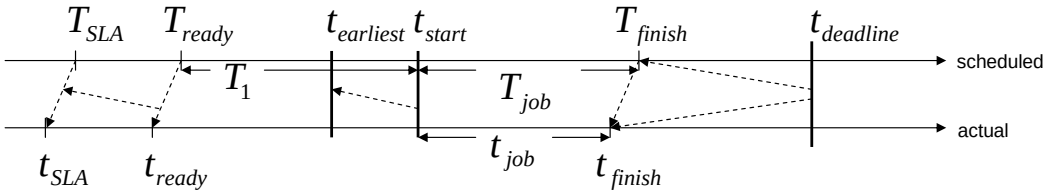

Figure 5.2 Relationship of times in job scheduling.

3. When applying the EDF algorithm, the schedule needs to be adapted so that the job will not be scheduled before T_{SLA} and T_{ready}.

5.3.2 User-specified Earliest Start Time

The user might specify an earliest start time $t_{earliest}$ as part of the SLA. This can happen for the co-scheduling of resources for parallel execution or the scheduling of jobs that constitute a workflow.

The user specified earliest start time $t_{earliest}$ imposes a lower bound on job start time t_{start}, i.e. job start time t_{start} must be later than $t_{earliest}$ (see Figure 5.2). This can be considered as a generalization of the deadline-only case, where the $t_{earliest}$ can be considered to be 0, or now. In another extreme, where $t_{deadline} - t_{earliest}$ is very close to $\hat{t}_{job}$, it becomes the traditional reservation-based scheduling.

For n jobs, there will be $P_n^n = n \cdot (n-1) \cdot (n-2) \cdot \cdot 2$ possible execution sequences. Therefore, finding the optimal schedule by traversing the whole search tree with n jobs has a complexity of $\sim O(n!)$. Pruning the search tree by abandoning the branches when the addition of any node to the current path causes a missed deadline can reduce the complexity, but does not help the worst case. Heuristics for global searching like genetic algorithm or simulated annealing help to reduce the time for looking for possible schedules, but mostly are not able to derive the optimal schedule.

On the other hand, a heuristic algorithm can be derived by adapting the original EDF algorithm to jobs with earliest start times that are first sorted with an ascending deadline and then are scheduled with each job after the previous job and its earliest start time. Although such an algorithm will not give the optimal schedule, it can still be applied to derive suboptimal schedules. This can be inferred from Figure 5.3 where the relationships between two jobs with earliest start time and deadline are illustrated. Except for a rare situation in Case 4 (see Figure 6.3) with same deadline and different earliest start time and two subcases of Case 6 with containment relationship, EDF holds true for their relationship. The distinction between the first two cases and the last case in Case 6 is determined by the length of jobs, which can not be generalized to multiple jobs.

In Section 5.6, we will compare the performance of several different approaches, including a branch-and-bound algorithm, genetic algorithm, and EDF-based heuristic as described above.

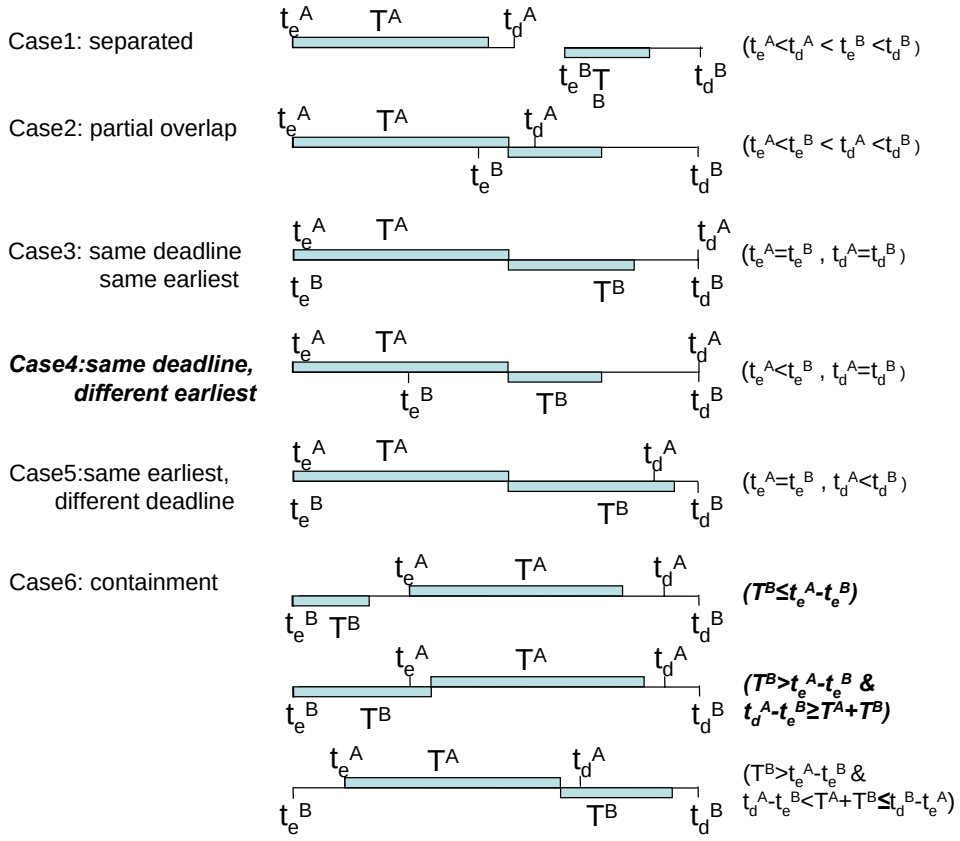

Figure 5.3 Schematic relationship of two jobs with earliest start time and deadline. In this figure, T_{job} is abbreviated to T, $t_{earliest}$ to t_e, and $t_{deadline}$ to t_d.

5.3.3 Lazy Termination

In the above discussions, we assume that each job will be terminated if the actual job execution exceeds the estimated time. An alternative is to attempt to execute the job for a longer period of time than the original schedule. In such cases, job postponement can affect the start time of the next job. As a consequence, the next job's probability of running longer than its scheduled finish time and thus having to be postponed as well will also change. If we consider the postponement as a status of the job represented by X, the postponements of all jobs form a Markov chain [81].

There are simplified cases when termination of the current job is postponed only prior to the scheduled start time of the next job and bounded by its deadline. In such cases, the probability of the next job running longer than its scheduled finish time and thus having to be postponed is not changed, as the postponement of the current job will not impact the existing schedules.

5.3.4 Alternative SLA Offers

Another possible scenario is the existence of alternative QoS parameters in the SLA. For example, suppose several alternative sets of time constraints like earliest start times $t^{(j)}_{deadline}$ and deadlines $t^{(j)}_{deadline}$, each with different business values like price $p^{(j)}$ and penalty (or fine) $f^{(j)}$.

There are several strategies to deal with alternative SLA offers. In one possible strategy, one out of the several alternative QoS parameters is selected for scheduling until it is successful with one of the alternatives. In another strategy, schedules for all alternative QoS parameters are performed, which might result in multiple possible schedules. A comparison and selection can be performed based on some internal criteria like the expected benefit U or unit benefit $U/\hat{t}_{job}$.

5.4 Scheduling Phases

For each job handled by the local resource manager, the scheduling activities can take place several times during and after SLA negotiation. Such scheduling activities can be divided into three different phases with respect to agreement establishment, including pre-scheduling, scheduling, and re-scheduling.

Pre-Scheduling : Pre-scheduling is an optional phase which takes place during SLA negotiation sessions with SLA acknowledgement or asynchronous parameter submission (ref. Section 5.3.1). Take the case of SLA acknowledgement for example: For each agreement offer initiated by the service provider, a temporary schedule is made so that the resource can be reserved before the client acknowledges the agreement offer. The temporary schedule will be canceled if the client rejects the agreement offer or does not respond before a deadline for SLA acknowledgement T_{SLA}.

Scheduling : Scheduling happens when an agreement offer is accepted by the service provider or service consumer. In the later case, the temporary schedule resulted from pre-scheduling can be simply consolidated.

Re-Scheduling : The existing schedule can be changed during re-scheduling. Re-scheduling usually takes place when new jobs are scheduled, but it can also happen on certain critical points, for example:

1. The actual SLA acknowledgement t_{SLA} is far ahead of scheduled SLA acknowledgement deadline T_{SLA} and the resource is free.
2. The actual ready time t_{ready} is far ahead of the scheduled ready deadline T_{ready} and the resource is free.
3. The actual finish time of a job t_{finish} reaches the scheduled finish time T_{finish}, and the scheduler decides to give it additional time $t_{allowance}$.

5.5 Scheduler Design and Implementation

A scheduler based on probabilistic service execution time prediction has been designed and implemented. This scheduler assumes space sharing of resource, i.e., each running job will have exclusive access to the allocated resource and other jobs have to wait until the previous job has finished or exceeds the schedule and when resource becomes available again.

A single queue of pending jobs is maintained by the scheduler. Jobs are dequeued when they become the currently running job. Jobs are sorted according to the scheduled start time. The job is represented by Java class SchedulingEntry, which contains the following information:

- Job parameters
- SLA parameters (earliest start time, deadline, price, penalty etc.)
- Performance prediction results (estimated execution time, estimation confidence etc.)
- Scheduling information (scheduled start time, scheduled finish time)

For evaluation purposes, we have implemented several scheduling algorithms in the scheduler, including Branch-and-Bound (BB) search, heuristics based on EDF as described in Section 5.2.3, and a genetic algorithm (GA). Based on performance comparisons (ref. Section 5.6), we use a hybrid approach for the scheduling, in order to balance between optimality and time taken for scheduling. If the total number of jobs are less than 20, BB search is used, otherwise use the heuristic based on EDF.

5.6 Experimental Results

We did experiments to examine the performance of three different algorithms. The experiments are done with simulation where scheduling requests are randomly generated. The run time of jobs are simulated with a negative exponential distribution, with a normal distribution of prediction error. The earliest start time is generated with a uniform distribution, and the deadline is simulated with a negative exponential distribution after the earliest start time plus job run time.

We performed experiments with different amount of jobs, different algorithms, and different confidence intervals. We examined the number of jobs that are accepted, and the jobs that are successfully executed before the deadline.

Figure 5.4 shows the run time of scheduling for different algorithms with a problem size up to 25. While the run time with heuristics based on EDF can be neglectable for such a small problem size, that of Branch-and-Bound search grows so fast that on-line scheduling is not possible with a problem size over 20. Genetic algorithm begins with a very small run time that rises as the number of jobs approaches 10 and stays at a run time of around 25,000 ms, which corresponds to 100 iterations after which if no improvements are found the iteration will be stopped.

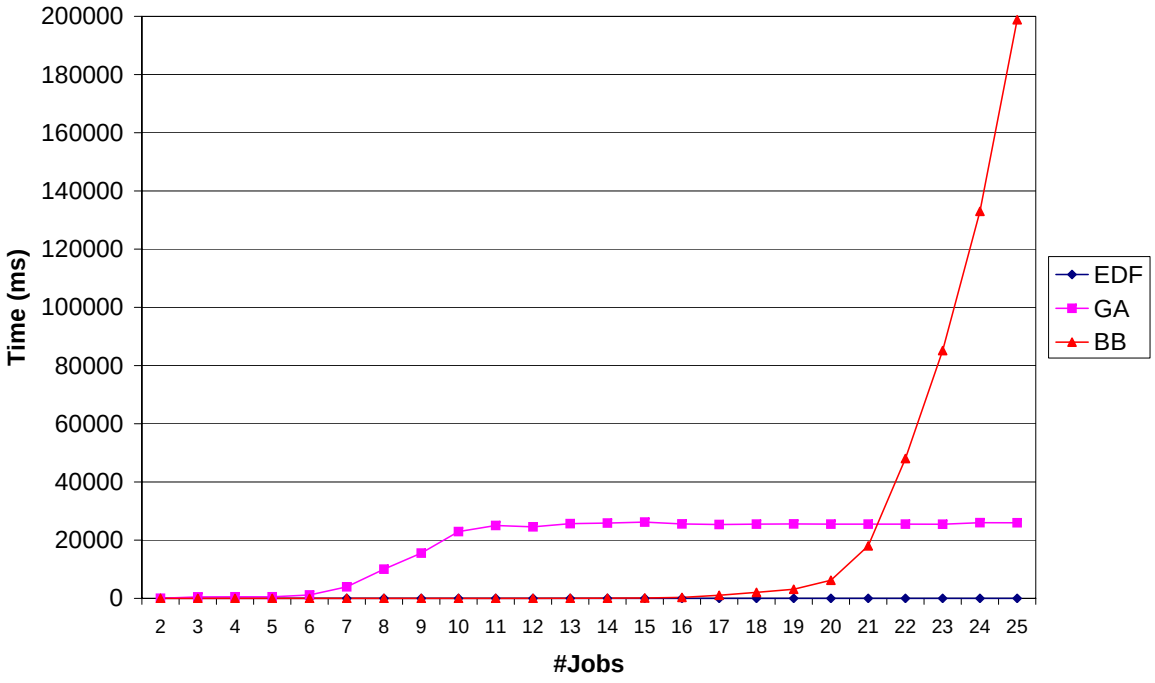

Figure 5.4 Run time of scheduling for different algorithms with small problem size.

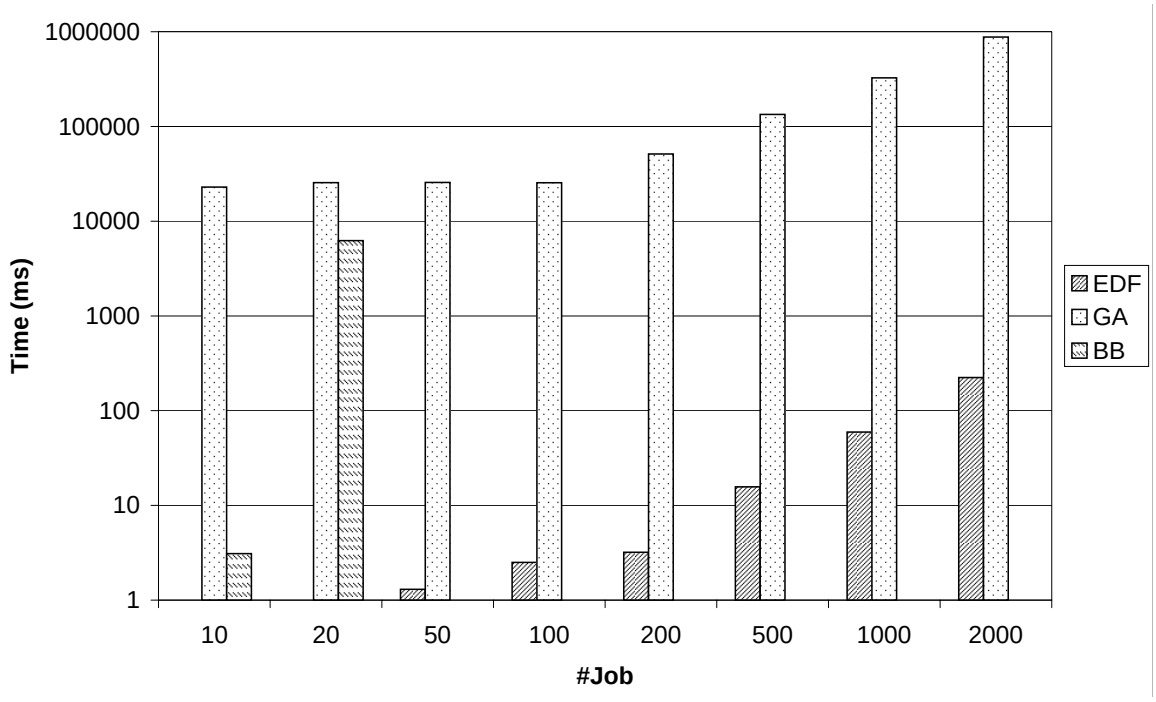

Figure 5.5 Run time of scheduling for different algorithms with large problem size.

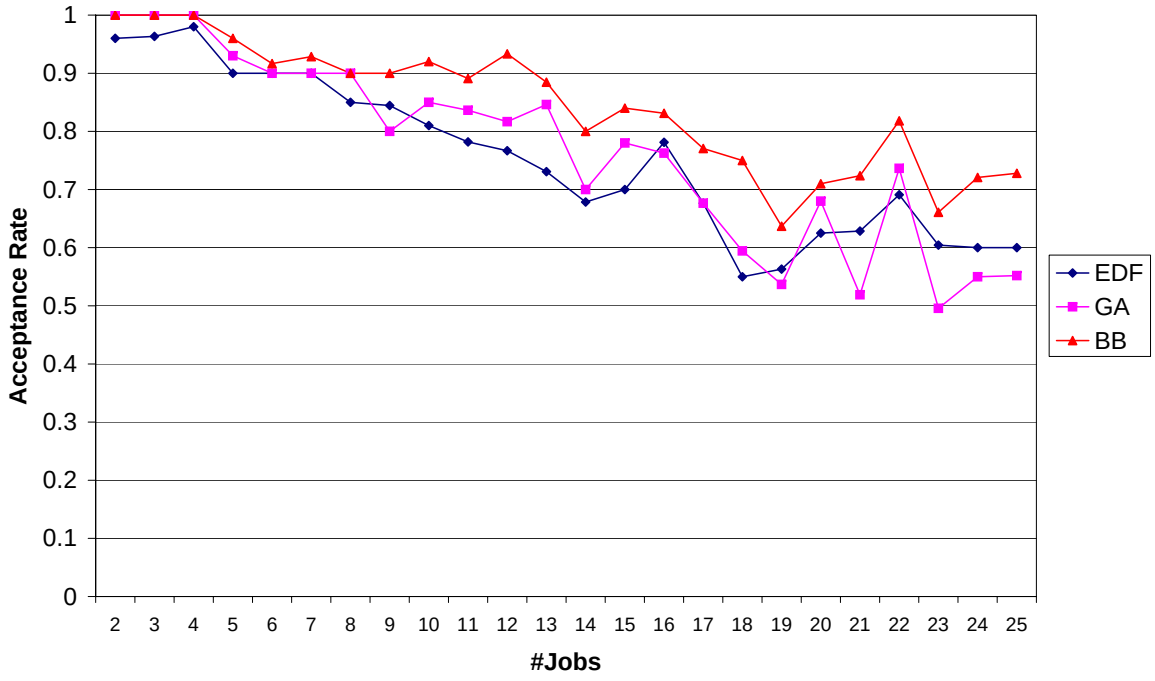

Figure 5.6 Acceptance rate for different algorithms with small problem size.

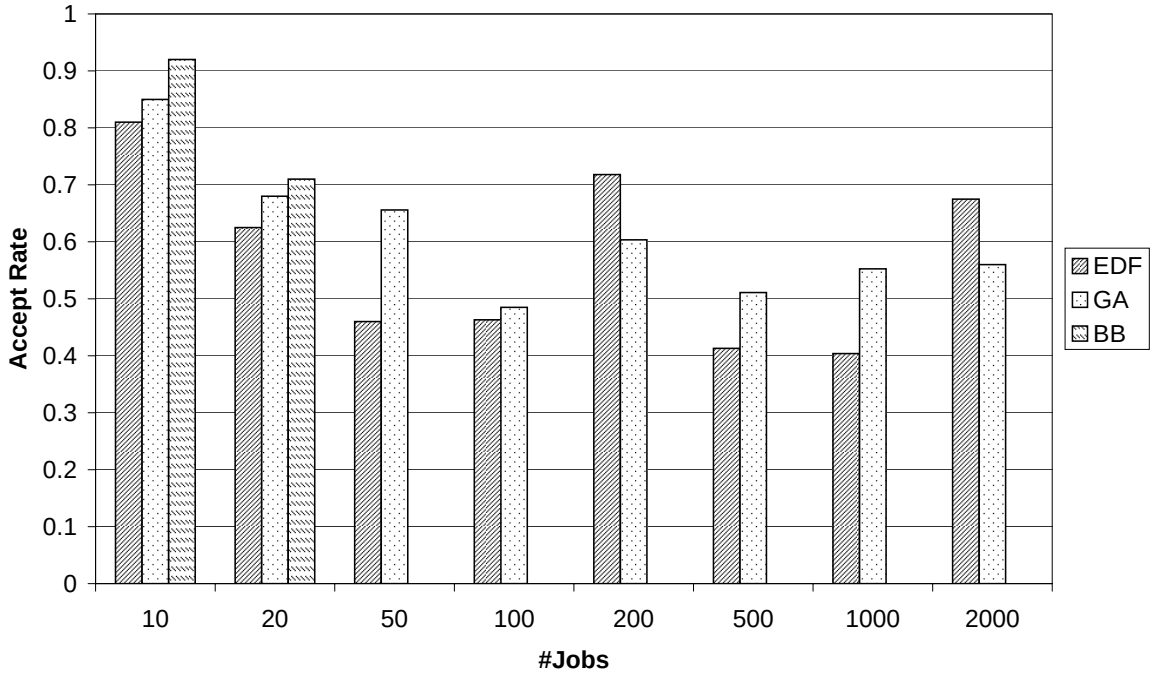

Figure 5.7 Acceptance rate for different algorithms with large problem size.

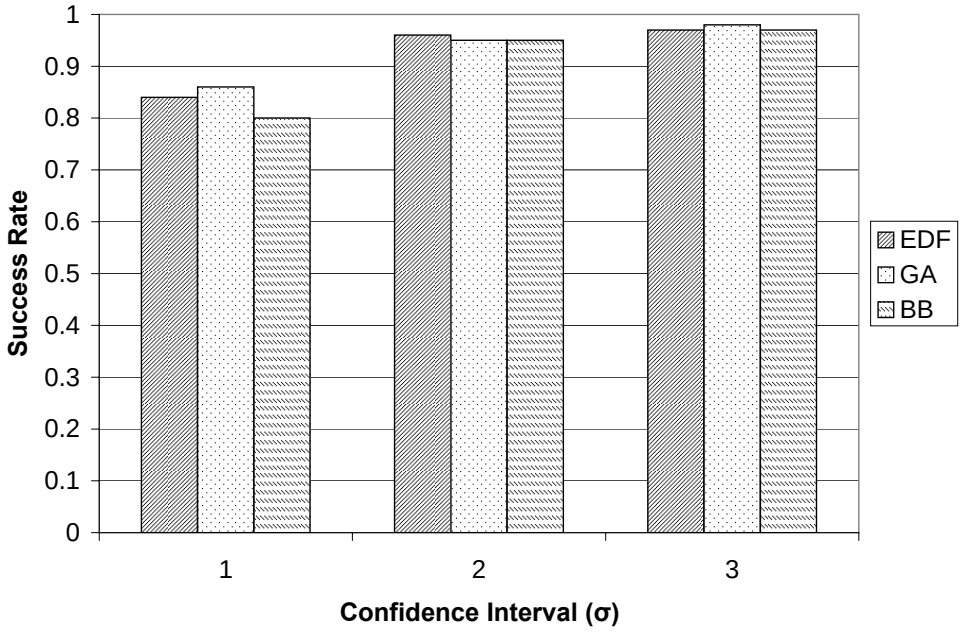

Figure 5.8 Effects of prediction error on the rate of successful execution.

