Design and Implementation of a Service Discovery and Recommendation Architecture

by

Muhamed Sukkar

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

Abstract

Increasing number of software vendors are offering or planning to offer their applications as a Software-as-a-Service (SaaS) to leverage the benefits of cloud computing and Internet-based delivery. Therefore, potential clients will face increasing number of providers that satisfy their requirements to choose from. Consequently, there is an increasing demand for automating such a time-consuming and error-prone task. In this work, we develop an architecture for automated service discovery and selection in cloud computing environment. The system is based on an algorithm that recommends service choices to users based on both functional and non-functional characteristics of available services. The system also derives automated ratings from monitoring results of past service invocations to objectively detect badly-behaving providers. We demonstrate the effectiveness of our approach using an early prototype that was developed following object-oriented methodology and implemented using various open-source Java technologies and frameworks. The prototype uses a Chord DHT as its distributed backing store to achieve scalability.

Acknowledgements

All praises and thanks are due to Allah for giving me the ability, the patience, and the strength to complete this task. Thanks to Allah for facilitating my path to the right. May Allah accept this work and render it useful and beneficial for my faith and for my life.

To Prophet Mohammad, peace be upon him, the beloved, the master, the mercy sent to the human kind, who commended to seek knowledge anytime and anywhere; may Allah help us be good followers to his message.

I would like to express my appreciation to my supervisor, Professor Raouf Boutaba for his help and support throughout my Masters degree. I would also like to thank my readers, Professor Richard Trefler and Dr. Khuzaima Daudjee for their insightful comments and discussions.

This thesis would not have been possible without the assistance of many people, who I need pages and pages to list them all. I express my deepest gratitude and appreciation to Dr. Noura Limam for her support, guidance, and encouragement. She has helped me to explore the research area and provided me with invaluable feedback to keep my focus and to improve the quality of my research.

Many thanks are due to Professor Martin Karstin and Dr. Jin Xiao for their helpful comments and advice, especially at the beginning of my graduate life at Waterloo. I also thank Margaret Towell and the staff of the Computer Science Graduate Office for their help and support.

I am truly indebted to my father Hesham and my mother Lamaa, without their love, confidence, support and prayers, I would never succeed in life. I thank my grandfather Omar, my sisters, Salam, Raghad, Deemah, A'aisha and my young brother Abdurrahman for their constant love and support.

Special thanks are due to my father-in-law, Professor Abdallah El-Kettani, my mother-in-law Basema, and my brothers-in-law Jaafar, Ibrahim, Mohannad, Yahya, Nazeer and Abdulhakeem; their constant support, prayers, and encouragement was very helpful and motivating.

To my late grandfather, Sheikh Muhamed Sukkar, my late grandmother Najah, the late Mr. Mohammad Daadoush, I started with you but here I am without you at the end, may Allah bless and reward you.

I cannot forget to thank Professor Mohammed Sqalli of my former institute, King Fahd University of Petroleum and Minerals (KFUPM) for his encouragement and support to pursue my graduate studies here at Waterloo. To my dearest friends, Moataz, Abdulkareem, Ayman, Mahmoud, Hassan, Abdallah, Khaled, Salamah, Sinan, Obaida, Amer, Wael, Saleem and the rest of you all: you made my life at KFUPM and beyond lovely and enjoyable; I really miss you.

My little princess Lemya, your eyes are my guide to the future, your smile was my strongest motivation to reach the end of this road, may Allah bless and protect you.

My beloved wife Shifaa, where to begin and where to finish; truly without your sincere love, infinite patience, continuous encouragement, unfettered confidence, and huge support I would have never finished this journey. You are the best thing that has ever happened to me. I cannot thank you enough!

To my small world, Shifaa and Lemya

Contents

List of Figures

Chapter 1

Introduction

The Internet as we know it today is much different from what it was known 30 years ago. Back then, the main focus was just on moving bits and pieces of information from one location to another. It was a joy back then to receive a response from a ”pinged” server.

Over time, the Internet evolved with the invention of the Web to a distribution platform for content and media. It is the content available from worldwide locations that mattered to the end-users, and it became less and less important where the location of the content is. However, the content on the Internet grew so fast that it became so large for anyone to make any sense of it. Search engines started to make the scene more interesting as the largely disconnected islands of information suddenly became available with a few keystrokes of keywords.

The Internet today is going though a third wave of evolution. The fact that everyday content and media is available mainly on the Internet rather that on traditional means is no longer intriguing or special. In fact, we are suffering today from an information overload; simply, there is much more consumable information available than one has time for in a lifetime. Increasingly, we feel the need for better tools to make sense of this available information in a concise and efficient manner. We need more systems to automate tasks and functions that we increasingly see as cumbersome and time-consuming.

In the past decade, efforts were made to bring more structure to the information available on the Internet. Initiatives like Web Services, Semantic Web Technologies, Service Oriented Architectures among others are trying to better connect the isolated information systems available from individual organizations.

Today, applications are expected to be released more as online services on the web than as heavyweight desktop applications. There are several reasons for this trend. First, the ability to connect to the web is no longer restricted to PCs or even laptops;

smart phones and other limited-capability devices are quickly becoming the largest base of web access.

Second, everyone expects all the services they need to be available for them anytime and anywhere. Universal Internet access is a universal phenomenon. Moreover, legacy utility networks, such as mobile, landline, and even power networks are quickly converging to the universal data network and competing to be the access method for accessing the Internet; so called the unified communication vision.

Third, as users embrace wider choices of heterogeneous hardware platforms and operation systems, it is logical for application developers to write their software once, deploy it on their infrastructure, and offer it to anyone who needs it on the web instead of developing a separate version for each combination of user choices. Software maintenance and upgrade issues largely hide away from users and relieves them from a huge burden.