Figure 5.5 shows the same run time with large problem sizes. With the problem size growing from 10 to 2,000, the run time for heuristics based on EDF grows from less than 1 ms up to about 200 ms. The run time of genetic algorithm also rises from 25,000 ms to 900,000 ms. The run time of Branch-and-Bound search was not measured for problem sizes greater than 25 because of its poor scalability.

The quality of different algorithms can be evaluated by comparing the job acceptance rate. The result of Branch-and-Bound search provides optimal results that serve as the maximum of jobs that can be scheduled. For small amount of jobs in Figure 5.6, genetic algorithm and EDF-based heuristic shows comparable performance, with a drop of 5% to 15% average job acceptance rate compared to that of Branch-and-Bound search. A similar conclusion holds for large amount of jobs (see Figure 5.7), except that no comparison was made against the Branch-and-Bound search for jobs over 20 because of its huge run time.

Figure 5.8 shows the impact of different confidence levels on the success rate, i.e. the ratio of jobs that are successfully executed within all jobs that are accepted. The results for all three algorithms are quite similar. While the success rate improves a lot when the confidence level increasing from 1σ to 2σ, it does not improve very much when the confidence level further increases to 3σ.

5.7 Related Work

5.7.1 Scheduling Algorithms

Moore [88, 87] presents a genetic algorithm to schedule tasks on clusters, targeting make-span minimization. Braun *et. al.* [26] [25] compared eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. There are no deadlines for the tasks, and the objective of the scheduling is to minimize make-span. Some of the heuristics are specific to mapping tasks to multiple resources, such as Opportunistic Load Balancing, Minimum Execution Time, Minimum Completion Time, Min-Min, Max-Min, and Duplex. Others are general ones that can be applied to our scenario as well, including Genetic Algorithms, Simulated Annealing, Genetic Simulated Annealing, Tabu Search, and A* (a Branch-and-Bound tree search algorithm). According to the comparison result, GA is out performs among all the algorithms and we assume this also holds true for our case. Basically, we can also extend our research to cover all different algorithms.

Spooner *et. al.* [111] studied the local scheduling on a cluster of hosts, where each job has an associated deadline and can be mapped onto multiple hosts. It also employs a performance prediction to evaluate the performance of applications under different system configurations. It evaluates and compares three different heuristics, including deadline sort (i.e. EDF) and genetic algorithm that uses a fitness function that balances make-span, processor idle time, and deadline fulfillment. This work supplements ours as we are focusing on the provisioning of services that have exclusive access to the local resources. Other work on the application of genetic algorithms on job scheduling targeting make-span minimization is also available in [51], [93], and [7].

5.7.2 Performance Prediction Assisted Scheduling

In [125], scheduling of jobs on a cluster of hosts using predicted run time and variance has been presented. Similar to our approach, it uses the mean plus variance as the run time for scheduling. [107] applied predicted run time derived from workload archive to help estimate queue wait times and improve scheduler performance. Both researches are limited to independent jobs without any constraints on the execution time.

An analytical study of the impact of prediction inaccuracy on execution scheduling is discussed in [65]. It introduces the degree of misprediction, a performance metric that represents the probability that the predicted execution times of jobs display different ordering characteristics from their real execution times due to inaccurate prediction. The impact of prediction inaccuracy on make-span of several Grid scheduling algorithms, including Min-Min, Max-Min, Sufferage, and Fast-Greedy, is simulated in [126].

5.8 Conclusion

Local resource scheduling is a major component of the local resource management infrastructure. This chapter discusses major issues that arise in the scheduling of service invocations in SLA-based resource management environments, where normal schedulers

can not fulfill their specific requirements. Based on comparison of different algorithms, a scheduler implementation is also introduced.

While the support infrastructure of WS-Agreement presented in Chapter 3 allows the specification and management of SLAs, the actual decision on SLAs has to be made with the admission control functionality of the local scheduler, which examines the availability of resources against the request. The run time prediction facility presented in Chapter 4 offers run time estimations for a specific request, and is the basis of job scheduling.

The local scheduler only solves the problem of job scheduling of individual resource providers, and the issue of mapping jobs to appropriate resource providers is a collective effort of the SLA management infrastructure. In Chapter 6, the local resource management is supplemented with a global infrastructure that assists the SLA negotiation and enforcement among multiple clients and unknown resources.

Chapter 6

Setting Up a Global Infrastructure

6.1 Overview

The local management infrastructure provides fundamental support for SLA-based resource management. Based on the local infrastructure, a client application can negotiate with a known server for formal contracts about service usage on an individual basis. In a realistic Grid system, where clients seek for available servers and dynamically select the appropriate server that can meet their requirement, a global infrastructure is required to assist such SLA negotiation processes.

In the spirit of service-oriented Grid, the basic building blocks of the global infrastructure are services of various types. Due to the open nature of Grid systems, the type of services that constitute the global infrastructure is also not closed. As part of the ARM4SLA project, a set of services have been implemented or integrated, that collectively form a minimal global infrastructure for SLA-based resource management. The services include the index service that provides a catalogue of available services together with their SLA templates, and the broker service that helps the client to negotiate with multiple services. In addition, a gateway service enables status and asynchronous execution of traditional stateless Web services, so that they can be managed by the local SLA management infrastructure. Those services provide support for three major steps for running a Grid job [22] - resource discovery, resource selection and job execution.

The rest of this chapter is organized as follows. Section 6.2 briefly introduces the roles of index services, broker services and gateway services in the establishment of a global infrastructure. Section 6.3 presents the index service with a focus on the registration of SLA templates associated with Grid services. Section 6.4 discusses the functionality, design and implementation of the gateway service, and Section 6.5 the broker service. Possibilities of more advanced infrastructure services beyond those basic ones, which address scalability issues with market-based mechanisms, are introduced in Section 6.6. This chapter ends with a summary in Section 6.7.

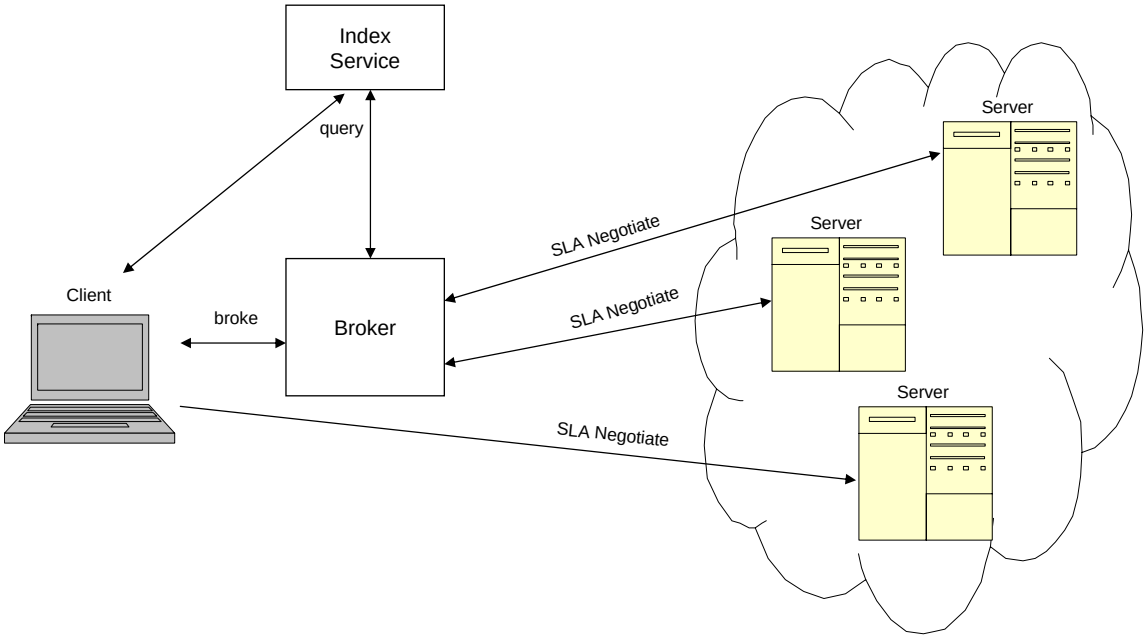

Figure 6.1 A minimal global infrastructure supporting SLA-based resource management.

6.2 The Global Infrastructure

6.2.1 Establish SLA with Index and Broker Services

A minimal global infrastructure can be formed with an index service and a broker service, as is shown in Figure 6.1. The index service provides a list of candidate services to the broker or the client. The broker service helps the client to establish an agreement with one of the candidate services that can fulfill the QoS requirement.

The index service maintains a list of resources that are registered to it. Attached to each resource, it also maintains a list of selected resource properties. In order to assist agreement establishment, the index service should provide information about available application services and associated agreement factory services. Optionally, agreement templates can also be registered. The registration has to be performed with custom implementation and configuration script.

Upon the request of a client, a broker service tries to establish an agreement for service usage with specified QoS requirements. The broker service relies on the index service for a list of available application services and associated agreement factory services, possibly also the templates of each agreement factory. Based on this information, the broker service makes a pre-selection of the candidates and tries to establish service usage agreements with services in the candidate list. The exact sequence of negotiation depends on internal evaluation of the broker service. The result of agreement negotiation will be notified to the client, with a reference to the established agreement when successful.

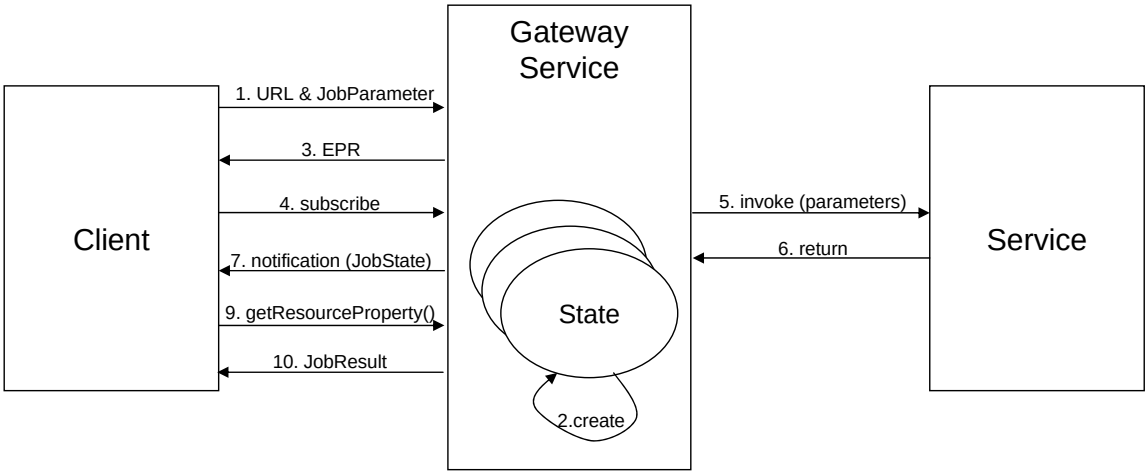

Figure 6.2 Gateway service adds status to stateless services.

6.2.2 Enforce SLA with Gateway Services

Many existing Web services are stateless services. Every invocation of the service results into a prompt execution of the service, with the underlying resource consumed. For such services, the effectiveness of negotiated SLAs on resource management is limited - if there is an agreement negotiated, the service will be executed, otherwise the service invocation will be rejected. In order for the SLA-based resource management approach to work, in which the execution of services is handled by the local resource manager, we need to enable asynchronous execution of the services. Key to the enablement of asynchronous service execution is the representation of service lifecycle and results with status. For services that are custom implemented, adding status to the service is usually an issue of additional coding. However, for many existing stateless services, it is impractical and sometimes impossible to modify their implementation. The gateway service is designed to address this issue. It enables status and asynchronous execution of existing services without modifying their implementation.

The gateway service serves as a generic facade (ref. Figure 6.2) to the stateless Web services in the backend. Instead of directly invoking the backend service, the client sends the URL and parameters of the target service to the gateway service. The gateway service initializes state for the service invocation and returns an EPR to the client. With the EPR, the client can query details of the current status at any time. Optionally, the client can subscribe to notifications to be notified with any changes of the state. The actual invocation of the backend service is thus an asynchronous process, which is controlled by the local resource manager. The state reflects the status of the invocation lifecycle as well as parameters and result.

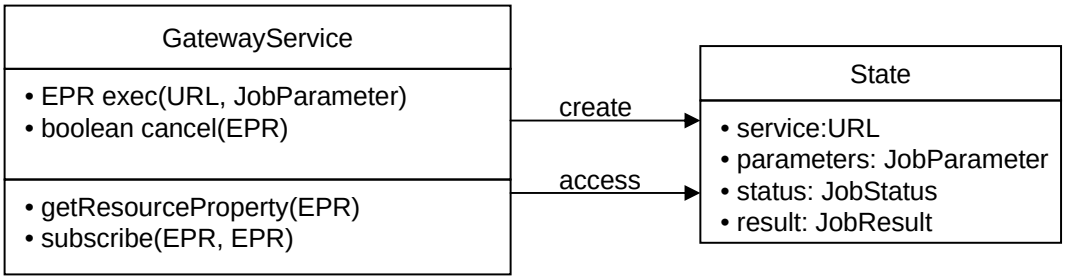

Figure 6.3 Gateway service and associated job resource.

6.3 Index Service

Index service is part of WS MDS (Web Service Monitoring and Discovery System) module of GT4. It collects information about Grid resources and publishes that information as a service group [5]. Index service information can be queried using standard WS-ResourceProperties operations or retrieved through subscription/notification operations specified in WS-Notification.

The index service relies on the WS MDS Aggregator Framework [103] to collect information from aggregator sources. An aggregator source is a Java class that implements an interface (defined as part of the aggregator framework) to collect XML-formatted data. For service resources, a query source uses the WS-ResourceProperty mechanism to poll a WSRF service for resource property information.

In order to publish agreement templates in the index service, we rely on the GT4 API to publish resource property as information. The registration is done in the agreement template resource home class, which is responsible for the actual creation and initialization of resources. The most important component of the API is the ServiceGroupRegistrationClient class. It allows the user to specify registration parameters and initialize a timer for periodical registration of resource parameters. The parameters can be retrieved from a configuration file in predefined format. The configuration file specifies refresh interval, poll interval, and a list of resource property names which in our case is the wsag:Template.

6.4 Gateway Service

The gateway service provides a stateful facade for backend stateless Web services. It is designed and developed following the WS-ResourceProperties and WS-Notification specifications (ref. Figure 6.3). The State resource represents the state of a service invocation. It is created and managed through interfaces offered by the GatewayService, including custom ones for launching and canceling a service invocation and operations inherited from WS-ResourceProperties and WS-Notification port types for accessing the resource properties and subscribing and notifying property changes. The properties of the State resource include URL of the target service, job parameters, job status (pending, running, finished, error), and job result.

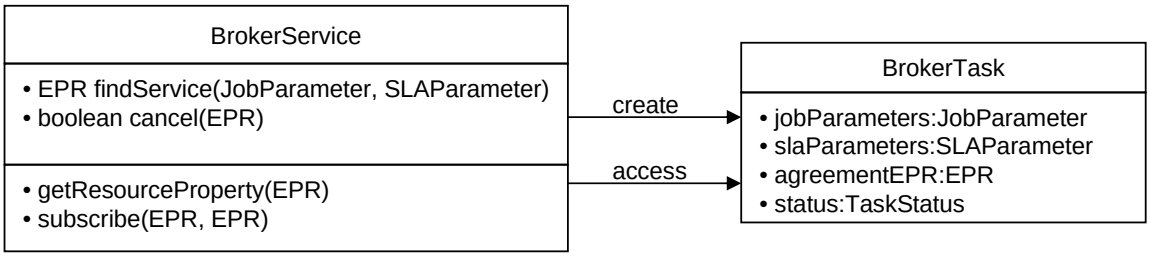

Figure 6.4 Broker service and associated resource.

Gateway service creates the State resource when its exec() operation is invoked. It is initialized with supplied service URL and job parameters, and the initial status of the job is "pending". The follow-on modifications to the job status and result are performed internally by the local resource manager which takes care of the actual execution of the backend service. The changes of status and job result can be promptly notified to the client or accessed at any time by the client.

6.5 Broker Service

6.5.1 Broker Service and Resources

Creating an agreement with multiple candidate services is a long running process, therefore the broker service has been designed and implemented as a stateful service (ref. Figure 6.4). The broker service inherits the operations defined by the port types specified by the WS-ResourceProperties and the WS-Notification specifications for accessing the resource properties and subscribing and notifying property changes. The findService() operation is the starting point of a brokering process, with which the client specifies the parameters of the job and its SLA and get a reference to the created broker task. It also provides a method for canceling the brokering task.

The BrokerTask resource represents the state of a broker task. It is created and initialized when findService() operation of the broker service is invoked. The job parameters and SLA parameters are kept for later retrieval, and the status is initialized to idle. Later updates to the status are performed by an internal worker thread (ref. Section 6.5.2) that performs the actual brokering. If brokering is successful, the resource property will also be updated with the resulting agreement EPR. Any updates of the resource property can be promptly notified to the client or accessed at any time by the client.

6.5.2 Asynchronous Broker Thread

When the user invokes the service with findService() operation, a reference to the created broker task will be promptly returned. The actual work is performed in an asynchronous way, which includes the following activities:

1. Query index service for all candidates.

2. Pre-selection according to specific QoS requirements (minimal price, for example).

3. Prioritize the remaining candidates.

4. Negotiate with each candidate until succeed.

Those activities are performed in a thread managed by the Web service container. It is implemented with the help of Work Manager framework supported by GT4 WS Core. The key components of the Work Manager framework are WorkManager and Work. WorkManager is responsible for the management of a series of Works. The Work is where the actual activities are implemented. The following code snippet is an example of retrieving WorkManager from the Web service context and the registration of a new Work:

```
Work work = ...
Context ctx = new InitialContext();
WorkManager workManager =  (WorkManager)
  ctx.lookup("java:comp/env/wm/ContainerWorkManager");
workManager.schedule(work);
```

6.5.3 Brokering Strategy

For each job to be brokered, there can be multiple candidate services. The broker negotiates SLAs with those candidates following a specific sequence. Appropriate choice of the sequence will reduce the amount of messages between the broker and candidate services. In addition to a static brokering strategy that follows a predefined sequence of the candidate services for all the jobs, we have also considered the following strategies that the broker takes to determine the negotiation sequence: random, round-robin, and information-based prioritization.

With random brokering strategy, candidate resources are chosen on a random basis for each job, i.e. a random choice of the resource is performed and the chosen resource is the next to negotiate with. This is equivalent to prioritizing the servers randomly for each job. Caution is taken to make sure that visited resources will not be revisited for the same job.

With round-robin brokering strategy, candidate resources are negotiated one by one on a round-robin basis. Additionally, it also makes sure that the same resource will not be revisited for the same job.

Traditional Grid brokers utilize utility information published by each service to prioritize appropriate resources in order to improve the schedule. In a Grid system with SLA-based resource management, no global knowledge about the resource utility is published. Therefore, the utility can only be roughly estimated by the broker. The broker maintains local records of previously brokered jobs, and uses the success rate of previous job brokerings as a rough indication of the utility.

We have performed experiments to compare the effectiveness of different brokering strategies. The experiment is done with discrete event simulation that simulates the behaviors and interactions of Grid clients, broker service, and Grid servers. The simulation

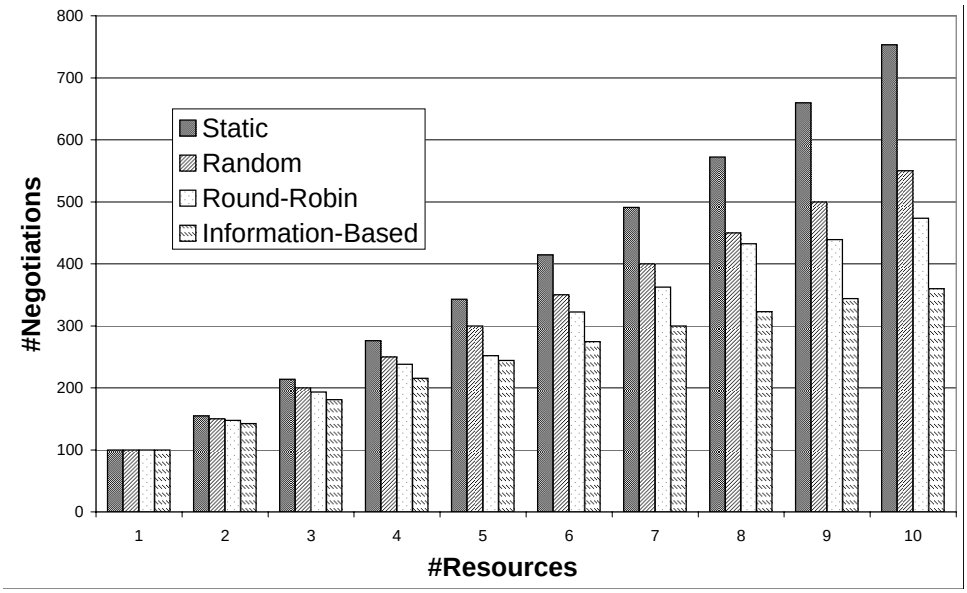

Figure 6.5 Number of negotiations for different brokering strategies.

program is implemented on top of SimJava [62] [1], a discrete event simulation library for Java. The Grid server is simulated as a time-sharing computational resource, with all the jobs scheduled using the EDF algorithm introduced in Chapter 5. The broker service is simulated with different brokering strategies, and the Grid clients submit random jobs.

Figure 6.5 shows the number of negotiations between the broker and the candidate servers for 100 jobs. With the increase of the number of resources from 1 to 10, the number of negotiations between the broker and candidate servers increases for all brokering strategies. According to the speed of such increases, the four brokering strategies can be sorted from static, random, round-robin to information-based.

A static brokering strategy performs worse than random strategy because it follows a fixed sequence of negotiation and thus has higher chance to fall into the situation where the following job is rejected by the same server of the last negotiated job due to conflicting schedule. For the same reason, since round-robin strategy avoids this situation completely, it performs even better than the random strategy. Compared with the above strategies, information-based strategy performs the best, because it is capable of capturing the utility information of the servers.

The observations from Figure 6.5 are two-folds. Despite the capability of information-based brokering strategy in reducing the number of negotiations, the scalability for Grids with larger number of resources still constitutes a major issue of agreement negotiation with simple brokers. This can be solved by introducing more advanced infrastructure services based on market mechanisms, which will be introduced in Section 6.6.

[1] http://www.icsa.inf.ed.ac.uk/research/groups/hase/simjava/

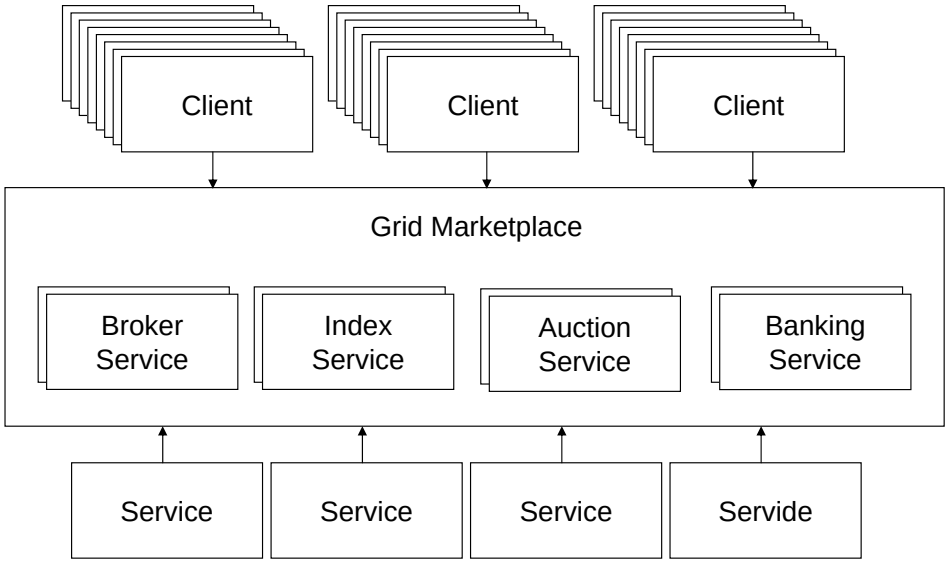

Figure 6.6 Infrastructure services in a Grid marketplace mediating agreement negotiation.

6.6 Beyond the Basics

The global infrastructure introduced in former sections that are constituted by several basic services possesses scalability issue. For Grid systems with a large number of resources, agreement negotiation using simple broker services can result into too many messages between the broker and the candidate resources. This can be solved by introducing infrastructure services that assists agreement negotiation with more advanced mechanisms. A major type of such services is originated from ideas based-on market mechanisms - auction service, for example, accepts messages from both clients and services and establishes agreements with an auctioning process. It can be combined with other techniques like mobile agents (ref. [49]) to reduce the amount of messages for agreement negotiation.

Services based on market mechanisms usually demand supports on financial issues that deal with currencies. Grid currencies can either represent and can be transformed into or from real money, or are simply virtual tokens. This again can be assisted by additional services. For example, there can be banking services that provides the support of currency exchange. All such services collectively form a Grid marketplace (ref. Figure 6.6), where the Grid clients and services can trade with each other.

6.7 Conclusion

The local management infrastructure presented in Chapter 3 to Chapter 5 provides fundamental support for SLA-based resource management. This chapter supplements the local management infrastructure with a global infrastructure that provides services to assist the establishment of agreements between clients and services.

In Chapter 8, the global infrastructure is applied to a demonstration based on a concrete

application scenario of distributed data mining for banking.

The global infrastructure presented contains a minimal set of services that are needed. More advanced ones can be implemented to extend the infrastructure, for example those supporting market-based service trading. This and other relevant issues constitute part of future work that is summarized in the last chapter.

Chapter 7

Development Tools and Support Environment

7.1 Introduction

Applications managed by the SLA-based resource management infrastructure are formed by Grid services. Such services include not only the standard ones, but also include those that provide custom functionalities. The development of custom services and their integration with the infrastructure demand development tools. In addition, the development of client applications and the management of the Grid application demand support environments.

A set of development tools and support environments are developed, which will be introduced in this chapter. Section 7.2 discusses the requirement of different user roles and explained the necessity of using component-based runtime platform like Eclipse. Section 7.3 introduces the tooling environment for Grid service development, Section 7.4 introduces the base client environment for Grid services, and Section 7.5 the environment for managing Grid infrastructure. Section 7.6 presents related work on Grid service development tools and Grid client and management environments. Section 7.7 concludes this chapter with a brief summary.

7.2 Application Development using Weaveable Components

7.2.1 User Roles and Functional Requirements

The tools and environments (see Table 7.2.1) are designed to meet the functional requirements of different user roles. These include service development tools that help service developers to develop Grid applications, execution clients that are used by Grid users to control the execution of jobs, and management environments that are used by Grid managers to administrate the Grid system.

Many functional requirements of different user roles are overlapping, therefore the tools and environments need to be developed in a component-based approach that enables the reuse of components between different applications. The actual types of services deployed in the service-oriented Grid system, which are management by the SLA-based resource

management infrastructure, are dynamically changing. Besides, many users need additional tools. Therefore, the management tools and environments should be extensible to work with different types of services and be able to integrate additional tools.

	Development Tool	Client Environment	Management Environment
Platform	Eclipse IDE	Eclipse RCP/IDE	Eclipse RCP
Purpose	program services	execute job	manage system
User role	Grid developer	Grid user	Grid manager

Table 7.1 Development tools and support environments.

The above mentioned user roles are neither distinct nor mutually exclusive. Rather, there are chances that a specific person has multiple roles and needs to have corresponding tools and environments to support his work. For the user's convenience, the tools and environments for different roles should be able to be integrated consistently and seamlessly.

Another noticeable fact is the overlapping of functions in the above mentioned tools and environments. For example, both the development tool and the management environment need to deploy Grid services; both the client environment and the management environment need to explore the index services; and all of them require the functionality of executing Grid services. Therefore, component reuse should be enabled.

The management client has both a domain specific part (that relies on the concepts, terms and program details of the specific service, for example the result visualization) and a domain independent part (that is generic to any usage). The client needs to provide a capability of extension so that the client can be easily adapted to a different scenario.

7.2.2 Eclipse as Component Platform

In response to the above requirements, we need a platform that supports component-based program deployment enabling extensions, and supports both IDE and normal client applications (rich client in terms of Eclipse). The best known platform that meets such criteria is Eclipse [1] which provides plug-in based application development, role-based UI layout into perspectives, fundamental platforms for IDEs and rich client applications with its Rich Client Platform (RCP). Certain components of the Eclipse platform can be directly integrated into our environment, for example, the CVS support etc.

Following Eclipse, there are also other similar platforms emerging, for example NetBeans [2]. However, regarding the popularity and functionality (esp. Eclipse provides native OS look and feel), Eclipse is currently the preferred platform chosen as our implementation basis.

[1] http://www.eclipse.org
[2] http://www.netbeans.org

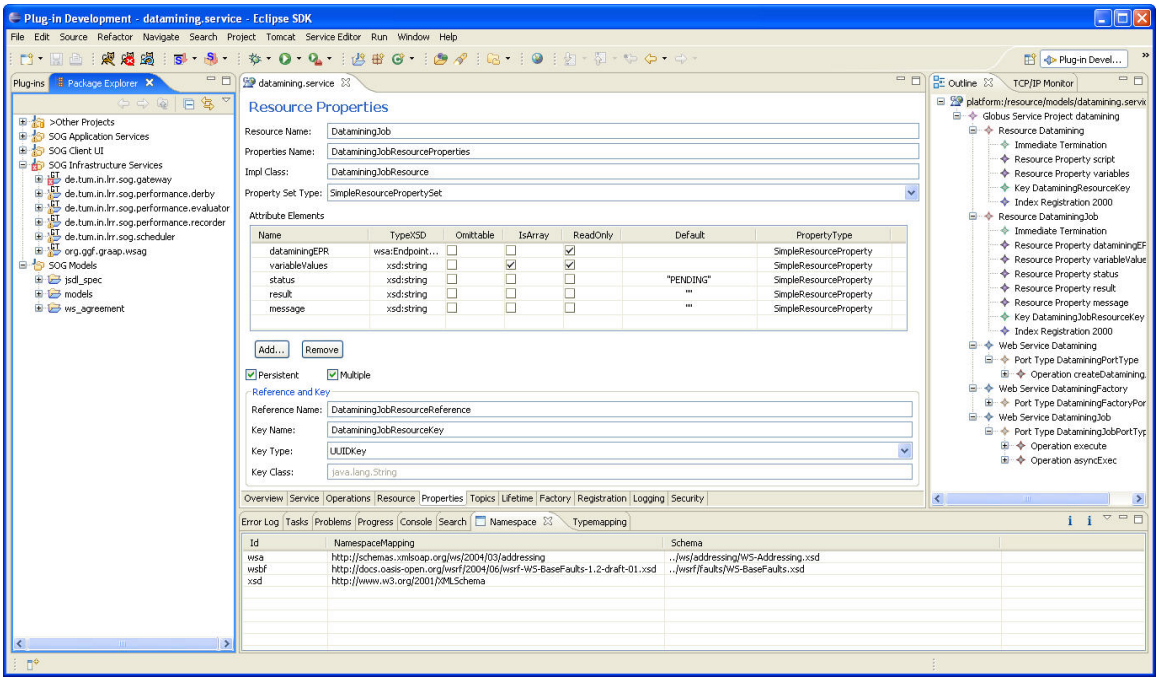

Figure 7.1 The Globus Service Development Environment (this screenshot shows the user interface for service modeling).

7.3 Grid Service Development Environment

7.3.1 The Demand

Programming services for the Grid is a tedious and error-prone process, due to both the large amount of work for correctly writing the service interface document and configuration files, and the effort in writing a group of classes following predefined patterns (service, resource, resource home etc.). The Globus Toolkit provides only very basic command-line tools for stub-generation and service deployment. All the service definition and configuration files have to be prepared by the user, which is quite challenging for most of the developers. Although existing Web service development tools and environments provides support for editing service interface definition, functionalities specific for Grid services development (e.g. the configuration files for resource registry, security etc) does not exist. More importantly, programming with such tools means that the user has to go down into the details of WSDL, service deployment scripts etc, which are too complex. We need a tool that allows the programmer to program at a higher level and help the user to do all (or most of) the tedious works. This is the objective of the Globus Service Development Environment (GSDE).

7.3.2 Functionalities

GSDE is intended to be such a programming tool that supports the development of Grid services in a natural way that is familiar to most Java developers. It does not require the programmer to follow a specific development process, and it does not restrict the developed Grid services to follow specific patterns. Towards this aim, major functions of GSDE are divided into three groups that can be used separately or collaboratively:

1. Service modeling and code generation. It allows users to develop models as templates for Grid services, which can be customized and used to generate skeletons of Grid service codes.

2. Service development. This includes project creation, configuration and building support, as well as deployment, execution and debugging of the developed service.

3. Service deployment. It allows the project to be exported into GARs or deployed into different runtime servers, currently including Globus and Tomcat.

Modeling Grid Services

GSDE supports the modeling of Grid services with user-friendly editors. The following aspects can be modeled:

1. Service-Resource relationship. This includes the association and creation relationships between services and resources. This is a generalization of some typical patterns like factory service and singleton service. However, it allows more complex relationships to be modeled.

2. Service model. This includes parameters such as name, operations etc.

3. Resource model. This includes the specification of resource parameters such as name, properties, persistence etc., resource lifetime (immediate destruction or scheduled termination), and resource notification topics including topics representing resource property change and custom topics.

4. Configuration model. This includes aspects like index registration, logging requirements, and security configurations.

To meet the needs of two different types of users - experts and newbies, two editors are provided: Advanced Editor and Simple Editor. The Advanced Editor provides a tree-based interface that can be applied to model complex relationships (many to many) between services and resources. The Simple Editor offers a more user friendly interface based on forms (see Figure 7.1), but is currently restricted to model simple service-resource relationship following the singleton or factory design pattern. The two editors can also be used in a combined way - use the Simple Editor to edit the properties of services and resources, and use the Advanced Editor to compose their relationships.

Grid Service Skeleton Code Generation

Models of Grid services can be used to generate classes and scripts. The generated classes and scripts serve as a skeleton of the actual Grid service implementation, which can be modified or customized in different aspects.

The classes include service implementation, resource implementation (including resource interface, resource, persistent resource, resource home etc.), client implementation (base classes for client, notification listener etc), and assistant classes (constant definition, etc).

The scripts include WSDL, deployment descriptors (service and client), namespace2package.mapping and jndi-config-deploy files, post deployment scripts, security descriptors (service, resource, client), and registration configuration files.

Grid Service Project Management

A wizard is provided that helps to create and configure projects for Grid service development. It guides the users through a series of steps for project creation and configuration, which is otherwise a complex and dispersed process:

1. A Java project is created with a custom tag, or "nature" in terms of Eclipse. This allows the project to be identified and distinguished.

2. The existence of Sysdeo Tomcat Launcher plug-in [3] is automatically detected, and the project is configured to work with Tomcat if the plug-in exists.

3. The classpath of the project is initialized with Globus jars added. Three different source folders each for the services, the stubs and the client classes are created. Other folders for schema, libraries, etc. are also created. The standard schema files from Globus WS-Core are also copied into project.

4. The project is configured with a series of builders to enable automatic or manual build of the project, from the standard Java builder to the custom GSDE builders.

Automatic Grid Service Build

During the project creation process, several builders are configured to work collaboratively for Grid service project build. A group of custom GSDE builders work together to translate the compact WSDL into stubs. The resulting codes for stubs, together with service codes generated from the service model, are compiled into Java classes by the standard Java builder offered by Eclipse JDT.

All builders are managed by the Eclipse platform and can be configured to work in different ways for automatic or manual project build. With automatic build, modifications of codes and scripts will automatically result in an update of the compiled Java classes.

[3] http://www.eclipsetotale.com/tomcatPlugin.html

Service Deployment, Execution and Debugging

GSDE supports the deployment of Grid services into a standard Globus server or Tomcat server on the local machine. It can also export Grid service projects into GARs, which can be used for remote deployment. Additional deployment plug-ins can be developed to support other servers.

With functionality contributed by third-party Sysdeo Tomcat Launcher plug-in, Tomcat servers can be started, stopped, or restarted directly from GSDE.

Based on advanced features of Eclipse debugging framework, GSDE can debug the Grid service and client directly from inside Eclipse.

Leveraging Eclipse Functionalities

Eclipse provides rich functionalities that can be leveraged in GSDE. In the above descriptions, we have already seen how GSDE uses JDT of Eclipse to assist Java coding, Eclipse debugging framework for debugging Grid services and clients. Many other features of Eclipse can be utilized as well, for example the TCP/IP Monitor can be used to assist debugging, and CVS support can be used for source code management. Web Tools Platform (WTP) provides various editors for Web service related documents and can be seamlessly used for artifacts created in the GSDE.

7.3.3 Integrating Agreement Management

GSDE is designed to support the development of arbitrary Grid services that can be deployed on any Globus servers, without dependence on a specific resource management mechanism. To integrate the developed services with a specific resource management infrastructure, additional codes and scripts must be developed and deployed together with the services.

GSDE can be customized to provide additional assistance to develop services that are to be managed by our SLA-based resource management infrastructure. This is done by extending the service model and code generation templates to generate additional codes and configuration scripts, including custom agreement providers as is described in Section 3.3.4, custom data recorder and predictor as is described in Section 4.5.3, and proactive handler registration service described in 3.4.4 and later in Section 4.6.4.

While the generated agreement providers provide only a skeleton for the custom agreement provider, the generated data recorder and predictors, service launcher, and proactive handler registration services can be normally applied without much modifications if they follow the most common patterns.

7.3.4 Components and Dependencies

GSDE is a set of plug-ins that can be organized into three functional groups (”features” in terms of Eclipse), each for modeling and code generation, project configuration and building, and deployment.

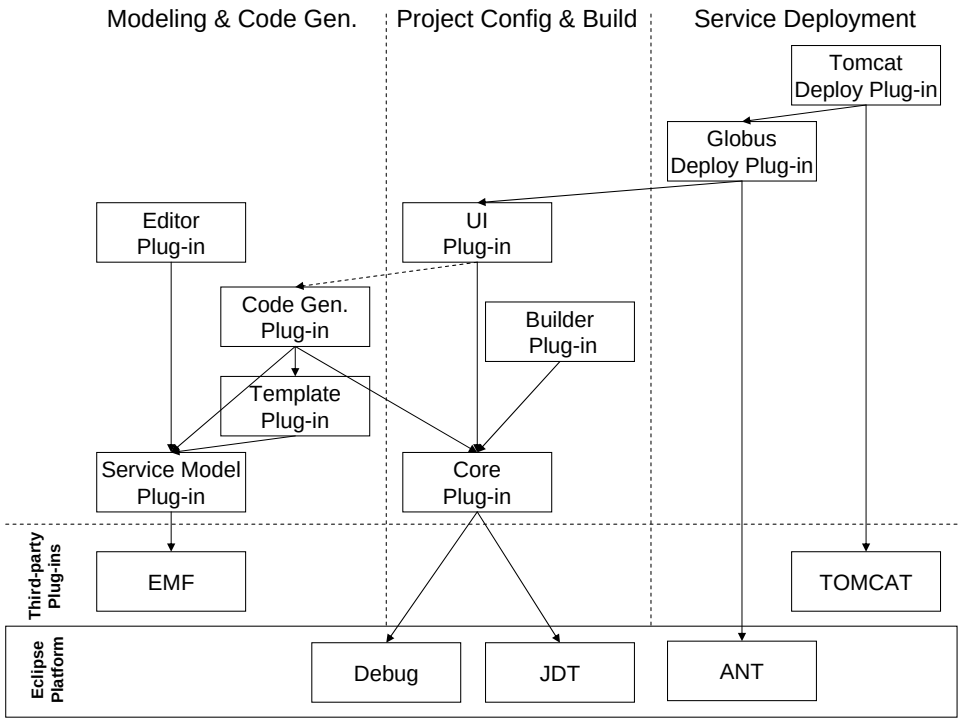

Figure 7.2 GSDE architecture and dependencies among plug-ins.

Figure 7.2 shows the dependencies among the GSDE plug-ins, third-party plug-ins and Eclipse platform. The dashed lines denote optional dependencies. GSDE plug-ins has been designed and implemented to be composable so that different runtime configurations are possible.

Among the dependencies, Ant, JDT and Debugging are part of the standard Eclipse distribution. The availability of Eclipse Modeling Framework (EMF) [28] and Tomcat Launcher plug-ins will decide whether a specific feature is available to GSDE - if EMF is not installed on the platform, the modeling and code generation feature is not activated; and if Tomcat Launcher plug-in does not present, the feature for deploying the project into tomcat is deactivated. However, the basic core function for project creation, configuration and build will be working on most platforms as it does not depend on plug-ins not existing in the standard Eclipse platform.

7.3.5 Service Modeling and Code Generation

The modeling and code generation feature of GSDE follows the model-driven development practice. We use EMF to define the meta-model for Grid services, from which we automatically generate the plug-ins that contain implementations of the meta-model and model editor. Using the generated meta-model and model editor, we can create and edit the model for Grid services.

As part of EMF, Java Emitting Templates (JET) provides a powerful language for defin-

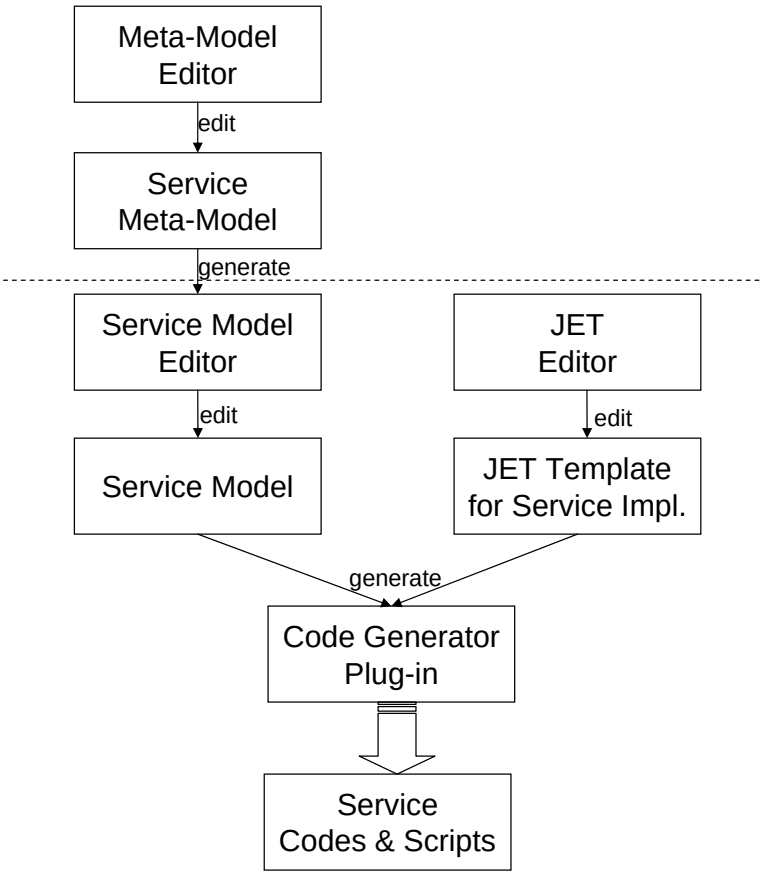

Figure 7.3 Service modeling and code generation in GSDE.

ing templates with JSP-like syntax. The editing of the templates is assisted with JET editor, from the EMFT (Eclipse Modeling Framework Technology) subproject of Eclipse. Compared to hard coding code generators, using templates enables future extension and customization. In order to support code reuse among templates, we define reusable templates as template snippets and utilize the "@include" directive of JET to import those snippets into the template. A total of 22 templates and 35 snippets are developed for different classes and scripts of Grid services. The service model and templates are weaved together by the code generator to generate the code and scripts. This is integrated as part of the project creation process.