Not only applications targeted at individuals are offered online, but an increasingly popular trend is to offer larger scale services and complete infrastructures for whole companies or organizations. This trend is captured in the over hyped term of Cloud Computing. The basic idea is that not every company or person wants to run and maintain the computing infrastructure and supporting systems they need. This trend is often compared to the early 20th century when companies no longer maintained their power generation facilities and this responsibility was given to specialized power companies that offered this utility as a service to anyone in its network area.

As more services are offered in this manner, potential clients will be faced with more choices. It would be quickly a frustrating experience to search for those services and research them individually before making a decision to choose one of the potential providers and offerings. There would be no guarantee that clients choices are optimal for their own requirements. There is even the risk that a provider does not fulfill its promises and therefore possibly cause a financial loss to the client if a disconnected service was mission-critical to the business. This is especially important with the lack of trust in today's Internet; something that is still under intensive research but falls outside the scope of this work.

Therefore, we see a clear need for an online mediated marketplace-like for the emerging market of online services, or Software-as-a-Service (SaaS) as it is commonly known. In this thesis, we attempt to investigate this problem and offer our attempt to design a solution that will help match potential clients with the best providers that satisfy their requirements, while ensuring the trustworthiness of those providers in an automated and time-efficient manner.

1.1 Motivation

As the Internet becomes the main delivery channel for new applications and services, users will face the task of choosing the proper provider and offer for their requirements. The task of objectively researching, comparing and evaluating available choices manually is a daunting and an error-prone mission. Not doing the proper search may lead to undesired consequences and cost valuable time and money. While SaaS and cloud computing advocates ease of provisioning and configuration (e.g. getting a server up and running in terms of minutes rather than days), users still have to make the decision on which provider and which one of its offerings to work with. The problem is compounded when dealing with unknown and potentially untrustworthy providers.

One common technique for distinguishing trusted partners in online environments is the use of reputation systems. Reputation systems usually use feedbacks of previous users to help guide new ones. However, this subjective method is prone to many vulnerabilities such as Sybil attacks [16] and others. While many solutions in the literature have been proposed to tackle individual problems of reputation systems, an alternative is to avoid using human feedbacks altogether and instead rely on unbiased monitoring results to infer feedbacks and give users an objective and complete view.

However, such a methodology is not sufficient by itself. To be effective, monitoring results need to be compared to the service agreement between the provider and user that will define the obligations and penalties of each. The agreement, in turn, needs to describe unambiguously the service in question.

Moreover, the user needs a mechanism to search and discover such a service among the potentially hundreds or thousands of services. Users need a way to specify their requirements so they can later compare the matching offers using the objective method described above.

1.2 Objectives

This thesis aims to analyze and study the existing solutions to various service management problems that are related to the cloud environment and SaaS applications, and attempt to find a unifying theme among all of them. In particular, we attempt to design a service description scheme to represent service capabilities and quality attributes. We also focus on the storage and retrieval -advertisement and discovery- of service records. Another major objective is to design and integrate a solution for the service evaluation and recommendation problem based on service quality. While service and SLA monitoring systems are outside the scope of this work, it is our objective to facilitate the integration of these systems with our proposed platform to provide the information necessary for our core system.

1.3 Contributions

The main contribution of this thesis is that it presents a unified solution to the problem of obtaining recommendations of online SaaS services and cloud-based offers. This solution integrates four main components: (1) a service description scheme that covers both functional and non-functional aspects of a service in a Service Level Agreement, (2) a scalable service directory that allows flexibility to advertise and search using the description scheme, and (3) a service recommendation engine that responds to constraints to give user-specific rankings of service matches.

1.4 Organization

The rest of this work is organized as follows. Chapter 2 provides the necessary background for the technologies and concepts used throughout the remaining chapters. Chapter 3 provides a high-level overview of our proposed system including domain modeling, design requirements and usage scenarios. We then describe the specifications of our system design in Chapter 4 and how it is realized in our implemented prototype in Chapter 5. Chapter 6 summarizes our results and points out to potential future works.

Chapter 2

Background

2.1 Introduction

Developing an enhanced service directory platform like the one we propose in this work for SaaS applications involves solving a variety of different problems. For example, we need to define what a service is -in the new domain of cloud computing and SaaS applications-, and how it would be represented and described in an identifiable and expressive manner. Such description may also include quality information in the form of a Service Level Agreement (SLA). Then, such a service, or more precisely, its representative description, needs to be stored and cataloged by the system, raising issues such as where such description would come from, how it would be stored, and how it would be eventually found or *discovered*. When handling a collection of services of various providers, we need mechanisms for evaluating and comparing those services against each other to give the ultimate service user a hint about their *trustworthiness*, which is very critical when dealing with unknown online providers that may not even have a known physical location. The trustworthiness and quality assurance issue opens the door to problems like developing reliable *reputation* measures.

As mentioned earlier, each of those mentioned issues or problems is a research area on its own. Therefore, our goal in this chapter is to study and discuss the literature of those main problems and their proposed solutions. Our ultimate objective is to develop a unique approach to solve this composite problem by considering each of its subproblems. Therefore, we need to make sure that our integrated solution would solve the individual problems without introducing other complex problems or incompatibilities.

2.2 Chapter Organization

The remainder of this chapter is organized as follows. Section 2.3 provides the main definitions and terminology we are going to use in the rest of this thesis. Following that, we provide an overview of the main research areas that this work uses and builds on. Namely, we review the areas of: web services and service oriented systems and architectures in 2.4; service description and modeling in Section 2.5; service discovery in Section 2.6; service selection and ranking in Section 2.7; service trust and reputation in Section 2.8; and cloud computing in Section 2.9. Section 2.11 summarizes the findings of this chapter.