A major design decision is to separate service modeling and code generation from other parts of GSDE. Due to the complexity of a practical Grid service implementation, synchronization between the model and codes is impractical, which always brings additional constraints or limitations.

7.3.6 Service Building

In GSDE, custom builders are implemented to generate stubs from WSDL. They work collaboratively with the Java builder that compiles all Java source codes in the project to Java classes. Multiple custom builders are implemented, each responsible for one step in the stub generation process:

1. Mapping Merge Builder: merge the project namespace2package.mappings file with the global one. In automatic incremental build, it is invoked upon changes in the mapping file. In full build, it is always applied on the single mapping file.

2. WSDL Flatten Builder: flatten the compact WSDL by replacing inherited port types and data types with actual definitions. In automatic incremental build, it is invoked every time the WSDL has been modified. In full build, it is applied on all original WSDLs.

3. WSDL Binding Builder: generate binding and service WSDLs from the flattened WSDL. In automatic incremental build, it is invoked every time the flattened WSDL has been modified. In full build, it is applied on all flattened WSDLs.

4. WSDL Stub Builder: generate stub source codes from the service WSDLs. In automatic incremental build, it is invoked every time the service WSDL has been modified. In full build, it is applied on all service WSDLs.

Globus has provided Ant scripts for the generation of stubs from the original WSDLs. Our builders reuses the Java classes that implement the actual transformation, instead of directly invoking the Ant script due to its poor performance.

In both manual build and automatic incremental build, the builders are activated sequentially. Changes made by a builder will also be handled by subsequent builders. For Grid service projects, this sequence is defined as Mapping Merge Builder, WSDL Flatten Builder, WSDL Binding Builder, WSDL Stub Builder, and Java Builder. This guarantees that any change in the project will be correctly and fully handled.

7.4 Grid Service Execution Client

7.4.1 Overview

Grid clients help Grid users to utilize the rich functionalities provided by the services. Although command line clients can be used for simple Grid services, many services provide comprehensive functionalities and demand graphical user interface. On the other hand, due to the dynamic nature of Grid systems, new applications are continuously deployed on it in the form of custom services. In order to support them, the client must be extended and customized.

The Grid Service Execution Client (GSEC) is developed as a client environment that can be used as the base of custom client applications. It contains generic functionalities

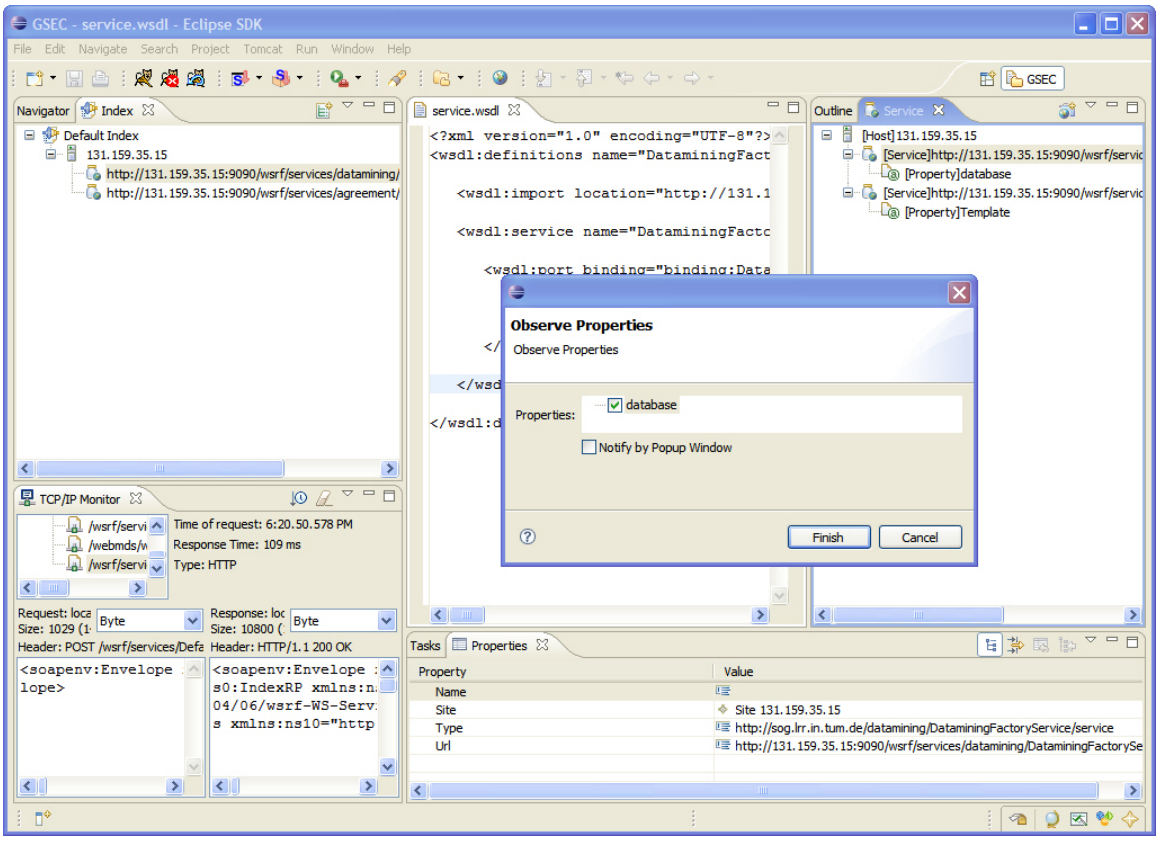

Figure 7.4 Grid Service Execution Client provides a base client platform (the screenshot shows the user interface for observing notifications of resource property changes).

and UI support for accessing the power of a service-oriented Grid built on top of Globus WS Core. Build on top of Eclipse, it is highly extensible and customizable. Client libraries and UI elements for additional custom services can be dynamically integrated into the application.

Figure 7.4 is a screenshot of the basic layout of GSEC. Its basic functionalities include:

1. Generic Web service support. It maintains a directory of services, which can be manually managed with add, remove, or modify. They can also be selectively imported from index services. It can be used to examine the WSDLs of existing services, and monitoring SOAP messages.

2. WSRF and WSN support. Besides accessing the basic operations supported by WSRF and WSN services, it can also start local notification listeners that observe changes of resource properties.

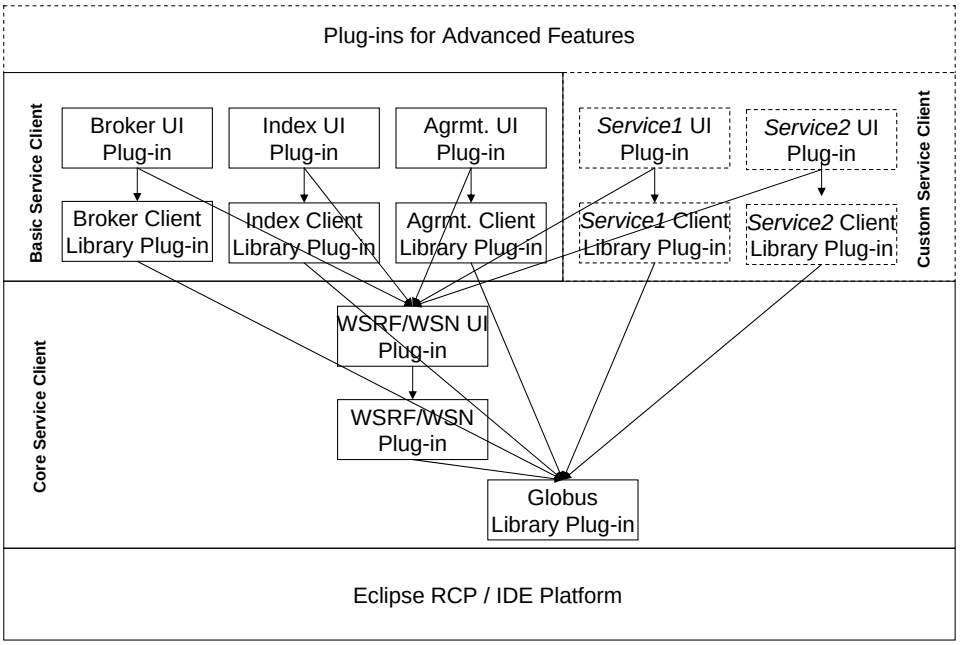

Figure 7.5 GSEC architecture and dependencies among plug-ins.

7.4.2 Agreement Infrastructure Support

Besides the basic functionalities, GSEC also incorporates customized support for our SLA-based resource management infrastructure. More specifically, it is extended to support the following infrastructure services:

1. Broker service. It allows the user to interact with a broker service and specify the parameters for the job to be brokered. It also allows checking the status of an existing job to be brokered.

2. Agreement service. It allows the user to establish agreements with service providers, browse the content and check the status of agreements, and cancel an existing agreement.

3. Index service. It allows the user to manage index services, and browse the content of index services.

API library and UI components for accessing each service are provided as plug-ins. They can be used out-of-the-box, or be integrated into the UI of custom services.

7.4.3 Components and Dependencies

Figure 7.5 shows the dependencies among the plug-ins. GSEC plug-ins can be virtually organized into several groups that are layered on top of each other. The first layer is a set of plug-ins that constitute the Core Service Client, including plug-ins that wrap the Globus

libraries, plug-ins that provide UI and client libraries for WSRF/WSN port types. The second layer is formed by plug-ins that provide UI and libraries for infrastructure services as the Basic Service Client, and those for custom services as the Custom Service Client. Beneath the Core Service Client is the Eclipse IDE or RCP platform, and above all these plug-ins are those that provide advanced features based-on all underlying layers.

From a higher level point of view, the architecture of the system can also be divided into UI layer and non-UI layer. It should be noted that non-UI plug-ins are not dependent on any UI-plug-ins. This makes it possible to further develop a non-GUI client based on the same architecture, with the support of Eclipse for "headless" (i.e. non-GUI) applications.

7.4.4 External Libraries and Class Loadpaths

The formal approach for introducing a third-party library into Eclipse is to implement a plug-in that physically contains the library. The advantage of this approach is that it maintains strict version dependency relationships among all Eclipse plug-ins. However, for libraries that are part of the dependent middleware platforms, wrapping them into plug-ins means duplication. This is also not convenient for libraries that are frequently updated.

Globus library plug-in allows dependent plug-ins to access Globus libraries. Globus is implemented in a way that strongly depends on system variables to decide its load path to access several configuration files like client-config.wsdd and client-jndi-config.xml if the client needs to receive notifications. Therefore, a Globus WS Core is usually installed on the client.

We have implemented the Globus library plug-in in a special way that allows libraries in an external Globus installation to be accessed by dependent plug-ins. In the MANIFEST.MF file that describes the classpath of the bundle, we manually replace references to jars to an external location. Besides, we put GLOBUS_LOCATION in the front of the classpath, to ensure that Globus finds the correct configuration files.

```
Bundle-ClassPath: external:$GLOBUS_LOCATION$,
 external:$GLOBUS_LOCATION$/lib/addressing-1.0.jar,
 external:$GLOBUS_LOCATION$/lib/axis.jar,
 ...
```

Due to a limitation of the Eclipse plug-in development environment, the references to external JARs are not recognized. So, the modified MANIFEST.MF can not be applied at development time, therefore the plug-in project still has to physically include Globus JARs. Once the plug-in are deployed, the modified MANIFEST.MF file will be applied and those JARs are no longer necessary at run time.

7.4.5 Dynamic Extensions for Custom Services

Eclipse allows easy extension of the basic client platform for new custom services with plug-ins. Eclipse plug-ins are actually OSGi [92] bundles, which is a Java JAR with additional manifest identifying the bundle and laying out its dependencies. The Eclipse runtime

is an implementation of OSGi framework, that manages the plug-ins (bundles) dynamically, allowing them to be dynamically loaded, unloaded, updated, started and stopped at run time. This also applies to the plug-ins that contains libraries and UI components for custom services.

In addition to the underlying OSGi mechanisms, Eclipse provides Update Manager as GUI-level support for the installation, update and management of plug-ins. The basic unit of the Update Manager is "feature", which represents a collection of tightly coupled plug-ins offering highly related functionalities. Therefore, the individual plug-ins that provide client libraries, non-UI functionalities, and UI contributions for a single custom service need to be wrapped into a feature. The feature can be published online as an update site, with which users of the Grid service can extend or update their client applications.

7.5 Grid Service Management Environment

7.5.1 Overview

The management of Grid systems and applications involves many discrete activities. Such activities include installation of Grid middleware, deployment of Grid services, monitoring Grid systems, etc. Therefore, Grid managers need support environments to assist those administrative activities. Service Management Environment (GSME) is developed as such an environment.

Figure 7.6 is a screenshot of the basic layout of GSME. Its basic functionalities include:

1. System management. It allows accessing remote file systems and execute commands in consoles. It reuses the Remote Systems explorer and Remote Shell from Eclipse DSDP/TM project.

2. System monitoring. Currently it reuses certain parts of GSEC to check the status of systems from information published by the Index Services. This can be extended to integrate with more advanced Grid monitoring systems.

3. Grid service deployment. It can be used to deploy Grid service GARs on remote Grid servers. It is based on parts of GSDE that offers such functionality.

4. Service clients. By reusing parts of GSEC, it allows to test the deployed services.

7.5.2 Components and Dependencies

GSME plug-ins are mostly reused from GSDE, GSEC and Eclipse DSDP/TM. Figure 7.7 provides a simplified picture that shows the composition of GSME. This provides a good example of developing new tools by weaving existing applications on the same platform.

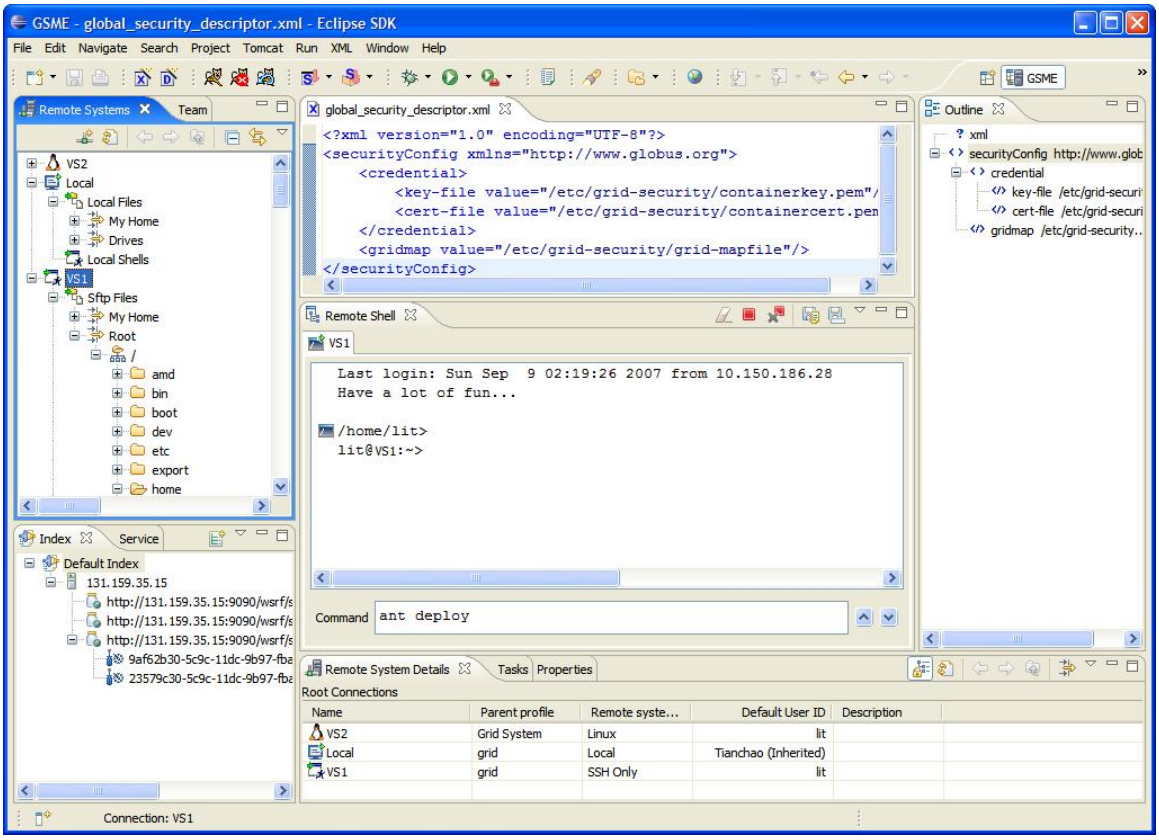

Figure 7.6 Grid Service Management Environment support remote system management.

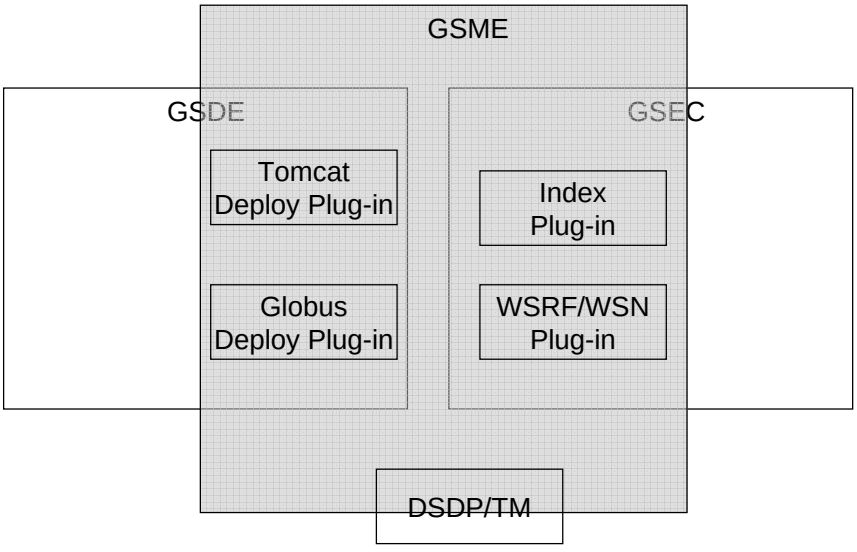

Figure 7.7 GSME is composed of plug-ins from GSDE, GSEC and DSDP/TM.

7.6 Related Work

7.6.1 Grid Service Development Tools

GT4IDE [4] was the first attempt to implement a development tool for GT4 based Grid services. It was built on top of Eclipse platform and JDT. It provides simple code generation functionality based on limited information provided in a wizard, and simple project configuration that creates the project structure and configures the classpath. It relies on custom Ant scripts for project build which is not integrated with Eclipse build system. Due to its limited functionality and buggy implementation, GT4IDE is not applied in actual development work. It has, however, inspired our development of GSDE and similar works that progressed independently.

The Grid Development Tools (GDT) [5] provides a programming environment similar to our service development tool. It requires the user to follow specific programming pattern by annotate Java interfaces with custom tags as means of service modeling. It is unable to model arbitrary relationships between Grid services and resources, and also restricts the user from programming complex services on top of the generated code.

The Introduce Grid Service Authoring Toolkit [6] is a self-contained GUI that provides service modeling, code generation and service deployment support. As a stand-alone application implemented with Java Swing, it is not able to benefit from the rich functionalities of a real IDE, such as Java coding support, automatic build, interactive debugging. It also does not allow the integration of third-party tools.

7.6.2 Grid Client and Management Environments

The g-Eclipse [7] project aims to build an integrated workbench framework on top of Eclipse to access the power of existing Grid infrastructures. The framework targets three user roles - Grid application developers, Grid application users, and Grid operators, and provides tools for application development, deployment and debugging, job management and monitoring, data management, as well as VO management, infrastructure configuration, monitoring and benchmarking. The current design of g-Eclipse only support standard Grid services and its current implementation is limited to gLite.

Intel Grid Programming Environment (GPE) [8] provides an environment for using the standard services from Globus [47] and Unicore [39]. It provides several client applications for Grid users with different expertise and a support environment for administrators. In addition to the standard Grid services, GPE allows to define higher-level custom services that take advantage of the standard ones. Service-specific user interfaces are defined with GridBeans, which are published in GridBean service and are dynamically integrated in different clients. Although the GridBeans allows GPE clients to display custom UI compo-

[4] http://gsbt.sourceforge.net/content/view/12/29/
[5] http://ds.informatik.uni-marburg.de/MAGE/gdt/
[6] http://www.cagrid.org/mwiki/index.php?title=Introduce
[7] http://www.geclipse.org/
[8] http://gpe4gtk.sourceforge.net

nents for custom services, more advanced customizations that involve custom interactions of multiple services is not possible.

Examples of simpler Grid client environments include those of P-GRADE [66], ASKALON [40] and GEMLCA [35]. All of them only work with standard or predefined services and does not support application-specific extensions or customizations.

7.7 Conclusion

A comprehensive set of tools and environments for the development, execution, and management of applications in the SLA-based Grid environment has been developed. It includes tools for application service development, basic client environment that can be extended to access the Grid applications, tools for the management of underlying Grid infrastructure and the deployed services. All the tools and environments are seamlessly integrated with the Eclipse platform, and can be freely composed according to the needs of the user as Grid service developer, Grid service user, or Grid system manager. They are also extensible and customizable for the usage in a special scenario.

The tools and environments have been practically applied in our daily work. They are used in the development of the SLA management services in Chapter 3, and custom application services in Chapter 8 where they are also used to develop custom application clients.