2.3 Definitions and Terminology

We define a *Service* in this environment to be a collection of functionalities provided over the Internet by an external provider.

Another definition that is also relevant is provided in [11]: in technological terms, *services* refer to software applications, methods, operations, or communications between two computing objects, or the interface between two software components.

A Service *Provider* is an entity or organization that supplies services online. A Service *Client* or *Customer* or *User* is a person or an organization that wishes to consume online services. We will use these terms interchangeably.

Providers specify their services in functional and non-functional terms. The functional specification is usually called a Service *Description* which basically describes what the service does. The non-functional specification includes issues like service quality and service level and is often described in a *Service Level Agreements (SLAs)*, either formally or informally.

Users usually state their *requirements* to describe the functionality they desire from the service, effectively searching over service description. We use the term *Constraints* to refer to user restrictions on the non-functional aspects of the service, or its SLA. Finally, we use the term *Selection Criteria or Preferences* to refer to the priorities of the user for selecting a service and provider. These criteria will be described in detail in later sections.

A Service *Broker* is an entity or organization that can intermediate between service requesters and providers. In the context of Service Oriented Architectures (SOA - see Section 2.4), a broker provides service location, brokered trust arrangements, and other facilities. The term *broker* has long been used in human affairs to refer to an intermediary with specialized knowledge who works toward a mutually desirable outcome through negotiation. In technical literature, and particularly in distributed systems, the most known example of a broker is the Object Request Broker (ORB), which is the core component in the Common Object Request Broker Architecture (CORBA). ORBs provide basic object interoperability functions between heterogenous object-oriented systems that are implemented using different programming languages and operating systems [6]. In essence, ORBs allow objects implemented in one system to invoke remote method calls to objects in another system. The broker in this case provides all the necessary plumbing to facilitate this interaction.

According to [2], Brokers can be classified in two main types. The first type is the *forwarding broker*, in which the broker plays the role of the intermediary between clients and servers for all interactions. The broker is thus the main plumbing that connect providers to their clients, and no direct interaction between clients and providers occur. This type presents a unified mechanism to abstract over, and allow connecting between, heterogenous entities or systems. The core ORB functionality described earlier is the most common example of such type. The second type, the *handle driven broker*, serves only as a kind of name server. It acts as a meeting place for demands and offers, where its main role is matchmaking. After clients locate the desired providers, they interact directly, without broker intervention. The second type is more often described as a naming or trading service, which is also one of the CORBA services that are additional to the main broker function.

Our proposed broker system is of the second type; once clients locate the required provider and offered SaaS using the system, they proceed to provision the service without intervention from the broker.

2.4 Web Services and Service Oriented Architecture

Service Oriented Computing (*SOC*) is a computing paradigm that utilizes services (see 2.3) as the fundamental elements for developing applications [37]. The standards body, OASIS, in their Service Oriented Architecture (*SOA*) Reference Model [35], define SOA as an architectural paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. According to the World Wide Web Consortium (W3C), a Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [49].

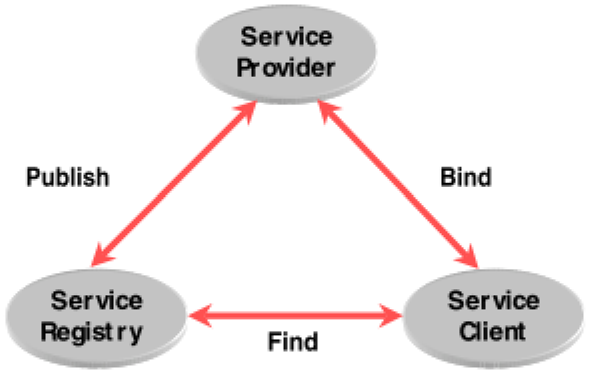

Figure 2.1: Service Oriented Architecture: Basic Roles [46]

There are typically three main roles in a service oriented environment. Namely, those are service providers, service clients, and service directories or registries (see Section 2.3 and figure 2.1).

In light of the previous definitions, we can attest that the concept of a service and service orientation is of paramount importance to realizing the visions of SOC/SOA. While it is widely believed that building a SOA means using web services *technology*, the fact remains that the stack of web services technologies, as defined by the standards of W3C (see figure 2.2), are but one way of doing SOA. Therefore, SOA should not be treated as a one way solution to all software problems; it is rather a way of thinking, a paradigm shift, when developing possible solutions.

From a software development point of view, we observe two main incarnations or usage of SOC/SOAs. The first one is the static, or development-time SOA. In this model, a developer would search for an existing software component published as a (web) service and attempt to develop her new application against the *interface* provided by that service as described in its contract (the *description*. However, once the new system is developed and deployed, its link to the used service is fixed and cannot change for whatever reason during runtime, such as a prolonged unavailability of the service that will render the dependent system unstable as well.

The other, less common in practice, form of SOA is the dynamic, late-binding model. In such incarnation, a developer would only fix the *requirements* needed for the correct operation in the specification of her new system. At run-time, an intermediate entity such as a broker or registry, would search its repository for a satisfying service instance and bind to it. Subsequent run of the dependent system may or may not use the same service instance. From our observation of industry practices, such powerful

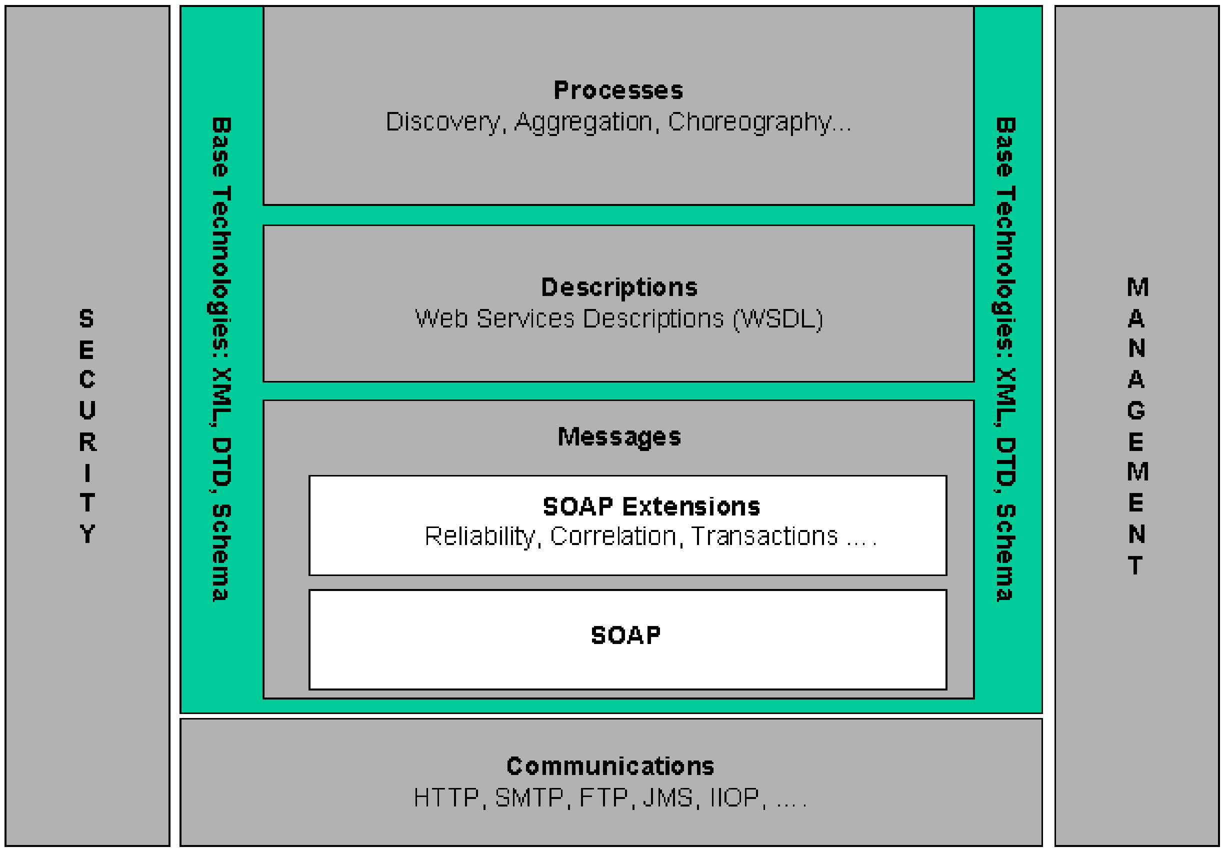
S E C U R I T Y
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
M A N A G E M E N T
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure 2.2: Web Services Architecture Stack [49]

model seems to be in rare usage. This might explain why the public UDDI servers of IBM, Microsoft and SAP were discontinued by 2005.

SOA provides an inspiring model for new generations of distributed, integrated, and Internet-scale applications today. SaaS and cloud applications might generally provide an API for programmatic access along with a human-friendly interface. Therefore, it is our goal in this work to try to support both kinds of system architecture developments. By utilizing SOA models and providing a service-oriented APIs for our system, we allow a multitude of unforseen programmatic interactions with our system at run-time. Whereas a human service requester looking for the best candidate online application to satisfy her immediate requirements can use a human-friendly web-based interface to issue the request and obtain the results.

We target SaaS applications (Section 2.9) as the fundamental entities in dirSaaS, our proposed directory system. While UDDI provides similar basic functionality to our proposed system, it does not fully meet our requirements for multiple reasons. First, while UDDI supports generic service description models using its tModel element, it was designed mainly for web services and its WSDL description model. While SaaS applications might provide a web sevice interface, they are usually offered at a higher level of abstraction. Second, UDDI architecture is centralized, providing little support for Internet-scale scalability. Third, service discovery in UDDI is rather limited and we provide a better alternative in dirSaaS. Fourth and most importantly, UDDI does not provide any facilities for customized or personalized recommendation on top of the matches returned by the discovery process. One of the main objectives in this work is to provide such a recommendation facility.

2.5 Service Modeling: Description, Quality, and Service Level Agreements

Since the service concept is a broad one, and since our focus is on the software aspect of a service, then we cannot manage a service directly but we need some *description* that represents the service and can be handled by machine processing. Metadata is machine-processable data that describes resources [20]. Thus, we view a service description as metadata that describes a service. A service description describes the characteristics, capabilities, and other information *about* the service to facilitate understanding and interaction with the service. The view of description as metadata allows the reuse of existing literature works on metadata processing and interoperability.

Based on this metadata view, we discuss service description by studying three different aspects. The first is the level of abstraction in describing the service, the second is the notion of functional and non-functional properties of services, and the third is the technical details and schema of service description records.

At the lowest level of abstraction, a service description is intended to specify the programmatic interface, including input and output parameters and data types, or the access point to the service, thus becoming a contract between the original service designer and potential service users. This is, for example, what Web Services Description Language (WSDL) [12] is intended to allow.