Chapter 8

Demonstration

8.1 Introduction

The WS-Agreement support infrastructure presented in Chapter 3, the run time prediction framework presented in Chapter 4, the probability aware scheduler presented in Chapter 5, and the global infrastructure presented in Chapter 6 constitute the major basis for a SLA-based resource management infrastructure outlined in Chapter 2.

As part of the Automated Resource Management for Large Scale Applications project, a demonstration of SLA-based resource management has been developed. The demonstration is based on a concrete application scenario of distributed data mining for banking. The SLA-based resource management infrastructure is integrated with custom Grid services for data mining. And custom clients that provide easy to use user interfaces are implemented by extending and customizing the GSEC base client. The development tools and support environments introduced in Chapter 7 provide considerable support for the development of related services and clients.

This chapter focuses on the development, deployment and usage of the demonstration. The remaining part of this chapter is organized as follows. Section 8.2 describes the concrete application scenario of the demonstration from the perspectives of different users, and Section 8.3 explains the resource management activities that happen behind the scene. Section 8.4 presents the deployment environment of the demonstration. The design and implementation of the custom data mining services are presented in Section 8.5, and that of two client applications for data specialists and normal data analyzers are presented in Section 8.6. Section 8.7 concludes this chapter with a brief summary.

8.2 A Concrete Scenario for Distributed Data Mining in Banking

The scenario is a concrete application of the distributed data mining scenario described in Section 2.4.

In a bank, transaction data are collected in data warehouses and data mining techniques are used to analyze the data to assist different business decisions. To meet the need of resources for such analysis, resource provisioning based on Grid techniques is applied. A pool of servers are started up with Grid middleware together with the SLA-based resource

management infrastructure pre-installed, and Grid services are deployed and managed by the resource management infrastructure to dynamically serve the request of users.

To assist normal users that have no knowledge of the IT infrastructure and no background of data mining techniques, the Grid services are not developed by the end users but by data specialists. With specialized tool support, data specialists specify details of the data mining job by composing different data and data processing methods into a mining flow. A set of parameters that are accepted by the mining flow is also specified. Multiple mining flows are developed, each for a different analysis. The developed mining flows are then deployed onto servers in the server pool in the form of custom data mining services. The deployed services automatically register themselves to the Grid index, so that they can be known. Each mining service is associated with method name that represents the type of analysis it is capable of, together with a description of that method. The Grid index virtually groups all services (hence the server) that belong to the same method together.

A colleague in the bank, say a marketing staff, wants to analyze the data. As a first step, he looks up the Grid index for existing data mining methods and chooses one that meets his needs. This is assisted with supplied client application that provides easy to use interface. With the same program, he submits a request together with mining parameters and specified deadline. After a while, he is notified that his request is accepted and scheduled. Otherwise, if his request is rejected because the existing resources are not sufficient to meet the deadline, he can modify his request and resubmit. From now on, he does not have to keep the client program running, but if he does he will be notified as soon as the job finishes. Otherwise, he can choose to check the status of his request whenever he wants. When the job finished, he can view the result of the analysis or in rare cases check the error.

8.3 Resource Management Activities in the Demonstration

Complexities of Grid resource provisisioning are hidden in the activities described above, because the SLA-based resource management infrastructure automates the whole process in the backend.

The client application handles user request by looking up Grid indexes for existing data mining services that supports the requested method. One out of the many candidates is chosen, and the client application tries to, through the WS-Agreement support infrastructure, establish an agreement on serving its work before the specified deadline. More concretely, the client application contact the associated WS-Agreement factory service to get the available SLA templates, and send a SLA offer by modifying the template with details of the analysis parameters and required deadline. With the help of performance predictors and the local scheduler, the WS-Agreement support infrastructure for the chosen data mining service will be able to decide whether it is able to serve the request before specified deadline and it will accept or reject the SLA offer accordingly. If the SLA offer is accepted, it will notify the client application with a reference to the created SLA. Otherwise, the client application will try to negotiate with another service until succeed. And if all the data min-

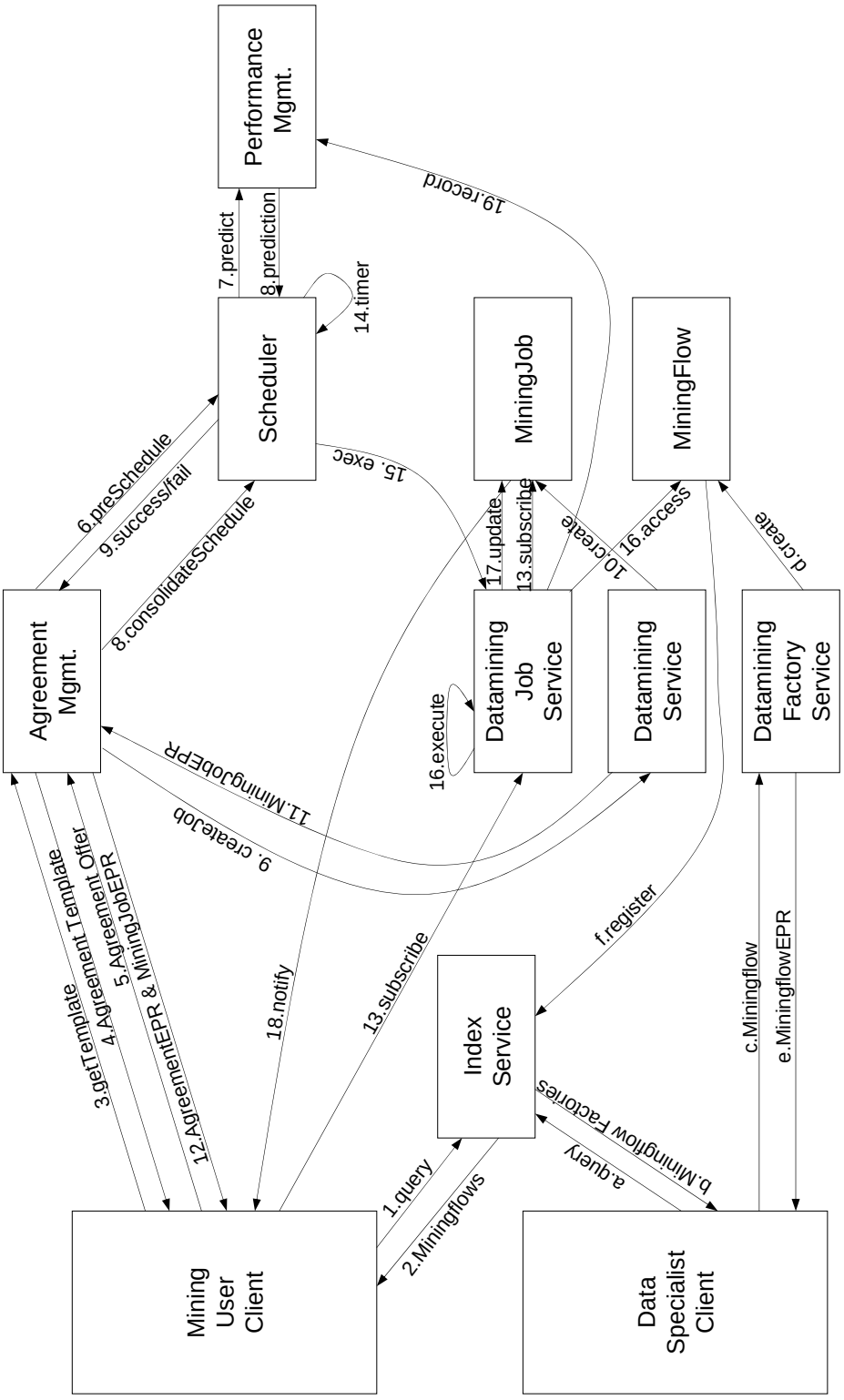

Figure 8.1 A schematic figure showing major components and activities of the demonstration, from the perspective of data specialist (a to f) and marketing staff (1-19).

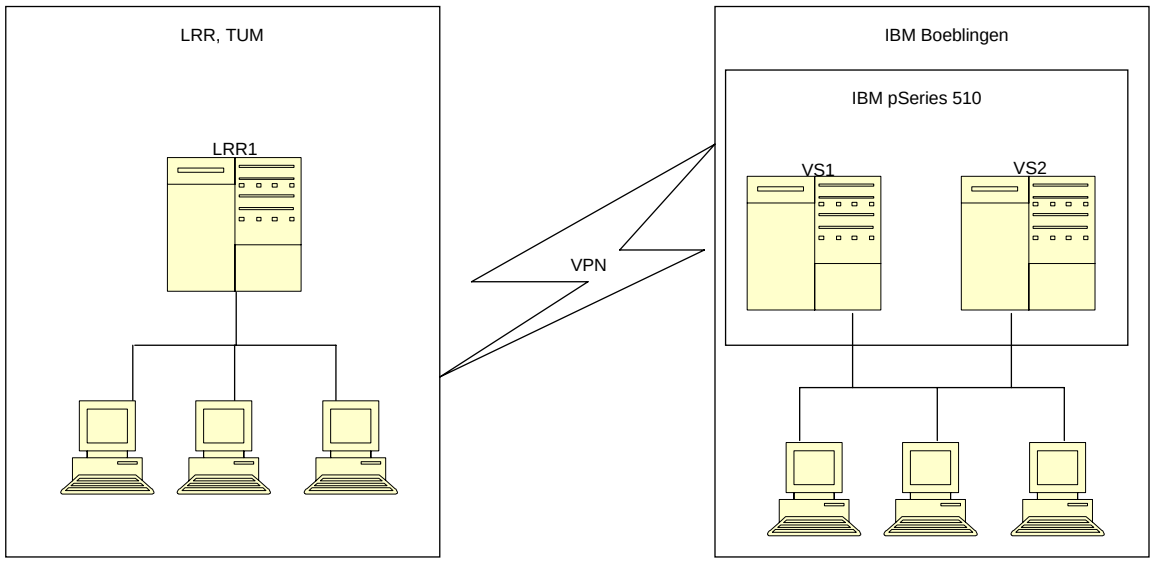

Figure 8.2 Deployment environment for distributed data mining demonstration.

ing services rejects the offer, the client application will notify the user. Once the SLA is created, the client application will invoke the data mining service with a reference to the SLA, which will result in the creation of a new data mining job. The data mining job is executed at a specific time controlled by the scheduler. Each time the job status changes, a notification message will be sent to the client application. The status and result of the job is also maintained as the service status, which can be retrieved at any time. A schematic graph of the involved components and major activities is shown in Figure 8.1. Activities labeled with a to f are those from the data analyst's perspective, and those labeled with 1 to 19 are for normal mining users, say the marketing staff.

Optionally, the client application can also delegate the negotiation of SLAs to one of the existing Grid brokers, which is itself a service deployed on one of the Grid servers. Before this, the client application has to look into the Grid index and choose the Grid broker. The broker acts on behalf of the client application to negotiate with the Grid servers, with the same process as described above. The benefit of this approach is that the user does not have to keep his program running during the whole process of SLA negotiation. But if he does, when the broker finishes the negotiation, he will be notified promptly. Otherwise, he can choose to check the result of negotiation at any time. From the user's perspective, this is also the only difference between using and not using a broker service.

8.4 Deployment Environment

As is shown in Figure 8.2, the demonstration is deployed onto a system formed by several servers located at IBM Boeblingen Laboratories (in short, IBM) and LRR of Technische Universitaet Muenchen (in short, LRR), and a number of clients. Through a VPN gateway, a virtual network between IBM and LRR is formed.

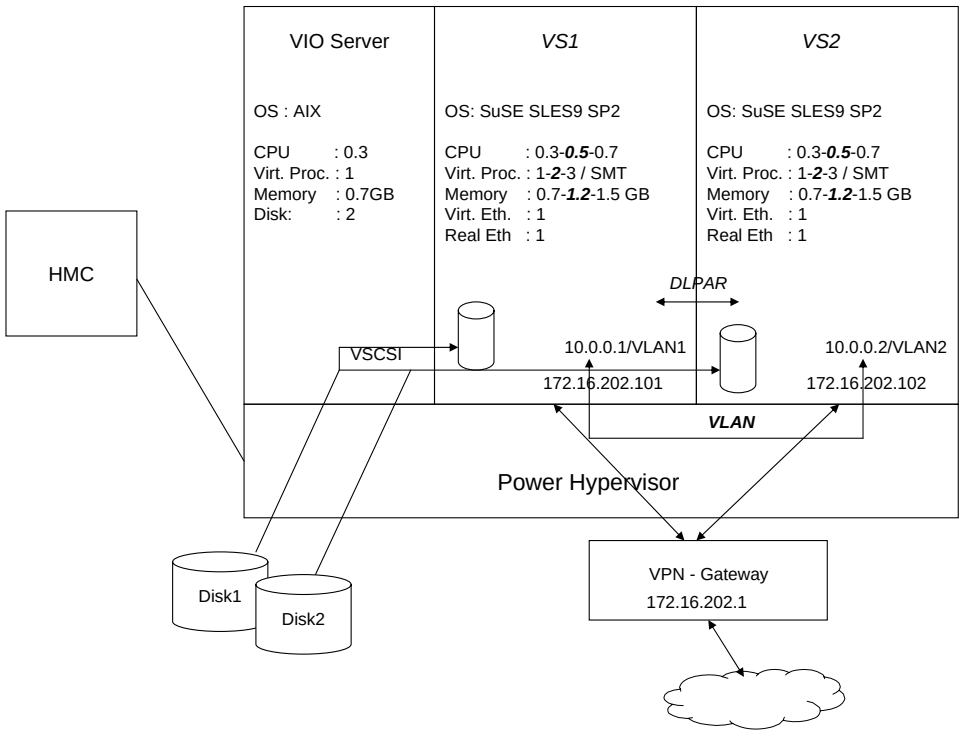

Figure 8.3 Virtual servers on IBM pSeries 510.

The server in IBM is an IBM pSeries 510 machine with 2 x 1.65GHz processors, 4 GB memory, and 2 x 70 GB hard disk. With advanced resource virtualization support of IBM hardware, two "virtual servers" are created, each with 2 x 0.5GHz CPU, 1.2 GB memory, and SuSE SLES9 SP2 as OS. Figure 8.3 shows the system structure, where those two servers are configured as logical partitions (LPAR). Besides, a special partition - virtual I/O (VIO) server - provides virtual disk storage (Virtual SCSI, VSCSI) and Ethernet adapter (Virtual LAN, VLAN) sharing. The Power Hypervisor is firmware for resource provisioning control, which can be operated from a Hardware Management Console (HMC).

The LRR server is Fujitsu-Siemens CELSIUS 670 XEON with dual Intel Xeon 2.8GHz processors, 2GB memory and 100GB hard drive. A Fedora 6 with Linux kernel version 2.4.20 is installed as OS.

Globus WS Core v4.0.4 and the SLA management infrastructure are installed on all servers as the platform. The data mining services implementation and customization bundle is deployed on the platform.

8.5 The Service Bundle

8.5.1 Data Mining Services

The deployment of mining flows and the management of mining jobs are supported by a set of services. The data mining services are designed and developed following the WS-

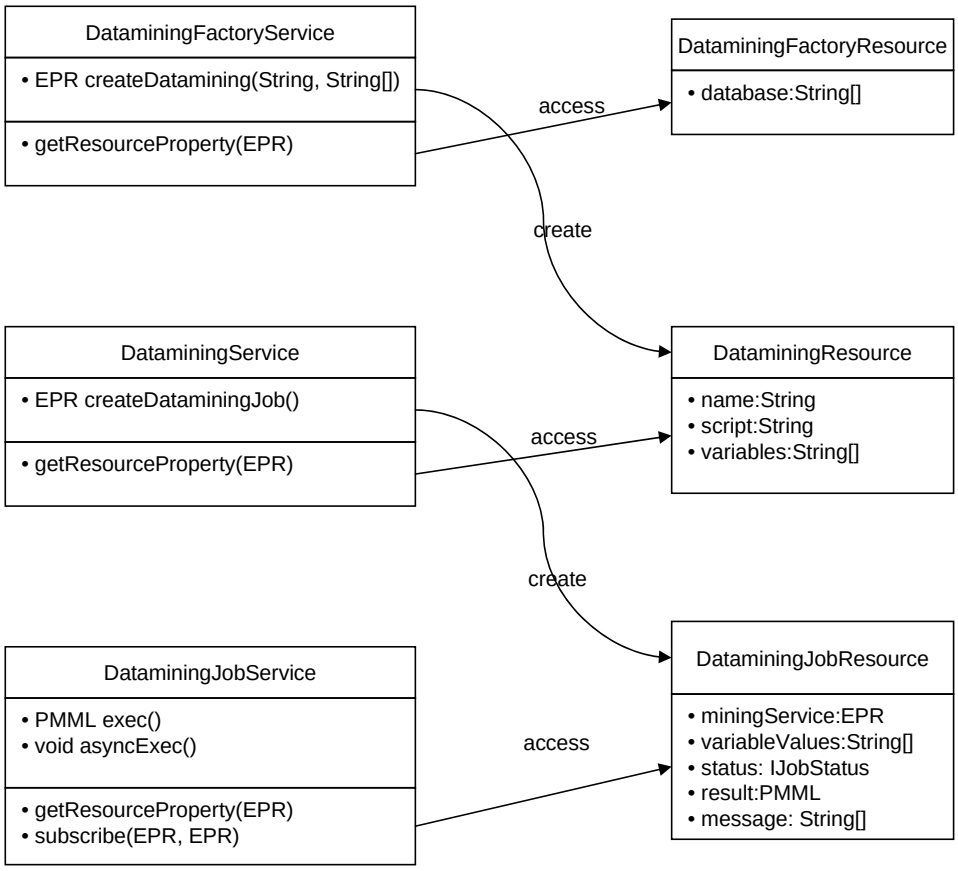

Figure 8.4 Application services for the data mining demonstration.

Resource and WS-Notifications specification. Three services provide accessible interface for three resources holding information on accessible data, and representing deployed mining flow and the concrete data mining job, as is shown in Figure 8.4.

DataminingResource and DataminingJobResource are resources for mining flows and mining jobs. DataminingResource holds the mining flow and the name of variables that are defined in the mining flow. DataminingJobResource holds the actual variable values, status of the job, the result of the mining flow in the form of PMML [58], and an array of messages representing details of the mining process. It also maintains an endpoint reference (EPR) to the associated DataminingResource, so that the corresponding mining flow can be accessed.

The DataminingFactoryResource is a special resource entity. It is a singleton and dedicates to providing information. As the only property of this resource, the actual list of databases that are accessible from this server is provided. The actual list of databases is initialized during service deployment, and can be updated later by privileged administrators.

The DataminingFactoryService provides an interface for the creation of DataminingResource, which is referenced by an EPR. The DataminingService, as a stateful Web service,

supports accessing the properties of the associated DataminingResource instance, through interfaces defined by WS-ResourceProperties. It also provides an interface for the creation of DataminingJobResource. DataminingJobService, another stateful Web service, supports accessing the properties of the associated DataminingJobResource instance through interfaces provided by WS-ResourceProperties and subscribing and notification of status changes through interfaces provided by WS-Notification. It also contains two methods that control the job execution in synchronous or asynchronous way.

For demonstration purposes, the implementation utilizes IBM Intelligent Miner as part of the IBM DB2 Data Warehouse Edition 9.1 product for data mining. However, other data mining toolkits or products can also be applied in a general sense.

8.5.2 Customization Associates in Service Bundle

Together with the implementation of the services and resources, the service bundle that is deployed on the server also contains code and scripts for customizing the SLA management infrastructure. While all necessary customizations have been described in previous chapters, we intend to provide a case study based on our demonstration.

Custom Agreement Providers

Two agreement providers are implemented, including DataminingAgreementTemplateProvider and DataminingAgreementCreationProvider. While DataminingAgreementTemplateProvider creates a custom agreement template, DataminingAgreementCreationProvider integrates the local resource manager with the WS-Agreement support infrastructure.

Custom Data Recorder and Predictor

Custom argument recorder DataminingJobArgumentRecorder and result recorder DataminingJobResultRecorder is implemented to use database to store data.

Custom performance predictor DataminingPerformanceEvaluator that applies data mining to predict performance.

Proactive Handler Registration

DataminingJobArgumentRecorder and DataminingPerformanceEvaluator are configured in server-deploy.wsdd in the way described in Section 4.6.1 and 4.6.4.

The DataminingInitializationService is configured to proactively register the agreement handlers, service launcher, and performance evaluator:

```
IAgreementHandlerConfiguration provider =
  new DataminingAgreementHandlerConfiguration();
ProviderRegistry.registerAgreementHandlerConfiguration(
  provider);
```

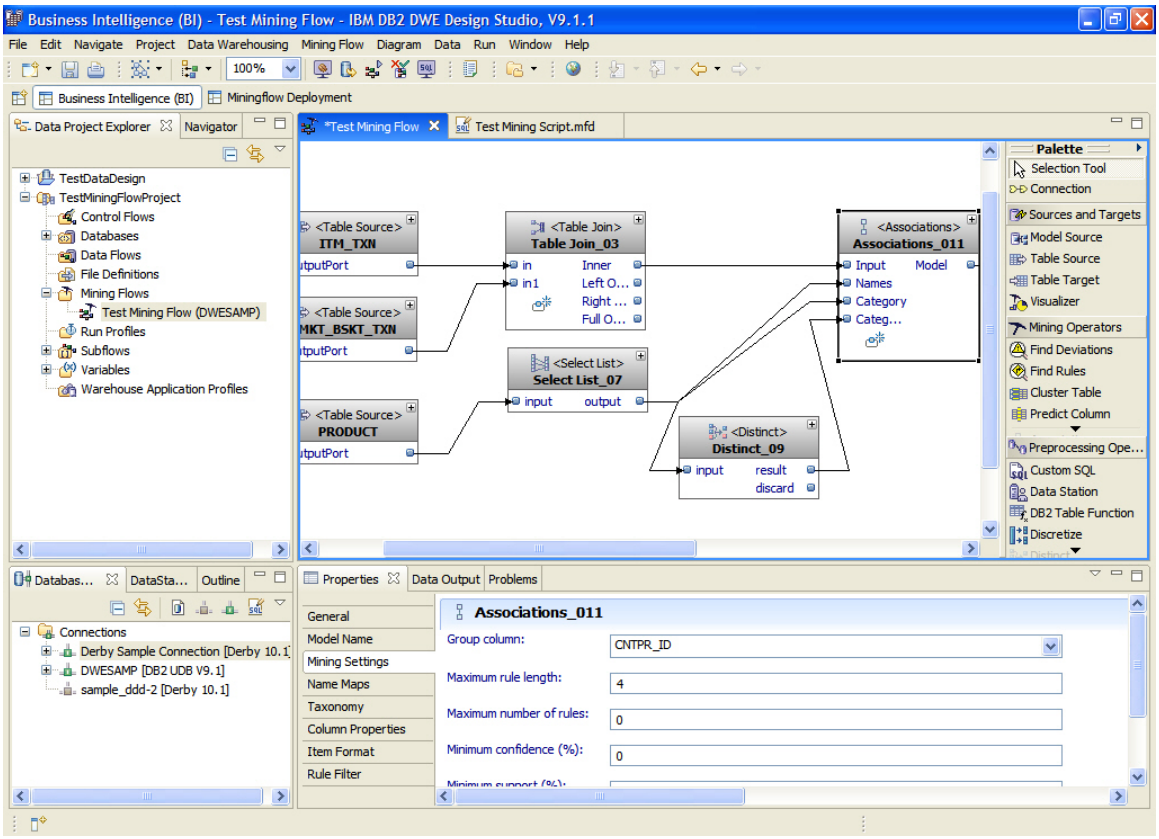

Figure 8.5 Designing mining flow with Mining Flow Development and Deployment Client.

```
ILauncher launcher = new DataminingServiceLauncher();
LauncherRegistry.register(
  DataminingJobParameters.class, launcher);

IPerformanceEvaluator evaluator =
  new SimpleDataminingPerformanceEvaluator();
PerformanceEvaluatorRegistry.register(
  DataminingJobParameters.class, evaluator);
```

8.6 Client Environments

8.6.1 Overview

By extending the base client environment GSEC described in Section 7.4, two custom client applications have been developed to meet the different needs of data specialists that design and deploy data mining services, and normal data analyzers that utilize existing data mining services.