At a higher level of abstraction, a service description attempts to specify the high-level semantics, attributes or capabilities of the service using formal or informal terminology. Description at this level may be targeted for human consumption, and thus focus on easy readability of the format, or targeted for intelligent machine processing, thus focusing more on expressive power and flexibility. Examples include OWL [9] and Unified Service Description (USD) [28]. USD is intended as a meta-description rather than a description; it encapsulates descriptions from different schemes and technologies into the *unified* description, achieving interoperability between them.

So far we mentioned the functional aspects of a service description, which is specifying *what* the service does. Another aspect of services to describe is the non-functional properties [36] that the service entails, which is *how* a service would perform its functions. Such properties include temporal constraints such as response time, reliability, security, and other properties.

There have been several proposals for a technical standard for describing quality of a service and representing it in the form of Service Level Agreements (SLAs). Among them we mention IBM's WSLA [15],[24], WSOL [45], OWL-Q [25], and SLAng [42]. These proposals provide some attempts to formalize the description of QoS attributes and dimensions.

2.6 Service Discovery

At an abstract level, the process of obtaining a set of services which can possibly fulfill a user request is called Service Discovery [41]. The essence of this process is to retrieve the service description documents that satisfy the user query from some back-end database of service description records. This process is often referred to as a service by itself, the discovery service.

There are a number of proposed systems for service discovery that are targeted at different objectives [4]. The most used of those approaches have been Service Location Protocol (SLP) [19], Jini [50], UPnP [5] and UDDI [34]. SLP is an IETF standard that is geared toward networked applications and devices using relatively lower level abstractions and mechanisms. Jini is a Java-based approach to building dynamic distributed software components, but it did not become a standard. UPnP is an industry standard backed mainly by Microsoft (it became an international ISO standard in 2008) that is targeted primarily at solving the needs of personal networked devices. the UDDI specification from OASIS is still considered the primary standard approach for discovery in the web services stack and SOA environments.

Service discovery systems are mainly classified based on the architecture used to store, distribute and access service information. Thus, the architecture of discovery systems range from fully centralized to fully decentralized. The degree of distribution of the discovery system is much dependent on the context it is used in and the requirements of that system. The survey in [4] presents a detailed classification criteria and evaluation of major discovery systems.

A major effort to interoperability between heterogeneous discovery systems was proposed in OSDA [28]. OSDA is itself a discovery system, allowing service requester agents implemented in a particular discovery system to issue requests and retrieve results from another system. OSDA achieves this function through a technology dependent adapter that translates messages between the discovery system specific representation to an intermediate format (Unified Service Description) and vice versa. The intermediate messages are forwarded by technology-independent broker to a distributed p2p service directory. In dirSaaS, We have reused and built on many parts of OSDA. For example, we have reused the Chord-based P2P architecture and its INS/Twine-based indexing mechanism. The description model, USD, was updated and enhanced to include SLA information, which was not considered in OSDA. The main broker, however, was considerably updated to reflect its new role in dirSaaS. The implementation of dirSaaS is discussed in Chapter 5.

2.7 Service Selection and Ranking

A naive service discovery process returns all the results found during discovery in no particular order. However, when the number of returned service matches is large, a mechanism is needed to assist the decision making for selecting the most appropriate of those results. The problem is that all services returned satisfy the discovery request requirements, and thus we need another mechanism to distinguish between them.

The classic solution to this problem is to use a ranking mechanism based on some useful criteria to the requester. One of the most useful ranking criteria is to evaluate the adherence of service providers to their obligations in interactions with previous clients.

As we described in Section 2.5, these obligations are usually expressed in the form of a service level agreement (SLA). The SLA would define the quality of service (QoS) parameters and their obligated thresholds.

There is a need to obtain the runtime values of these parameters and compare them against the promised offer values in the SLA. This process is usually called SLA monitoring [38]. Assuming an SLA monitoring agent is in place to collect runtime behavior of services (a topic that is outside the scope of this thesis), these monitored values can now be used for further processing for obtaining service ranking.

It is reasonable to assume that multiple criteria can be used to evaluate a service. Therefore, an algorithm is needed to combine these values (which have different domains and types) in a single aggregate service score. The most common method used is the Simple Additive Weighting (SAW) which is an essential technique in Multiple Attribute Decision Matching [21]. In the SAW technique, each attribute needs to be scaled to a normalized value and given a weight indicating its importance. The final score is the summation of the product of the scaled value and weight for all criteria. This approach is the most used one in proposed service selection algorithms [27], [53], [3], [7].

The mentioned approaches, however, differ in what attributes are used to derive the final score. Most of them apply SAW to quality attributes to derive the final score, yet each of them assume a different set of fixed quality attributes. On the other hand, [27, 26] proposes a unique two-step approach for the service ranking. The first step applies the SAW technique to scaled quality parameter values to combine them in an aggregate value, just like the other approaches. The second step uses the aggregate quality value with other criteria (cost and reputation) to derive The final score of the service. This method appears to reflect a more comprehensive evaluation of the service under consideration, and it is the one used in our proposed system.

2.8 Trust and Reputation

Trust plays a critical role in online environments where providers and requesters have minimal a priori knowledge about the trustworthiness of their partners. Trust is a multidisciplinary concept with roots in economics, psychology, as well as computation [33, 51, 22]. A first attempt at formalizing the concept of trust in computing was given by Marsh [29]. Thus, trust needs to be considered as an important criterion in the service evaluation and ranking process. There exist various computational models to compute trust. Reputation is most commonly used in practice and literature as a quantitative measure of trust in computer systems.

In many deployed systems as well as research works, reputation is derived from the human submission of his or her own feedback about using the system or service under review. An example of a system that uses such reputation mechanism is eBay [51]. This approach, however, must consider the fact that these human reports are often subjective and may involve cheating or bias. Thus, some research works like [48] focus on detecting false and dishonest user quality ratings from both providers and users under various cheating behaviors. They rely on having few trusted monitoring agents to provide initial trusted quality reports on some services. Further reports are compared to detect if they are honest or cheating in a trust-distrust propagation mechanism.

Multiple research works on service evaluation and recommendation consider reputation as an important criterion but each handles it from a different point of view. Some works [52],[53] consider reputation as a generic selection criterion and therefore do not provide any special handling of it as it is assumed given using some form of reputation system. Others [30, 31] take the point of view that reputation of a service is a summary or aggregation of its quality attributes. Thus, reputation is not an explicit measure but rather an abstract notion implicit in the underlying service quality attributes.

A unique approach is presented in [27], [26], where reputation is handled differently from the other two selection criteria (those are quality, cost, and reputation). Reputation is forecasted statistically based on the time series of previous performance feedbacks. This method ensures that any potential bias in the subjective user feedbacks is circumvented and instead objective and reliable service performance is used instead to obtain an estimate of the service reputation. We adopt this approach in this work.

2.9 Cloud Computing and the as-a-Service Model

Cloud computing is the latest trend in the industry that tries to reflect the current level of technology penetration in society. Since it is still an evolving paradigm, there is no current widely accepted definition for it. However, we find that the following definition from the National Institute of Standards and Technology (NIST) [32] reflects much of the current buzz in the industry. NIST defines cloud computing as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model promotes availability and is composed of five essential characteristics, three service models, and four deployment models.

The essential characteristics that distinguish cloud computing are those of: on-demand self-service; broad network and device accessibility; resource pooling; rapid elasticity;

and measured service. Those characteristics are not unique on their own to cloud computing. Rather, it is their combination that distinguishes the cloud from previous computing paradigms.

Services branded with cloud computing generally work as one of the following service models, depending on the level of abstraction and the amount of user control over the provisioned service:

Software-as-a-Service (SaaS)

This software delivery model and term existed long before the cloud term itself existed. The most familiar example of a SaaS is a web-based email such as GMail. The application is provided through an accessible thin interface (most likely a web-based one). The service provider is responsible for all maintenance and management operations of the cloud infrastructure that underlies the service. This model is targeted mostly at human end-user consumers. Most people are actually using one SaaS or another unknowingly, since any website that provides some ”functionality” or application other than static web pages could be described as SaaS. The most cloud-related benefit to SaaS applications comes actually from the other two models, since they open the door for potentially infinite possibilities of new SaaS applications to be available in a rapid manner.

Platform-as-a-Service (PaaS)

This is a relatively newer model whose aim is to deploy customer-developed software to a hosted, managed cloud platform, using tools and programming languages supported by the provider. The provider again is responsible for managing the platform infrastructure. However, the user may have some control over the deployed application and some of its environment configuration. This model allows small software development firms to benefit from the vast infrastructure available at the cloud providers. This model is not equivalent to project hosting portals like SourceForge, since the hosted projects are just static files available for download, not deployed live code, and the user still has to deploy and manage the software on her own. Salesforce’s force.com is one of the early examples of PaaS, but the most popular today seem to be Google App Engine and Microsoft Azure.

Infrastructure-as-a-Service (IaaS)

In this model, consumers have even greater control ”almost to the metal” to manage operating systems, storage, arbitrary deployed applications as they deem reasonable. The provider is still responsible for the control and management of the underlying "physical" platform infrastructure. For example, the user has control or knowledge of the actual physical machine used to host his *virtual image*. OS Virtualization techniques play paramaount role in this model, as they isolate each user image completely from the underlying hardware. This allows, for example, an image to be *migrated* between actual hardware servers without the user noticing anything.

Cloud computing solutions have been adapted for different deployment models to satisfy different requirements and restrictions, whether they are technical and legal ones. These models are:

Public Cloud

The infrastructure hosting the cloud platform is owned by an organization that is interested in selling shared usage of the platform to the general public.

Private Cloud

The infrastructure is operated for the benefit of a single organization although it might be hosted or managed off-premise by a third-party. Some analysts e.g. [14] argue that "there is no such thing as a private cloud" since the organization still needs to budget a capital expenditure to procure, manage, and run this infrastructure, which makes the benefit of cloud computing much less obvious. However, we argue that there are still large benefits (based on the characteristics mentioned earlier) to end users inside the organization to use the cloud model rather than *the old ways*. IT operations inside an organization would be much more streamlined and organized (e.g. there should be no more struggle to know if a lone old server in the machine room is actually needed anymore or not). Regulation compliance, for example, might be a much important factor than cost to force organizations to consider the private cloud model instead of a public or hybrid one. Despite the potential higher cost, the benefits mentioned are well worth the investment.

Community Cloud

The infrastructure (which might be hosted and managed off-premises) is only shared between organizations of a specific community or having shared concerns or policies. Thus, this model shares characteristics of both public and private models.

Hybrid Cloud

This model allows an organization to use a combination of the above three models to support its needs by binding those models in a standardized or proprietary interfacing technology while keeping the composing clouds separated and unique.

As shown above, we can see that cloud providers are trying to offer a cloud solution for each IT problem. The fact remains, however, that migrating to cloud computing solutions might not be the best approach for every one. Security of data stored in remote cloud infrastructure, for example, is still a big concern for individuals as well as organizations. However, it was argued [18] that the security provided by a cloud provider is probably much more vigilant than what an IT shop in a small organization is providing currently.