Based on the component infrastructure of GSEC, the clients are implemented as a set

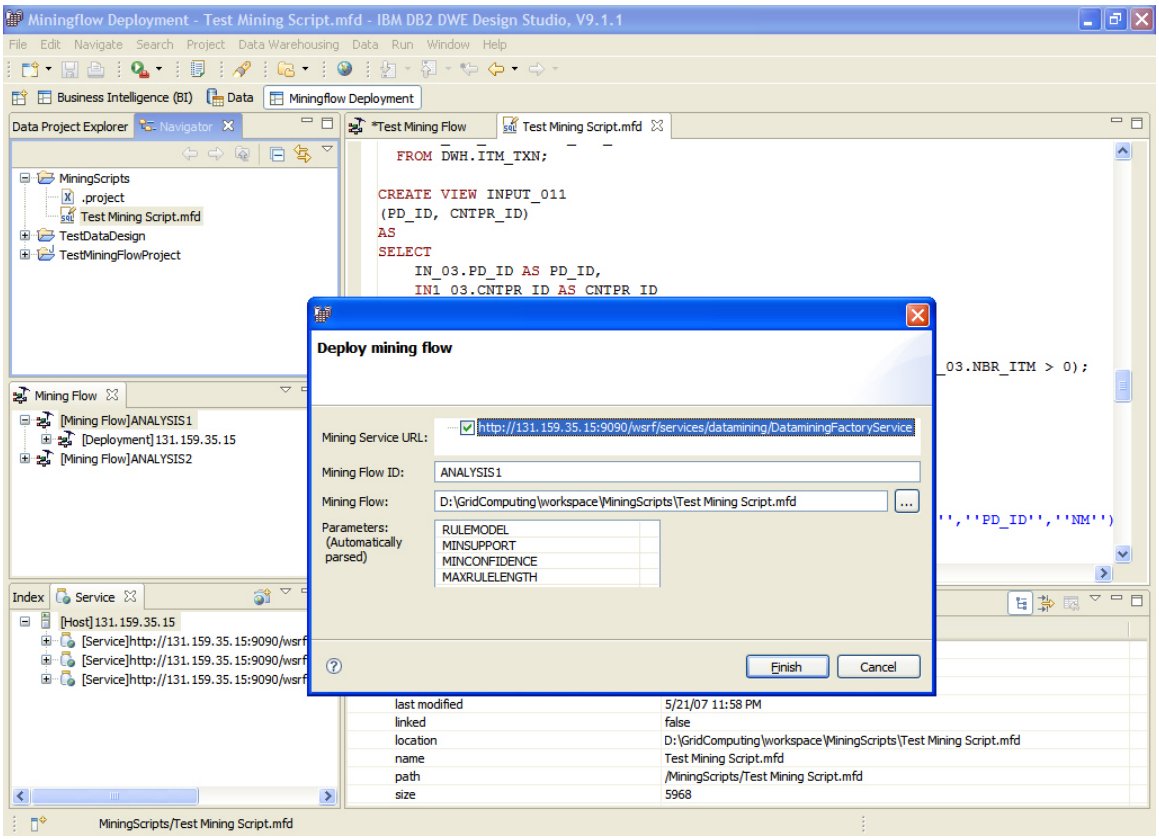

Figure 8.6 Deploying mining flow with Mining Flow Development and Deployment Client.

of plug-ins. A MiningService Library plug-in wraps the client libraries for data mining services. A MiningService UI plug-in implements the basic UI extensions that are shared between the two clients. Two separate UI plug-ins provide the higher-level UI components to data specialist and normal data analyzers.

The common UI components offered by MiningService UI plug-in includes a Mining Service View and a Mining Flow and Job View. The former provides information of available servers that are feasible for mining flow deployment by maintaining a list of DataminingFactoryServices. The later provides information of all deployed mining flow and associated mining jobs. It is a customization of the standard service view - instead of listing all deployed services, it focuses on DataminingFlowService and DataminingJobService. It groups existing DataminingFlowServies according to their method name and maintains a hierarchical relationship between DataminingFlowServices and DataminingJobServices.

8.6.2 Mining Flow Development and Deployment Client for Data Specialist

The Mining Flow Development and Deployment Client is a customized client application that helps data specialists to design and deploy mining flows. In addition to the standard

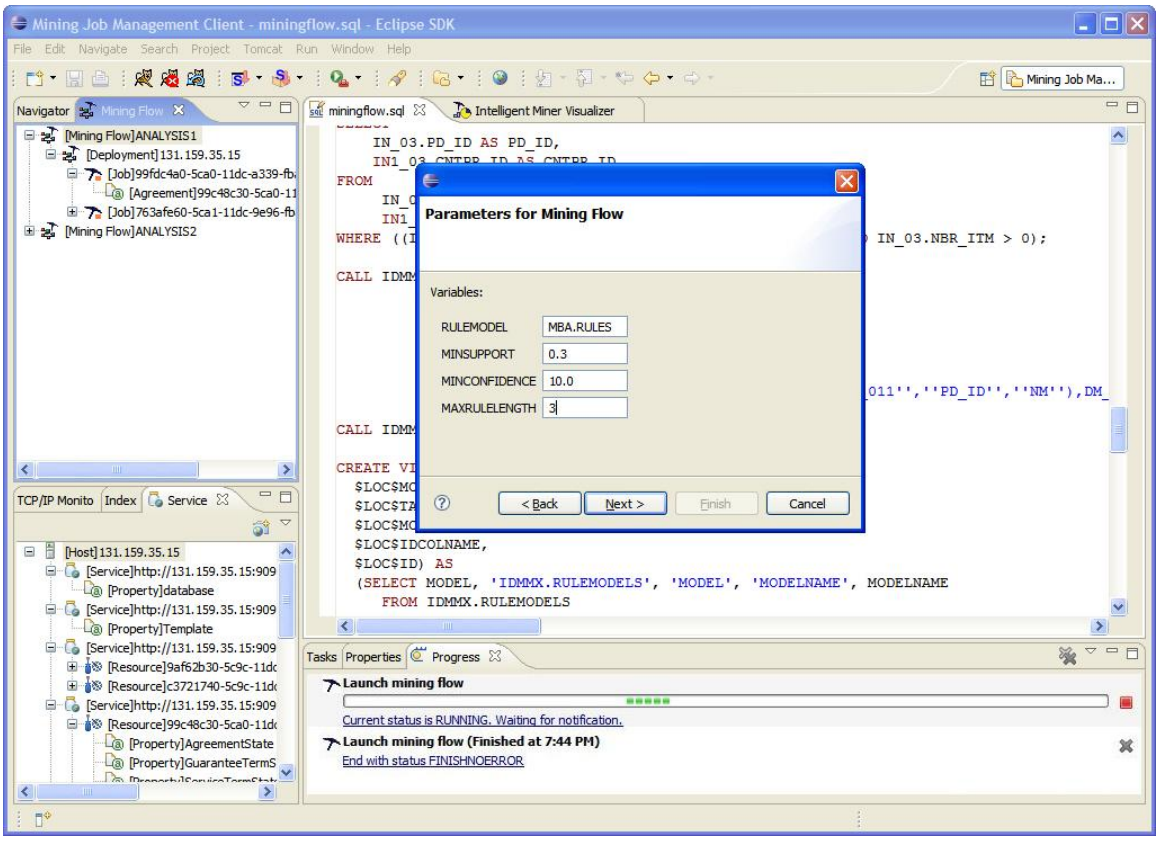

Figure 8.7 Start mining job with Mining Job Management Client.

functionality of GSCE and the basic library and UI plug-ins for data mining services, it provide high-level user interface, including Mining Flow Editor and Mining Flow Deployment Wizard.

The Mining Flow Editor is a graphical editor for mining flow design, originally part of IBM Design Studio for DB2 Data Warehouse. This gives an example of easy integration of third-party tools that are based on the same Eclipse platform into our client application.

The Mining Flow Deployment Wizard is a wizard that helps data specialist to deploy the developed mining flow to selected servers. It is based on functionalities and UI components of lower level plug-ins for index, agreement, and the custom data minining services.

Figure 8.5 presents a screenshot of the client application in mining flow design. And deploying mining flow with the client application is shown in Figure 8.6.

8.6.3 Mining Job Management Client for Data Analyzer

The Mining Job Management Client for Data Analyzer is a separate client application that helps normal data analyzers to use the mining flows deployed on the Grid. In addition to the standard functionality of GSCE and the basic library and UI plug-ins for data mining services, it also provides high-level user interfaces including Mining Job Launch Wizard

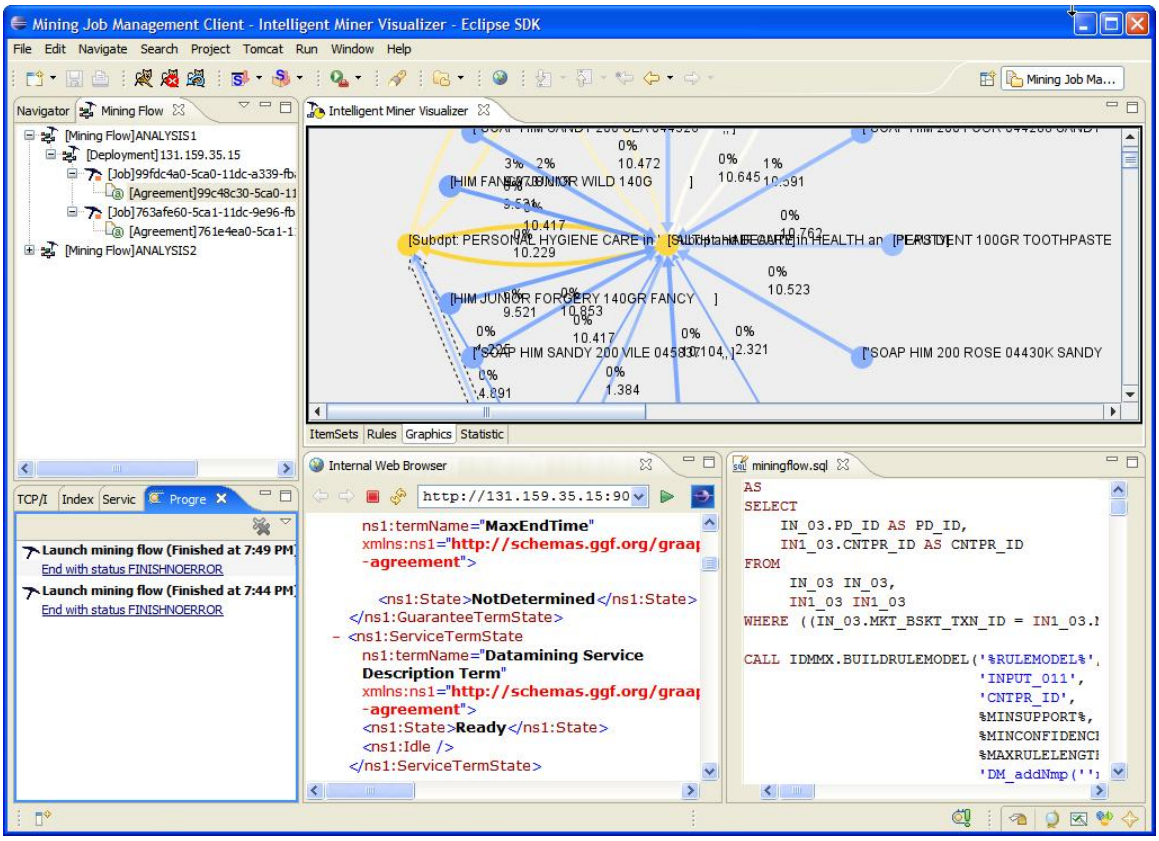

Figure 8.8 View mining job, mining model, and agreement with Mining Job Management Client.

and Mining Job Result Viewer.

The Mining Job Launch Wizard provides step by step guidance that helps the data analyzer to request the execution of specific mining flow. It is based on functionalities and UI components of lower level plug-ins for index, agreement, and the custom data minining services.

The Mining Job Result Viewer visually renders the resulting data mining model. It is developed by wrapping existing functionalities from IBM DB2 Intelligent Miner Visualization. This gives an example of easy integration of third-party tools into our client application by wrapping them into plug-ins.

Figure 8.7 presents a screenshot of the client application in launching mining job. And viewing mining job, agreement, and mining model with the client application is shown in Figure 8.8.

8.7 Conclusion

To illustrate the applicability of our SLA-based resource management with predictive performance evaluation approach, a demonstration for distributed data mining has been im-

plemented. It is consisted of custom application services for mining flow deployment and mining job execution. Custom clients are developed to meet the need of different users. It is deployed and tested on a small-scaled Grid system in a virtual network between LRR of TU Muenchen and IBM Boeblingen Laboratories, which justifies the applicability of our development.

The demonstration also shows how application specific customization of the SLA-based resource management infrastructure can be easily achieved with the help of accompanied support environments. Custom agreement handler, data recorder and predictor, as well as service launchers, can be easily implemented by extending existing implementations and configured. Implementation of client environments can also be greatly simplified by extending the base client platform GSEC.

Chapter 9

Conclusions

9.1 Summary

The work presented in this thesis focuses on applying service level agreements in service-oriented systems for autonomous resource management. The basis of this approach is a SLA-based resource management infrastructure, which incorporates a global infrastructure assisting SLA negotiation and a local management infrastructure for SLA enforcement which manages local resources and handles agreement negotiation and service provisioning. Major issues towards the establishment of such an infrastructure have been addressed in this work.

The specification of SLAs is accomplished with an emerging standard language WS-Agreement, developed by the GRAAP working group of Open Grid Forum. We have been actively participating in this group and contributed to the finalization of this specification. A generic and extensible support infrastructure for WS-Agreement has been designed and developed, which utilizes a provider mechanism for custom domain specific processings. It is the first full implementation of WS-Agreement on GT4 platform and the only one that can be dynamically extended for different Grid services coexisting on the same server.

For application performance prediction, we have designed and implemented a generic run time monitoring and prediction framework for service-oriented Grid applications. The framework achieves non-intrusive run time monitoring with custom handlers inserted in the SOAP handler chain. It allows incorporating different performance prediction techniques, including analytical modeling, statistics simulation, and historical data analysis. A set of generic performance predictors that follows a systematic approach for applying data mining techniques has been developed and evaluated.

As a major requirement of the local resource management infrastructure, the local scheduler controls the execution of services that are provisioned. For jobs with a run time that are derived with prediction techniques, the probabilistic nature of the estimated run time imposes additional requirement on the scheduling. Besides, jobs in a SLA-based resource management infrastructure has additional features that need to be considered, for example the different possibilities of earliest job start time, lazy termination and alternative SLA offers. A hybrid scheduling approach that combines global scheduling and local scheduling has been developed. A scheduler that controls the execution of provisioned services has been implemented based on the hybrid scheduling approach, which is integrated with the WS-Agreement support infrastructure and the run time monitoring and prediction

framework to form the local resource management infrastructure.

A global infrastructure that assists the establishment of SLAs with unknown services and the enforcement of SLAs for stateless Web services has been developed. This infrastructure incorporates three services that help the discovery and selection of service providers and assist provisioning of stateless Web services - index service, broker service, and gateway service. The open nature of the global infrastructure makes it possible to incorporate other services, for example those supporting service trading, to support advanced means of SLA negotiations with improved scalabilities.

In order to support the development, usage and management of Grid systems managed by our SLA-based resource management infrastructure, a comprehensive set of development tools and support environments has been designed and implemented. The environment is intended to support users of three different roles - Grid application developers, Grid application users, and Grid managers. Based on platforms like Eclipse that support component-based application development, the tools that are developed can be dynamically composed or custom extended to meet the needs of a specific user.

Based on a concrete scenario of distributed data mining for banking, a demonstration of the SLA-based resource management approach has been implemented. The demonstration incorporates techniques that have been developed for WS-Agreement support, application performance prediction and scheduling of jobs with probabilistic run time, and utilizes the global infrastructure that assists resource discovery, resource selection and job execution. The demonstration also serves as a general evaluation of the SLA-based resource management approach and related techniques that has been developed in this work.

9.2 Future Work

The work presented in this thesis has been a major step towards automated Grid resource management with SLA-based approach. On top of this, a number of possible areas exist that deserves to be continuously explored in the future.

9.2.1 Workflow Support

Individual services can be composed into workflows, the execution of workflows need to be investigated. The SLA-based resource management infrastructure provides the fundamental support for co-scheduling the execution of different services, which can be leveraged for workflow support. This mainly involves the development of a workflow service that plans the execution of each individual service of the workflow and makes reservations through the SLA management infrastructure. In addition, the development and management tools need to be extended to support the composition and execution of workflows, including the development of a workflow editor and client for workflow services. The existing development tools and support environments can be used in the development and deployment of the resulting services and client components.

9.2.2 Grid Economy

As one of the topics for future work, we are looking forward to a generic area of resource management with Grid economy, i.e. market-oriented resource negotiation. This can be achieved with an extension of current global infrastructure with services that support service trading and financial management. Besides the various services outlined in previous chapters that support different trading methods, services for financial management are also needed. They regulate and manage Grid currency or virtual Grid currency (quota) and supporting the financial activities during service trading.

9.2.3 Subcontract SLAs

Besides co-scheduling for workflow support and Grid economy supporting market-oriented resource negotiation, another topic that can be investigated is to subcontract SLAs. Service provider might negotiate with other service providers to make a subcontract for taking over whole or part of the service provisioning. The application scenarios of subcontract SLAs as well as the exact means of achieving this goal need to be investigated. Again, the current SLA-based resource management infrastructure can be leveraged as the basis.

9.3 Concluding Remarks

In summary, the work presented in this thesis has implemented a comprehensive approach for automated resource management with service level agreements in the context of service-oriented Grids. Infrastructure and techniques for automated negotiation and management of service level agreement, learning-based resource requirement prediction, and probability-aware scheduling are designed and developed, that forms a local resource management infrastructure. This is complimented with a global infrastructure that assists the negotiation of SLAs between clients and unknown services, as well as the enforcement of SLAs.

The infrastructure and techniques developed in this work establish the fundamental basis for future developments.

Appendix A

Adaptation of WS-Agreement for GT4

A.1 Namespaces

The namespaces used by WS-Agreement specification are consistent with those of Apache WSRF, a newer but currently less popular WSRF implementation. The current release of GT4 uses older namespace definitions compared to those used by WS-Agreement specification, so it is necessary to replace them with those supported by GT4. A list of the modifications to the namespaces is provided in Table A.1.

Of course, there are foreseeable cross-platform interoperability issues. However, before GT4 is updated to conform to the newer namespaces in forthcoming versions, there are no better solutions. The developers of GT4 have been aware of this issue and are working on it.

A.2 WSDL and XSD Imports

The original WSDLs and XSDs of the WS-Agreement specification have WSDL imports and XSD imports from addresses specified in URLs. GT4 does not allow the import of external WSDLs and XSDs. Instead, it requires all WSDLs and XSDs organized in the schema directory.

Furthermore, as is already mentioned, the namespaces used by GT4 are different from those of the specification, so the locations of the imported WSDLs and XSDs should be changed accordingly. Please see Table A.2 for a list of the import locations.

A.3 Faults

WS-Agreement specification references wsrf-rw:ResourceUnavailableFault. The corresponding namespace used by GT4 is wsrf-rpw. However, although the older definitions used by GT4 do share most of the fault types like wsrf-rpw:ResourceUnknownFault and wsrf-rpw:InvalidResourcePropertyQNameFault, wsrf-rw:ResourceUnavailableFault is not supported. A work around is to use an existing similar fault type wsrf-rw:ResourceUnknownFault.

NS	WS-AG Spec	GT4
wsrf-rp	http://docs.oasis-open.org/wsrf/rp-2	http://docs.oasis-open.org/wsrf/2004/06/ wsrf-WS-ResourceProperties-1.2-draft-01.xsd
wsrf-bf	http://docs.oasis-open.org/wsrf/bf-2	http://docs.oasis-open.org/wsrf/2004/06/ wsrf-WS-BaseFaults-1.2-draft-01.xsd
wsa	http://www.w3.org/2005/08/addressing	http://schemas.xmlsoap.org/ws/2004/03/addressing
wsrf-rpw	http://docs.oasis-open.org/wsrf/rpw-2	http://docs.oasis-open.org/wsrf/2004/06/ wsrf-WS-ResourceProperties-1.2-draft-01.wsdl
wsrf-rw	http://docs.oasis-open.org/wsrf/rw-2	N/A (use wsrf-rpw instead)

Table A.1 Necessary modifications to namespaces in WS-Agreement WSDLs.