As more providers and start-ups jump on the cloud bandwagon, however, a multitude of problems would face potential users that are not related to a specific provider but rather to an industry that would be a victim of its own success. The closest example is that of the Web 1.0 era, when the WWW was flooded by a huge number of new web sites every day that users had no idea they even existed. So one major issue with cloud computing in the future would be *finding* the appropriate service and service provider. Another related one is the *evaluation* of potential services against each other and the *trustworthiness* of the potentially unknown providers. Today, we have a few big well known providers that supply mostly the IaaS and PaaS infrastructure. Once smaller vendors arrive at the scene to utilize the potential of such available infrastructure, as we are only beginning to see today, then these problems would continue to grow.

2.10 P2P Systems and Architectures

Peer-to-Peer (P2P) systems are an important paradigm of distributed systems that lack centralized control or hierarchical organization. There is no clear distinction between a client and a server, or put differently, a *node* can play both client and server roles [44], [1].

Indexing content in P2P systems is an essential task to facilitate retrieval of the shared content. Depending on the scheme used to index the content, P2P systems can be categorized as unstructured, semi-structured, or structured [4], [13]. Only in structured schemes like distributed hash tables (described below), the indexing information is homogeneous among the nodes, thus providing efficient searching and routing.

Architectures of P2P systems can be classified as centralized, decentralized or partially decentralized [4] depending on whether the content indexing information is stored in a central node, in each peer, or in a subset of *super-peers*, respectively.

A Distributed Hash Table (DHT) is a distributed version of a centralized hash table. The main function of a hash table is to store key-value pairs and retrieve the values associated with a given key. Distributed Hash Tables provide insert and lookup information in a small (usually logarithmic to the number of nodes) number of routing hops. Systems built to provide such an abstraction are usually self-organizing structured P2P overlay networks that serve as a substrate for a large-scale P2P application [10]. Main example systems include Chord [44], CAN [39], Pastry [40], and Tapestry [54]. In a DHT system, the full domain set of keys (e.g. 32-bit key size provide 2^{32} keys) is partitioned among the set of participating peer nodes so that each node is responsible for insert and lookup operations for its own set and will forward requests to other keys to neighbor nodes.

The use of a P2P architecture to distribute a service registry has been proposed in the literature before in few works like [47], [17], [28]. P2P substrates, especially DHTs, provide many nice attributes that are of particular interest to our goal of building a scalable service management platform. Thus, our choice is to utilize a DHT P2P system (Chord) as the back-end for distributed storage.

2.11 Summary

The system proposed in this thesis touches on many interrelated problems in service oriented systems and platforms such as service description, quality, discovery, selection, and reputation. In addition, we discussed peer to peer systems and the emerging cloud computing paradigm. The knowledge we gained from studying these issues allowed us to proceed in designing the proposed system, whose overview we cover in the next chapter.

Chapter 3

Overview, Design Requirements and Usage Scenarios

3.1 Introduction

In the last chapter, we found that quite a few solutions are proposed in the literature to solve the problem of locating, evaluating and comparing Internet-based services. As more such services are offered, however, the need for more effective solutions and systems will continue to rise.

In this chapter, we are going to study this problem in greater detail, and try to develop a model for representing the domain of service provisioning in general along with all involved parties. We then present a high level overview of the proposed solution architecture. We describe the goals and requirements such a solution should have, and finally describe the scenarios of how the solution system would be used to solve the problem.

3.2 Chapter Organization

In section 3.3, we present our effort at modeling service provisioning scenarios using two-party and three-party models. Based on the three-party model, we present the overall proposed system architecture. Section 3.5 develops the overall goals and requirements that the design of the proposed broker system should fulfill. Section 3.6 will describe the major use cases of the system - advertisement, discovery and recommendation, feedback and administration. We conclude the chapter in 3.7.

3.3 Modeling Service Interactions

3.3.1 Basic Client-Provider Interaction Model

In basic Internet applications, there are only two interacting parties: the client and the provider. Clients need to locate providers themselves using a variety of methods. Providers also used a variety of (marketing) techniques to expand their reach of clients.

Figure 3.1 shows more details about this basic 2-party service interaction model for cloud services. The obligations of service providers are usually specified in the form of an SLA. A single service offering may have a single description of all service capabilities and one or more SLAs that different clients can choose from. Each SLA between a client and provider allows for one or more service usages by the client. The provider is responsible for maintaining the agreed service level for all service usages and to offer compensation when service level violates the agreement in some cases.

There are multiple potential issues with this basic model. First, the client has to have prior knowledge of the provider and service existence before being able to access any services. This is not a major problem when there are only a few well-known providers and few possible services. However, it is evident from the tremendous Internet growth that there are potentially endless possibilities for introducing new services and potentially many competing providers for the same service. To make matters worse, a service client has neither guarantees that the contracted service will maintain the agreed level nor knowledge about the service performance with previous clients. The client has to rely on its own observations to detect any service level variations. There is not even a guarantee whether the provider will act honestly or maliciously. There is basically no guarantee on the optimality of the chosen service in terms of cost, quality or trustworthiness.

3.3.2 Advanced Broker-based Interaction Model

As shown earlier, there are severe problems with the basic client-provider model above. In this thesis, we propose to introduce a specialized broker (Section 2.3) entity to solve these problems. The introduction of an intermediate broker into a two-party interaction is well known as a useful solution for abstracting and solving many problems in computing or in everyday life in general. The role of the broker entity is to provide a trusted platform that mediates between clients and providers. This role is metaphorically similar to the broker role in real life (as in real estate brokerage, for example).