NS	WS-AG Spec	GT4
wsrf-rpw	http://docs.oasis-open.org/wsrf/rpw-2.wsdl	../../wsrf/properties/WS-ResourceProperties.wsdl
wsrf-rp	http://docs.oasis-open.org/wsrf/bf-2.xsd	../../wsrf/faults/WS-BaseFaults.xsd
wsrf-rw	http://docs.oasis-open.org/wsrf/rw-2.wsdl	N/A (no import)
wsa	http://www.w3.org/2006/03/addressing/ws-addr.xsd	../../ws/addressing/WS-Addressing.xsd
wsrf-bf	http://docs.oasis-open.org/wsrf/bf-2.xsd	../../wsrf/faults/WS-BaseFaults.xsd

Table A.2 Necessary modifications to import locations in WS-Agreement WSDLs.

WSDL	definition name
Agreement.wsdl	Agreement
AgreementAcceptance.wsdl	AgreementAcceptance
AgreementFactory.wsdl	AgreementFactory
AgreementState.wsdl	AgreementState
PendingAgreementFactory.wsdl	PendingAgreementFactory

Table A.3 Adopted name attribute of wsdl:definition in WS-Agreement WSDLs.

A.4 Compact Schema

WS-Agreement port types extends port types of WSRF, thus the original WSDLs of WS-Agreement contains definitions "inherited" from the WS-ResourceProperty and WS-Lifetime. The mixture of new and "inherited" definitions makes the WSDLs less readable.

GT4 support WSDLPreprocessor which helps to simply WSDL definition with port type extension. The WSDLs are modified to use WSDLPreprocessor, by removing the "inherited" definitions and adding wsdlpp:extends attribute on wsdl:portType element:

```
<wsdl:portType name="Agreement"
  wsdlpp:extends="wsrf-rpw:GetResourceProperty"
  xmlns:wsdlpp=
  "http://www.globus.org/namespaces/2004/10/WSDLPreprocessor"
```

The original WSDLs from WS-Agreement specification does not define name attribute for wsdl:definitions element, which is optional according to the WSDL specification. However, the WSDL preprocessor of GT4 depends on this attribute for proper flattening of the WSDL document, for example:

```
  <wsdl:definitions name="Agreement" ...
```

Therefore, name attribute are added to the WSDLs. Table A.3 provides a list of the names that are added to the WSDL definition.

A.5 xs:simpleRestrictionModel and xs:typeDefParticle

ItemConstraint element in WS-Agreement specification is defined using xs:simpleRestrictionModel and xs:typeDefParticle:

```
      <xs:element name="ItemConstraint">
        <xs:complexType>
          <xs:choice minOccurs="0">
            <xs:group ref="xs:simpleRestrictionModel" />
            <xs:group ref="xs:typeDefParticle" />
          </xs:choice>
        </xs:complexType>
      </xs:element>
  ...
```

Due to a limitation of the data binding component in GT4, it fails to generate stubs for both xs:simpleRestrictionModel and xs:typeDefParticle. Compared to other platforms like Axis2, where different or alternative data binding can be applied, this reveals a major limitation of GT4.

In our implementation, workaround has been performed by modifying the definition to use xs:anyType like the following:

```
<xs:element name="ItemConstraint" type="xs:anyType" />
```

Manual parsing are then applied on the message element to correctly handle item constraints. There is a potential benefit with this modification, however, that it allows the usage of alternative restriction models. An example of such restriction models includes those defined by JSDL [15].

A.6 TermCompositorType

The definition of TermCompositorType in WS-Agreement specification is based on `xs:choice maxOccurs=”unbounded”`:

```
<xs:complexType name="TermCompositorType">
  <xs:sequence>
    <xs:choice maxOccurs="unbounded">
      <xs:element name="ExactlyOne"
        type="wsag:TermCompositorType" />
      ...
      <xs:element name="GuaranteeTerm"
        type="wsag:GuaranteeTermType" />
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

However, GT4 is not able to handle such elements. As a workaround to this issue, the TermCompositorType definition has been slightly modified to avoid the usage of `xs:choice maxOccurs=”unbounded”` as the following:

```
<xs:complexType name="TermCompositorType">
  <xs:sequence>
    <xs:element name="ExactlyOne"
      type="wsag:TermCompositorType"
      minOccurs="0" maxOccurs="unbounded" />
    ...
    <xs:element name="GuaranteeTerm"
      type="wsag:GuaranteeTermType"
      minOccurs="0" maxOccurs="unbounded" />
  </xs:sequence>
</xs:complexType>
```

This modification keeps the essences of the original definition and allows GT4 to process TermCompositorType elements. There is however a minor difference that the later requires the elements to appear in certain sequences. This can be considered to be a workaround before GT4 is upgraded to the newest version of Axis.

Appendix B

WS-Agreement Samples for Data Mining Service

B.1 WS-Agreement Template

```
<wsag:Template
  xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
  wsag:TemplateId="DataminingServiceAgreementTemplate">
  <wsag:Name>Datamining Service Agreement Template</wsag:Name>
  <wsag:Context xsi:type="wsag:Context">
    <wsag:AgreementResponder>
      http://grid:9090/wsrf/services/DataminingService
    </wsag:AgreementResponder>
    <wsag:ServiceProvider>
      AgreementResponder
    </wsag:ServiceProvider>
    <wsag:ExpirationTime>
      2007-05-09T13:50:04.719Z
    </wsag:ExpirationTime>
    <wsag:TemplateId>
      DataminingServiceAgreementTemplate
    </wsag:TemplateId>
    <wsag:TemplateName>
      Datamining Service Agreement Template
    </wsag:TemplateName>
  </wsag:Context>
  <wsag:Terms>
    <wsag:All>
      <wsag:ServiceReference
        wsag:Name="Datamining Service Reference"
        wsag:ServiceName="DataminingService"
        xsi:type="wsag:ServiceReference">
        <xsd:anyURI>
          http://grid:9090/wsrf/services/DataminingService
        </xsd:anyURI>
      </wsag:ServiceReference>
```

```
      <wsag:ServiceProperties wsag:Name="deadline"
        wsag:ServiceName="DataminingService">
        <wsag:VariableSet>
          <wsag:Variable wsag:Metric="job:Second"
            wsag:Name="endTime">
            <wsag:Location />
          </wsag:Variable>
        </wsag:VariableSet>
      </wsag:ServiceProperties>
    </wsag:All>
  </wsag:Terms>
  <wsag:CreationConstraints>
    <wsag:Item wsag:Name="Penalty Value Constraint">
      <wsag:Location>
        //wsag:GuaranteeTerm/wsag:BusinessValueList
        /wsag:Penalty/wsag:ValueExpression
      </wsag:Location>
      <wsag:ItemConstraint>
        <svrl:LowerBoundedRange
          xmlns:svrl="http://lrr.in.tum.de/arm/2006/10/svrl"
          exclusiveBound="false">
          0
        </svrl:LowerBoundedRange>
      </wsag:ItemConstraint>
    </wsag:Item>
    <wsag:Item wsag:Name="Penalty Unit Constraint">
      <wsag:Location>
        //wsag:GuaranteeTerm/wsag:BusinessValueList
        /wsag:Penalty/wsag:ValueUnit
      </wsag:Location>
      <wsag:ItemConstraint>
        <svrl:Exact
          xmlns:svrl="http://lrr.in.tum.de/arm/2006/10/svrl">
          0
        </svrl:Exact>
      </wsag:ItemConstraint>
    </wsag:Item>
    <wsag:Item wsag:Name="Reward Value Constraint">
      <wsag:Location>
        //wsag:GuaranteeTerm/wsag:BusinessValueList
        /wsag:Reward/wsag:ValueExpression
      </wsag:Location>
      <wsag:ItemConstraint>
        <svrl:LowerBoundedRange
          xmlns:svrl="http://lrr.in.tum.de/arm/2006/10/svrl"
          exclusiveBound="false">
```

```
          0
        </svrl:LowerBoundedRange>
      </wsag:ItemConstraint>
    </wsag:Item>
    <wsag:Item wsag:Name="Reward Unit Constraint">
      <wsag:Location>
        //wsag:GuaranteeTerm/wsag:BusinessValueList
        /wsag:Reward/wsag:ValueExpression
      </wsag:Location>
      <wsag:ItemConstraint>
        <svrl:Exact
          xmlns:svrl="http://lrr.in.tum.de/arm/2006/10/svrl">
          0
        </svrl:Exact>
      </wsag:ItemConstraint>
    </wsag:Item>
    <wsag:Item wsag:Name="Deadline Constraint">
      <wsag:Location>
        //wsag:GuaranteeTerm/wsag:ServiceLevelObjective
        /wsag:KPITarget/wsag:CustomServiceLevel
      </wsag:Location>
      <wsag:ItemConstraint xsi:nil="true" />
    </wsag:Item>
    <wsag:Item wsag:Name="Argument Constraint">
      <wsag:Location>
        //wsag:ServiceDescriptionTerm/job:job/job:arguments
      </wsag:Location>
      <wsag:ItemConstraint xsi:nil="true" />
    </wsag:Item>
  </wsag:CreationConstraints>
</wsag:Template>
```

B.2 WS-Agreement Offer

```
<AgreementOffer
  xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
  wsag:TemplateId="DataminingServiceAgreementTemplate"
  xsi:type="wsag:AgreementTemplateType">
  <Name>Datamining Service Agreement Template</Name>
  <Context>
    <AgreementResponder>
      http://grid:9090/wsrf/services/DataminingService
    </AgreementResponder>
    <ServiceProvider>AgreementResponder</ServiceProvider>
```

```
    <ExpirationTime>2007-05-09T13:50:04.719Z</ExpirationTime>
    <TemplateId>DataminingServiceAgreementTemplate</TemplateId>
    <TemplateName>
      Datamining Service Agreement Template
    </TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm
        wsag:Name="Datamining Service Description Term"
        wsag:ServiceName="DataminingService">
        <ns1:DataminingJobParameters
          xmlns:ns1="http://sog.lrr.in.tum.de/datamining
          /DataminingService/parameters">
          <ns1:dataminingEPR>
            <wsa:Address>
              http://grid:9090/wsrf/services/datamining
              /DataminingService
            </wsa:Address>
            <wsa:ReferenceProperties>
              <wsag:DataminingResourceKey
                xmlns:wsag="http://sog.lrr.in.tum.de
                /datamining/DataminingService">
                a84222f0-fe35-11db-ac4a-f38050f1049c
              </wsag:DataminingResourceKey>
            </wsa:ReferenceProperties>
            <wsa:ReferenceParameters />
          </ns1:dataminingEPR>
          <ns1:variableValues>SAMPLE_TABLE</ns1:variableValues>
          <ns1:variableValues>0.1</ns1:variableValues>
          <ns1:variableValues>0.5</ns1:variableValues>
        </ns1:DataminingJobParameters>
      </ServiceDescriptionTerm>
      <ExactOne>
        <GuaranteeTerm wsag:Name="MaxEndTime"
          wsag:Obligated="ServiceProvider">
          <ServiceScope wsag:ServiceName="DataminingService" />
          <ServiceLevelObjective>
            <KPITarget>
              <KPIName>endTime</KPIName>
              <CustomServiceLevel xsi:type="xsd:dateTime">
                2007-05-09T15:00:35.063Z
              </CustomServiceLevel>
            </KPITarget>
          </ServiceLevelObjective>
          <BusinessValueList>
```

```
          <Penalty>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>100</ValueExpression>
          </Penalty>
          <Reward>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>40</ValueExpression>
          </Reward>
        </BusinessValueList>
      </GuaranteeTerm>
      <GuaranteeTerm wsag:Name="MaxEndTime"
        wsag:Obligated="ServiceProvider">
        <ServiceScope wsag:ServiceName="DataminingService" />
        <ServiceLevelObjective>
          <KPITarget>
            <KPIName>endTime</KPIName>
            <CustomServiceLevel xsi:type="xsd:dateTime">
              2007-05-09T10:00:35.063Z
            </CustomServiceLevel>
          </KPITarget>
        </ServiceLevelObjective>
        <BusinessValueList>
          <Penalty>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>1000</ValueExpression>
          </Penalty>
          <Reward>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>400</ValueExpression>
          </Reward>
        </BusinessValueList>
      </GuaranteeTerm>
    </ExactOne>
  </All>
```

```
  </Terms>
  <CreationConstraints />
</AgreementOffer>
```

B.3 WS-Agreement

```
<Agreement
  xmlns:wsag="http://schemas.ggf.org/graap/2007/03/ws-agreement"
  wsag:TemplateId="DataminingServiceAgreementTemplate"
  xsi:type="wsag:AgreementTemplateType">
  <Name>Datamining Service Agreement Template</Name>
  <Context>
    <AgreementResponder>
      http://grid:9090/wsrf/services/DataminingService
    </AgreementResponder>
    <ServiceProvider>AgreementResponder</ServiceProvider>
    <ExpirationTime>2007-05-09T13:50:04.719Z</ExpirationTime>
    <TemplateId>DataminingServiceAgreementTemplate</TemplateId>
    <TemplateName>
      Datamining Service Agreement Template
    </TemplateName>
  </Context>
  <Terms>
    <All>
      <ServiceDescriptionTerm
        wsag:Name="Datamining Service Description Term"
        wsag:ServiceName="DataminingService">
        <ns1:DataminingJobParameters
          xmlns:ns1="http://sog.lrr.in.tum.de/datamining
          /DataminingService/parameters">
          <ns1:dataminingEPR>
            <wsa:Address>
              http://grid:9090/wsrf/services/datamining
              /DataminingService
            </wsa:Address>
            <wsa:ReferenceProperties>
              <wsag:DataminingResourceKey
                xmlns:wsag="http://sog.lrr.in.tum.de/datamining/
                DataminingService">
                a84222f0-fe35-11db-ac4a-f38050f1049c
              </wsag:DataminingResourceKey>
            </wsa:ReferenceProperties>
            <wsa:ReferenceParameters />
          </ns1:dataminingEPR>
          <ns1:variableValues>SAMPLE_TABLE</ns1:variableValues>
```

```
          <ns1:variableValues>0.1</ns1:variableValues>
          <ns1:variableValues>0.5</ns1:variableValues>
        </ns1:DataminingJobParameters>
      </ServiceDescriptionTerm>
      <GuaranteeTerm wsag:Name="MaxEndTime"
        wsag:Obligated="ServiceProvider">
        <ServiceScope wsag:ServiceName="DataminingService" />
        <ServiceLevelObjective>
          <KPITarget>
            <KPIName>endTime</KPIName>
            <CustomServiceLevel xsi:type="xsd:dateTime">
              2007-05-09T15:00:35.063Z
            </CustomServiceLevel>
          </KPITarget>
        </ServiceLevelObjective>
        <BusinessValueList>
          <Penalty>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>100</ValueExpression>
          </Penalty>
          <Reward>
            <AssessmentInterval>
              <Count>1</Count>
            </AssessmentInterval>
            <ValueUnit>GUVC</ValueUnit>
            <ValueExpression>40</ValueExpression>
          </Reward>
        </BusinessValueList>
      </GuaranteeTerm>
    </All>
  </Terms>
  <CreationConstraints />
</Agreement>
```

Abbreviations

A

ACT	Application Characterisation Tool
ANN	Artificial Neural Network
ARM4LSA	Automated Resource Management for Large-scale Applications

C

CHIP^{3}S	Characterisation Instrumentation for Performance Prediction of Parallel Systems
CPU	Central Processing Unit
CSF	Community Scheduler Framework

D

DNS	Domain Name Service
DSDP/TM	Device Software Development Platform / Target Management

E

EMF	Eclipse Modeling Framework
EMFT	Eclipse Modeling Framework Technology
EPR	Endpoint Reference

F

FIFO	First–In First–Out

G

GGF	Global Grid Form (see Open Grid Forum)
GRAAP	Grid Resource Allocation Agreement Protocol
GRAM	Grid Resource Allocation Management
GRM	Grid Resource Manager
GRMS	Grid Resource Management System
GSDE	Grid Service Development Environment
GSEC	Grid Service Execution Client
GSME	Grid Service Management Environment
GT	Globus Toolkit
GT2	Globus Toolkit version 2
GT4	Globus Toolkit version 4

H

HP	Hewlett Packard
HPC	High Performance Computing
HPF	High Performance Fortran

I

IBM	International Business Machines
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers

J

JDK	Java™ Development Kit
JDT	(Eclipse) Java Development Tools
JET	Java Emitting Templates
JMX	Java Management Extensions
JSDL	Job Submission Description Language

L

kNN	k-Nearest Neighbour

L

LAN	Local Area Network

M

MPI	Message Passing Interface
MPP	Massively Parallel Processor
MuSE	Multithreaded Scheduling Environment

N

NCName	"Non-Colonized" Names
NoW	Network of Workstations
NPB	NAS Parallel Benchmarks

O

OS	Operating System
OGF	Open Grid Forum
OGSA	Open Grid Service Architecture
OGSI	Open Grid Services Infrastructure

P

PACE	Performance Analysis and Characterisation Environment
PC	Processor Consistency *or* Personal Computer
PET	Positron Emission Tomography
PMB	Pallas MPI Benchmarks
PMML	Predictive Model Markup Language
PoP	Pile of PCs
POSIX	Portable Operating System Interface for uniX
PUNCH	Purdue University Network-Computing Hubs
PVM	Parallel Virtual Machine
PVP	Parallel Vector Processor

Q

QoS	Quality of Service

R

RAL	Rutherford Appleton Laboratories
RAM	Random Access Memory
RCP	Rich Client Platform
RPC	Remote Procedure Call

S

SLA	Service Level Agreement
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOG	Service Oriented Grid
SPMD	Single Program Multiple Data
SpMV	Sparse Matrix-Vector Multiply
SpMM	Sparse Matrix-Matrix Multiply
SQL	Structured Query Language
SUIF	Stanford University Intermediate Format

U

URI	Uniform Resource Identifier
URL	Uniform Resource Locator

W

WSDL	Web Service Description Language
WSMN	Web Service Management Language
WSMN	Web Services Management Network
WSOI	Web Services Offering Infrastructure
WSOL	Web Services Offering Language
WSRF	Web Service Resource Framework
WS-Addressing	Web Service Addressing
WS-Agreement	Web Service Agreement
WS-BaseFaults	Web Services Base Faults
WS-Notification	Web Services Notification
WS-Resource	Web Services Resource
WS-ResourceProperties	Web Services Resource Properties
WS-ResourceLifetime	Web Services Resource Lifetime
WS-Security	Web Services Security
WMI	Windows Management Interface

X

XML	Extensible Markup Language
XSD	XML Schema Definition
XPath	XML Path

Bibliography

[1] Community Scheduler Framework (CSF). Web site http://www.globus.org/grid_software/computation/csf.php.

[2] OASIS Web Services Resource Framework (WSRF). OASIS Web page http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[3] Pallas MPI Benchmarks. Pallas GmbH http://www.pallas.com/e/produces/pmb/.

[4] XML Path. OASIS Web page http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[5] Web Services Service Group 1.2 (WS-ServiceGroup). Online available at http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ServiceGroup-1.2-draft-02.pdf, June 2004.

[6] TrustCoM Basic Reference Implementation, Deliverable D19. Online available at http://212.0.127.139/trustcom/wp-content/uploads/2007/08/d19+basic+.pdf, September 2005.

[7] Mona Aggarwal, Robert D. Kent, and Alioune Ngom. Genetic Algorithm Based Scheduler for Computational Grids. In *HPCS*, pages 209–215. IEEE Computer Society, 2005.

[8] C. G. Akteson, S.A. Schaal, and A. W. Moore. Locally Weighted Learning. *AI Review*, 11,:11–73, 1997.

[9] Kurt Thearling Alex Berson. *Building Data Mining Applications for CRM*. McGraw-Hill, Inc., New York, NY, 1999.

[10] A. M. Alkindi, Darren J. Kerbyson, and Graham R. Nudd. Dynamic Instrumentation and Performance Prediction of Application Execution. In Louis O. Hertzberger, Alfons G. Hoekstra, and Roy Williams, editors, *HPCN Europe*, volume 2110 of *Lecture Notes in Computer Science*, pages 513–523. Springer, 2001.

[11] Gabrielle Allen, Tom Goodale, Gerd Lanfermann, Thomas Radke, Edward Seidel, Werner Benger, Hans-Christian Hege, André Merzky, Joan Massó, and John Shalf.

Solving Einstein's Equations on Supercomputers. *IEEE Computer*, 32(12):52–58, 1999.

[12] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in Public-resource Computing. *Communication of ACM*, 45(11):56–61, 2002.

[13] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Open Grid Forum, http://www.gridforum.org/documents/GFD.107.pdf, 2007.

[14] Artur Andrzejak, Sven Graupner, and Stefan Plantikow. Predicting Resource Demand in Dynamic Utility Computing Environments. In *ICAS*, page 6. IEEE Computer Society, 2006.

[15] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Andreas Savva. Job Submission Description Language (JSDL) Specification. Online available at http://www.gridforum.org/documents/GFD.56.pdf, November 2005.

[16] R. Aydt. The Pablo Self-Defining Data Format. University of Illinois at Urbana-Champaign, Department of Computer Science, 1992, latest revision 2003.

[17] Rosa M. Badia, Jess Labarta, Judit Gimenez, and Francesc Escale. DIMEMAS: Predicting MPI Applications Behavior in Grid Environments. Workshop on Grid Applications and Programming Tools (GGF8), www.cepba.upc.es/dimemas/docs/dimemas_updt.pdf, 2003.

[18] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS Parallel Benchmarks. *The International Journal of Supercomputer Applications*, 5(3):63–73, Fall 1991.

[19] Kevin J. Barker, Scott Pakin, and Darren J. Kerbyson. A Performance Model of the Krak Hydrodynamics Application. In *ICPP*, pages 245–254. IEEE Computer Society, 2006.

[20] Dominic Battré. On the Use of Service Level Agreements in AssessGrid. OGF20, May 7-11, 2007, Manchester, UK, 2007.

[21] Antonia Bertolino, Guglielmo De Angelis, and Andrea Polini. Automatic Generation of Test-beds for Pre-deployment QoS Evaluation of Web Services. In *WOSP'07: Proceedings of the 6th international workshop on software and performance*, pages 137–140, New York, NY, USA, 2007. ACM Press.

[22] Ponsy R. K. Sathia Bhama, Thamarai Selvi Soma Sundaram, Supriya Vasudevan, P. Al Niyas, and K. Swadha. Scheduling of Jobs in a Dynamic Heterogeneous Environment. In Hamid R. Arabnia, editor, *GCA*, pages 97–106. CSREA Press, 2006.

[23] Georg Birkenheuer, Karim Djemame, Iain Gourlay, Odej Kao, James Padgett, and Kerstin VoSS. Using WS-Agreement for Risk Management in the Grid. First WS-Agreement Workshop (OGF18), Washington, September 2006.

[24] Ian Bowman, John Shalf, Kwan-Liu Ma, and Wes Bethel. Performance Modeling for 3D Visualization in a Heterogeneous Computing Environment. Lawrence Berkeley National Laboratory. Paper LBNL-56977. http://repositories.cdlib.org/lbnl/LBNL-56977, June 2004.

[25] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra A. Hensgen, and Richard F. Freund. A Comparison Study of Static Mapping Heuristics for a Class of Meta-tasks on Heterogeneous Computing Systems. In *Heterogeneous Computing Workshop*, pages 15–29, 1999.

[26] Tracy D. Braun, Howard Jay Siegel, Noah Beck, Ladislau Bölöni, Muthucumaru Maheswaran, Albert I. Reuther, James P. Robertson, Mitchell D. Theys, Bin Yao, Debra A. Hensgen, and Richard F. Freund. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *J. Parallel Distrib. Comput.*, 61(6):810–837, 2001.

[27] Leo Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth, 1984.

[28] Frank Budinsky, Dave Steinberg, Ed Merks, Ray Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, 2003.

[29] D. Burton. Multidimensional Discretization of Conservation Laws of Unstructured Polyhedral Grids. In *Proc. 2nd International Workshop on Analytical Methods and Process Optimization in Fluid and Gas Mechanics*, 1994.

[30] Rajkumar Buyya. Grid Economy Comes of Age: Emerging Gridbus Tools for Service-Oriented Cluster and Grid Computing. In Ross Lee Graham and Nahid Shahmehri, editors, *Peer-to-Peer Computing*, page 13. IEEE Computer Society, 2002.

[31] Steve J. Chapin, Dimitrios Katramatos, John F. Karpovich, and Andrew S. Grimshaw. The Legion Resource Management System. In Feitelson and Rudolph [45], pages 162–178.

[32] M. Copelli, R. Eichorn, O. Kinouchi, M. Biehl, R. Simonetti, P. Riegler, and N. Caticha. Noise Robustness in Multilayer Neural Networks, 1997.

[33] K. Czajkowski, I. Foster, and C. Kesselman. Agreement-Based Resource Management. *Proceedings of the IEEE*, 93(3):631–643, March 2005.

[34] Karl Czajkowski, Ian T. Foster, Nicholas T. Karonis, Carl Kesselman, Stuart Martin, Warren Smith, and Steven Tuecke. A Resource Management Architecture for Metacomputing Systems. In Dror G. Feitelson and Larry Rudolph, editors, *JSSPP*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82. Springer, 1998.

[35] Thierry Delaitre, Tamás Kiss, Ariel Goyeneche, Gábor Terstyánszky, Stephen C. Winter, and Péter Kacsuk. GEMLCA: Running Legacy Code Applications as Grid Services. *J. Grid Comput.*, 3(1-2):75–90, 2005.

[36] Erik D. Demaine. Efficient Simulation of Message-Passing in Distributed-Memory Architectures. Master's thesis, Department of Computer Science, University of Waterloo, 1996.

[37] K. Deng and A. W. Moore. Multiresolution Instance-based Learning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Montreal, Canada, 1995.

[38] Murthy V. Devarakonda and Ravishankar K. Iyer. Predictability of Process Resource Usage: A Measurement-Based Study on UNIX. *IEEE Trans. Software Eng.*, 15(12):1579–1586, 1989.

[39] Dietmar W. Erwin. UNICORE - a Grid computing environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[40] Thomas Fahringer, Alexandru Jugravu, Sabri Pllana, Radu Prodan, Clovis Seragiotto Jr., and Hong Linh Truong. ASKALON: a Tool Set for Cluster and Grid Computing. *Concurrency - Practice and Experience*, 17(2-4):143–169, 2005.

[41] Robert D. Falgout and Ulrike Meier Yang. hypre: A Library of High Performance Preconditioners. In Peter M. A. Sloot, Chih Jeng Kenneth Tan, Jack Dongarra, and Alfons G. Hoekstra, editors, *International Conference on Computational Science (3)*, volume 2331 of *Lecture Notes in Computer Science*, pages 632–641. Springer, 2002.

[42] Usama M. Fayyad and Keki B. Irani. On the Handling of Continuous-Valued Attributes in Decision Tree Generation. *Machine Learning*, 8:87–102, 1992.

[43] Jane Fedorowicz. Database Performance Evaluation in an Indexed File Environment. *ACM Trans. Database Syst.*, 12(1):85–110, 1987.

[44] Dror Feitelson. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, 2006.

[45] Dror G. Feitelson and Larry Rudolph, editors. *Job Scheduling Strategies for Parallel Processing, IPPS/SPDP'99 Workshop, JSSPP'99, San Juan, Puerto Rico, April 16, 1999, Proceedings*, volume 1659 of *Lecture Notes in Computer Science*. Springer, 1999.

[46] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[47] Ian T. Foster. The Globus Toolkit for Grid Computing. In *CCGRID*, page 2. IEEE Computer Society, 2001.

[48] Ian T. Foster, Jonathan Geisler, Bill Nickless, Warren Smith, and Steven Tuecke. Software Infrastructure for the I-WAY Metacomputing Experiment. *Concurrency - Practice and Experience*, 10(7):567–581, 1998.

[49] Ian T. Foster, Nicholas R. Jennings, and Carl Kesselman. Brain meets brawn: Why grid and agents need each other. In *AAMAS*, pages 8–15. IEEE Computer Society, 2004.

[50] James Frey, Todd Tannenbaum, Miron Livny, Ian T. Foster, and Steven Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. *Cluster Computing*, 5(3):237–246, 2002.

[51] Yang Gao, Hongqiang Rong, and Joshua Zhexue Huang. Adaptive Grid Job Scheduling with Genetic Algorithms. *Future Generation Comp. Syst.*, 21(1):151–161, 2005.

[52] Jonathan Geisler and Valerie E. Taylor. Performance Coupling: A Methodology for Predicting Application Performance Using Kernel Performance. In *PPSC*, 1999.

[53] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A Users' Guide to PICL. Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, Tennessee, USA, 1990.

[54] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. PICL: A Portable Instrumented Communication Library, Reference Manual. Technical Report ORNL/TM-11130, Oak Ridge National Laboratory, Tennessee, USA, 1990.

[55] L. George, P. Muhlethaler, and N. Rivierre. Optimality and Non-Preemptive Real-Time Scheduling Revisited. Technical report, Rapport de Recherche RR-2516, INRIA, Le Chesnay Cedex, France, 1995.

[56] L. George, N. Rivierre, and M. Spuri. Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling. Technical report, Rapport de Recherche RR-2966, INRIA, Le Chesnay Cedex, France, 1996.

[57] Germán S. Goldszmidt and Jürgen Schönwälder, editors. *Integrated Network Management VII, Managing It All, IFIP/IEEE Eighth International Symposium on Integrated Network Management (IM 2003), March 24-28, 2003, Colorado Springs, USA*, volume 246 of *IFIP Conference Proceedings*. Kluwer, 2003.

[58] L. Grossman, S. Bailey, A. Ramu, B. Malhi, P. Hallstrom, I. Pulleyn, and X. Oin. The Management and Mining of Multiple Predictive Models Using the Predictive Modeling Markup Language (PMML). *Information and Software Technology*, 41:589–595, 1999.

[59] Dan Gunter and Brian Tierney. NetLogger: A Toolkit for Distributed System Performance Tuning and Debugging. In Goldszmidt and Schönwälder [57], pages 97–100.

[60] Mohammed H. Haji, Peter M. Dew, Karim Djemame, and Iain Gourlay. A SNAP-Based Community Resource Broker Using a Three-Phase Commit Protocol. In *IPDPS*. IEEE Computer Society, 2004.

[61] Adolfy Hoisie, Olaf M. Lubeck, and Harvey J. Wasserman. Performance and Scalability Analysis of Multidimensional Wavefront Algorithms on Teraflop-Scale Architectures. In *PPSC*, 1999.

[62] Fred Howell and Ross McNab. SimJava: a Discrete Event Simulation Package for Java with Applications in Computer Systems Modelling. In Society for Computer Simulation, editor, *Proc. First International Conference on Web-based Modelling and Simulation*, San Diego CA, Jan 1998.

[63] Jiandong Huang, Y. Wang, N. R. Vaidyanathan, and Feng Cao. GRMS: A Global Resource Management System for Distributed QoS and Criticality Support. In *ICMCS*, pages 424–432, 1997.

[64] Engin Ipek, Bronis R. de Supinski, Martin Schulz, and Sally A. McKee. An Approach to Performance Prediction for Parallel Applications. In José C. Cunha and Pedro D. Medeiros, editors, *Euro-Par*, volume 3648 of *Lecture Notes in Computer Science*, pages 196–205. Springer, 2005.

[65] Stephen A. Jarvis, Ligang He, Daniel P. Spooner, and Graham R. Nudd. The impact of predictive inaccuracies on execution scheduling. *Perform. Eval.*, 60(1-4):127–139, 2005.

[66] Péter Kacsuk, Gábor Dózsa, József Kovács, Róbert Lovas, Norbert Podhorszki, Zoltán Balaton, and Gabor Gombás. P-GRADE: A Grid Programming Environment. *J. Grid Comput.*, 1(2):171–197, 2003.

[67] Nirav H. Kapadia, Renato J. O. Figueiredo, and José A. B. Fortes. PUNCH: Web Portal for Running Tools. *IEEE Micro*, 20(3):38–47, 2000.

[68] Nirav H. Kapadia, José A. B. Fortes, and Carla E. Brodley. Predictive Application-Performance Modeling in a Computational Grid Environment. In *HPDC*, 1999.

[69] Darren J. Kerbyson, Henry J. Alme, Adolfy Hoisie, Fabrizio Petrini, Harvey J. Wasserman, and M. Gittings. Predictive Performance and Scalability Modeling of a Large-scale Application. In *SC*, page 37, 2001.

[70] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. SLAng: A Language for Defining Service Level Agreements. In *FTDCS*, pages 100–. IEEE Computer Society, 2003.

[71] M. Lavy and A. Meggitt. *Windows Management Instrumentation (WMI)*. Sams, 2001.

[72] Benjamin C. Lee, Richard W. Vuduc, James Demmel, and Katherine A. Yelick. Performance Models for Evaluation and Automatic Tuning of Symmetric Sparse Matrix-Vector Multiply. In *ICPP*, pages 169–176. IEEE Computer Society, 2004.

[73] Hui Li, Juan Chen, Ying Tao, David Gro, and Lex Wolters. Improving a Local Learning Technique for QueueWait Time Predictions. In *CCGRID*, pages 335–342. IEEE Computer Society, 2006.

[74] Tianchao Li and Michael Gerndt. A Generic Extensible WS-Agreement Implementation for Globus Toolkit 4. Technical report, Institut fuer Informatik, Technische Universitaet Muenchen, 2006.

[75] C. L. Liu and Layland. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment. *J. Assoc. Compu.*, 20:46-61, 1973.

[76] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreents. In Marco Aiello, Mikio Aoyama, Francisco Curbera, and Mike P. Papazoglou, editors, *ICSOC*, pages 65–74. ACM, 2004.

[77] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck. A Service Level Agreement Language for Dynamic Electronic Services. *Electronic Commerce Research*, 3(1-2):43–59, 2003.

[78] Heiko Ludwig, Alexander Keller, Asit Dan, Richard P. King, and Richard Franck. Web Service Level Agreement (WSLA) Language Specification, Version 1.0. IBM Research Web page, http://www.research.ibm.com/wsla/documents.html, 2003.

[79] Heiko Ludwig, Toshiyuki Nakata, Oliver Wäldrich, Philipp Wieder, and Wolfgang Ziegler. Reliable Orchestration of Resources Using WS-Agreement. In Michael Gerndt and Dieter Kranzlmüller, editors, *HPCC*, volume 4208 of *Lecture Notes in Computer Science*, pages 753–762. Springer, 2006.

[80] Vijay Machiraju, Akhil Sahai, and Aad P. A. van Moorsel. Web Services Management Network: An Overlay Network for Federated Service Management. In Goldszmidt and Schönwälder [57], pages 351–364.

[81] A. A. Markov. Extension of the Limit Theorems of Probability Theory to a Sum of Variables Connected in a Chain. reprinted in Appendix B of: R. Howard. Dynamic Probabilistic Systems, volume 1: Markov Chains. John Wiley and Sons, 1971.

[82] Matthew L. Massie, Brent N. Chun, and David E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(5-6):817–840, 2004.

[83] Mark M. Mathis and Darren J. Kerbyson. A General Performance Model of Structured and Unstructured Mesh Particle Transport Computations. *The Journal of Supercomputing*, 34(2):181–199, 2005.

[84] Mark M. Mathis, Darren J. Kerbyson, and Adolfy Hoisie. A Performance Model of Non-Deterministic Particle Transport on Large-scale Systems. *Future Generation Comp. Syst.*, 22(3):324–335, 2006.

[85] Carlos Molina-Jimenez, Jim Pruyne, and Aad van Moorse. Software Architectures for Service Level Agreements and Contracts. Online available at http://tapas.sourceforge.net/deliverables/D6/Appendix2.pdf.

[86] Ruben S. Montero, Eduardo Huedo, and Ignacio M. Llorente. Grid Scheduling Infrastructures Based On the GridWay Meta-scheduler. *TCSC Newsletter*, 8(2), 2006.

[87] Michelle Moore. An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster. In *IPDPS*, page 145. IEEE Computer Society, 2003.

[88] Michelle Moore. An Accurate Parallel Genetic Algorithm to Schedule Tasks on a Cluster. *Parallel Computing*, 30(5-6):567–583, 2004.

[89] P. Kacsuk N. Podhorszki. Design and Implementation of a Distributed Monitor for Semi-on-line Monitoring of VisualMP Applications. In *Proceedings. DAPSYSŠ2000 Distributed and Parallel Systems, From Instruction Parallelism to Cluster Computing*, pages 23–32, Balatonfüred, Hungary, 2000.

[90] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE - A Toolset for the Performance Prediction of Parallel and Distributed Systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, Fall 2000.

[91] Tuomas Nurmela. Web Services Level Management: Overview of Service Level Agreement Languages and Support Infrastructures, 2006.

[92] OSGi Alliance. OSGi Service Platform Core Specification Release 4. http://www.osgi.org, August 2005.

[93] Andrew J. Page and Thomas J. Naughton. Dynamic Task Scheduling using Genetic Algorithms for Heterogeneous Distributed Computing. In *IPDPS*. IEEE Computer Society, 2005.

[94] E. Papaefstathiou, Darren J. Kerbyson, Graham R. Nudd, and T. J. Atherton. An Introduction to the CHIP3S Language for Characterising Parallel Systems in Performance Studies. Research Report CS-RR-280, Department of Computer Science, University of Warwick, Coventry, UK, January 1995.

[95] Steve Perry. *Java Management Extensions*. O'Reilly, 2002.

[96] D. A. Reed, R. A. Aydt, T. M. Madhyastha, R. J. Noe, K. A. Shields, and B. W. Schwartz. The Pablo Performance Analysis Environment. Dept. of Comp. Sci., Univ. of IL, 1992.

[97] Matei Ripeanu, Adriana Iamnitchi, and Ian T. Foster. Cactus Application: Performance Predictions in Grid Environments. In Rizos Sakellariou, John Keane, John R. Gurd, and Len Freeman, editors, *Euro-Par*, volume 2150 of *Lecture Notes in Computer Science*, pages 807–816. Springer, 2001.

[98] B. D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, Cambridge, UK, 1996.

[99] A. Sahai, A. Durante, and V. Machiraju. Towards Automated SLA Management for Web Services, 2001.

[100] Akhil Sahai, Vijay Machiraju, Mehmet Sayal, Aad P. A. van Moorsel, and Fabio Casati. Automated SLA Monitoring for Web Services. In Metin Feridun, Peter G. Kropf, and Gilbert Babin, editors, *DSOM*, volume 2506 of *Lecture Notes in Computer Science*, pages 28–41. Springer, 2002.

[101] Jennifer Schopf. Structural Prediction Models for High-Performance Distributed Applications, February 1997.

[102] Jennifer M. Schopf and Francine Berman. Performance Prediction in Production Environments. In *IPPS/SPDP*, pages 647–653, 1998.

[103] Jennifer M. Schopf, Mike D'Arcy, Neill Miller, Laura Pearlman, Ian Foster, and Carl Kesselman. Monitoring and Discovery in a Web Services Framework: Functionality and Performance of the Globus Toolkit's MDS4. Technical report, Tech Report ANL/MCS-P1248-0405, 2005.

[104] Jennifer Melinda Schopf. Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters, January 1998.

[105] Mumtaz Siddiqui and Thomas Fahringer. GridARM: Askalon's Grid Resource Management System. In Peter M. A. Sloot, Alfons G. Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *EGC*, volume 3470 of *Lecture Notes in Computer Science*, pages 122–131. Springer, 2005.

[106] James Skene, D. Davide Lamanna, and Wolfgang Emmerich. Precise Service Level Agreements. In *ICSE*, pages 179–188. IEEE Computer Society, 2004.

[107] Warren Smith, Valerie E. Taylor, and Ian T. Foster. Using Run-Time Predictions to Estimate Queue Wait Times and Improve Scheduler Performance. In Feitelson and Rudolph [45], pages 202–219.

[108] Allan Snavely, Laura Carrington, Nicole Wolter, Jesús Labarta, Rosa M. Badia, and Avi Purkayastha. A Framework for Performance Modeling and Prediction. In *SC*, pages 1–17, 2002.

[109] Allan Snavely, Xiaofeng Gao, C. Lee, Laura Carrington, Nicole Wolter, Jesús Labarta, Judit Gimenez, and P. Jones. Performance Modeling of HPC Applications. In Gerhard R. Joubert, Wolfgang E. Nagel, Frans J. Peters, and Wolfgang V. Walter, editors, *PARCO*, volume 13 of *Advances in Parallel Computing*, pages 777–784. Elsevier, 2003.

[110] Allan Snavely, Nicole Wolter, and Laura Carrington. Modeling Application Performance by Convolving Machine Signatures with Application Profiles, November 2001.

[111] DP Spooner, GR Nudd, J Cao, and S Saini. Local Grid Scheduling Techniques using Performance Prediction. *IEEE Proc. - Comp. Digit. Tech.*, 250(2):87–96, 2003.

[112] Vladimir Tosic. *Service Offerings for XML Web Services and Their Management Applications*. PhD thesis, Carleton University, Ottawa, Ont., Canada, Canada, 2004. Adviser-Bernard Pagurek.

[113] Vladimir Tosic, Bernard Pagurek, Kruti Patel, Babak Esfandiari, and Wei Ma. Management applications of the web service offerings language (WSOL). *Inf. Syst.*, 30(7):564–586, 2005.

[114] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. WSOL - Web Service Offerings Language. In Christoph Bussler, Richard Hull, Sheila A. McIlraith, Maria E. Orlowska, Barbara Pernici, and Jian Yang, editors, *WES*, volume 2512 of *Lecture Notes in Computer Science*, pages 57–67. Springer, 2002.

[115] Sergi Girona Turell. *Performance Prediction and Evaluation Tools*. PhD thesis, Department of Computer Architecture, Polytechnic University of Catalunya, 2003.

[116] J. S. Vetter and D. A. Reed. Real-time Monitoring Adaptive Control and Interactive Steering of Computational Grids. *The International Journal of High Performance Computing Applications*, 2000.

[117] Rich Vuduc, James Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin Lee. Performance Optimizations and Bounds for Sparse Matrix-Vector Multiply. In *SC*, pages 1–35, 2002.

[118] Rich Vuduc, Attila Gyulassy, James Demmel, and Katherine A. Yelick. Memory Hierarchy Optimizations and Performance Bounds for Sparse A. In Peter M. A. Sloot, David Abramson, Alexander V. Bogdanov, Jack Dongarra, Albert Y. Zomaya, and Yuri E. Gorbachev, editors, *International Conference on Computational Science*, volume 2659 of *Lecture Notes in Computer Science*, pages 705–714. Springer, 2003.

[119] Erik Wilde. Web and XML Glossary. Online available at http://dret.net/glossary/.

[120] Michael Wilson, Alvaro Arenas, David Chadwick, Theo Dimitrakos, Jurgen Doser, Pablo Giambiagi, David Golby, Christian Geuer-Pollman, Jochen Haller, Stølen Ketil, and ... The TrustCoM Approach to Enforcing Agreements Between Interoperating Enterprises. In *Proc. Interoperability for Enterprise Software and Applications Conference (I-ESA'06)*, Bordeaux, France, March 2006.

[121] Robert P. Wilson, Robert S. French, Christopher S. Wilson, Saman P. Amarasinghe, Jennifer-Ann M. Anderson, Steven W. K. Tjiang, Shih-Wei Liao, Chau-Wen Tseng, Mary W. Hall, Monica S. Lam, and John L. Hennessy. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. *SIGPLAN Notices*, 29(12):31–37, 1994.

[122] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical Machine Learning Tools and Techniques with Java Implementations, 1999.

[123] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.

[124] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computer Systems*, 1999.

[125] Lingyun Yang, Jennifer M. Schopf, and Ian T. Foster. Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments. In *SC*, page 31. ACM, 2003.

[126] Y. Zhang and Y. Inoguchi. Influence of Inaccurate Performance Prediction on Task Scheduling in a Grid Environment. *IEICE Trans. Inf. Syst.*, E89-D(2):479–486, 2006.