The broker provides double-edge benefits: it provides clients with enhanced tools for searching and evaluating services; and at the same time enhances the visibility of

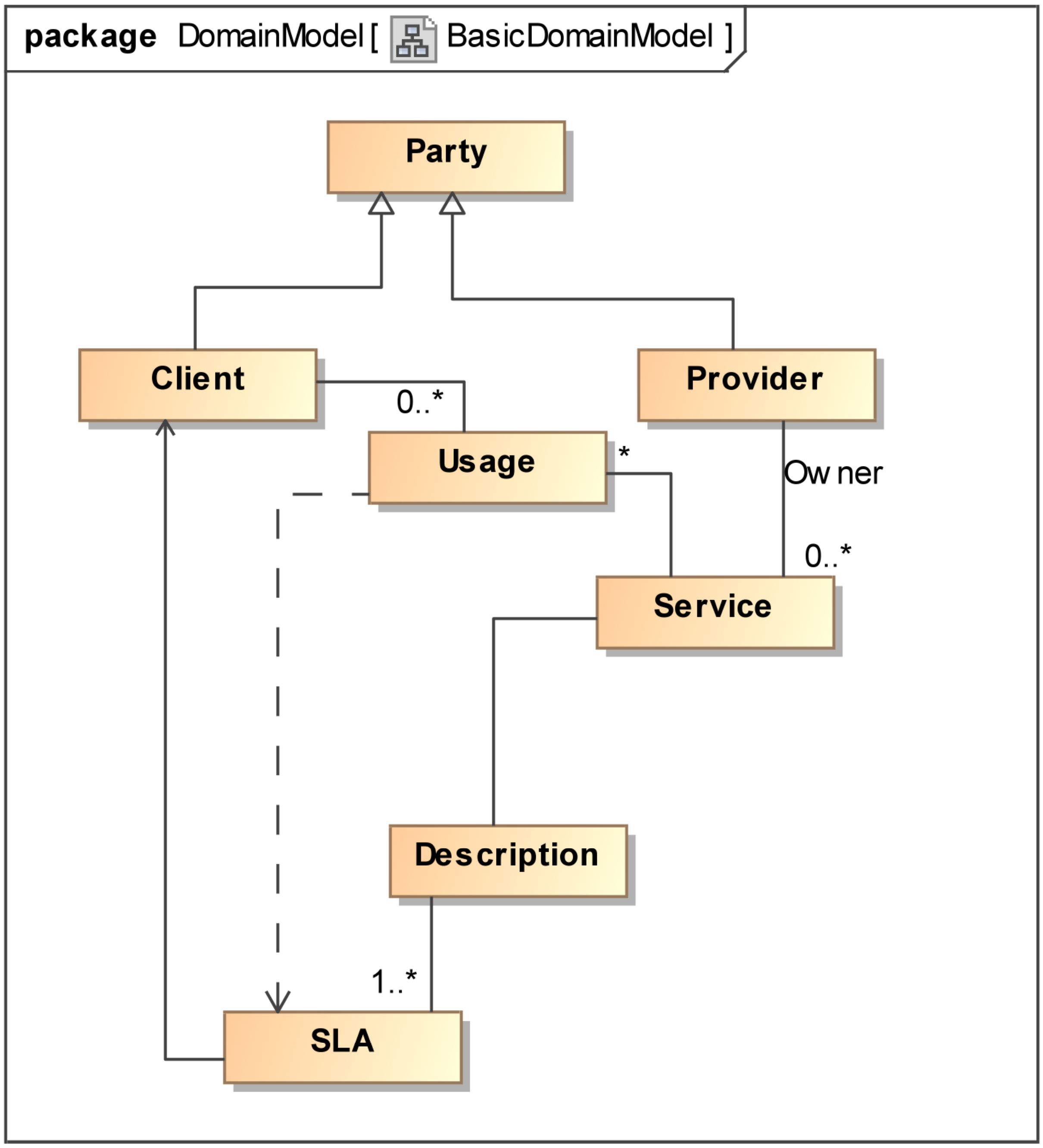

Figure 3.1: Basic Client-Provider Interaction Model

providers' services and gives them incentives for good behavior. The result is a much better experience for clients as well as a more competitive and higher quality service market. We believe it should be a win-win solution for all involved parties.

Figure 3.2 shows the interaction model when a broker is available. In enhancement to the basic interaction model described in Section 3.3.1, providers advertise their Service descriptions and SLAs with the broker. Clients use the broker search and recommendation facilities to locate and evaluate candidate services and providers. The broker's directory function provides a solution to the first problem clients face (how to locate and know services and providers). To solve the trustworthiness and service guarantee problem, the broker provides a service recommendation function that is based on objective reputations of offered services. The reputation information is obtained from the broker interface with an objective reputation system. This reputation system could be provided as an external service (to allow for different reputation mechanisms) or implemented inside the broker if no suitable reputation system exists.

3.4 Proposed Architecture: dirSaaS

The main entity or party in our proposed architecture is the broker. Brokers are independent organizations, and they can compete with each other. As we discussed earlier, there are different roles that a broker can serve. However, dirSaaS is mainly of the second matchmaking type. Its main role is to provide a powerful directory service with advanced search capabilities while maintaining quality guarantees on the search results. Clients turn to the broker to find the services relevant to their needs. The broker is not a gateway for service invocations and thus direct client-server interaction commences outside the proposed system after the client selects its chosen provider.

A high-level overview of the proposed architecture is presented in figure 3.3. The proposed broker basically functions as an enhanced load/store interface over a back-end directory or registry of services. It integrates a discovery system that provides interfaces for advertisement and discovery. Service providers are the entities that populate this directory with service offers they wish to advertise. The broker's main objective is to facilitate clients' discovery requests for services that match their requirements. Naive service discovery, however, does not provide flexibility or explicit ranking when presenting discovery matches to the requesting client. Therefore, the proposed broker also incorporates a ranking engine that provides the clients with personalized rankings and recommendations from initial discovery results. It is the ranking and recommendation engine that tremendously helps clients in their difficult decision making process of contracting an external service from the cloud.

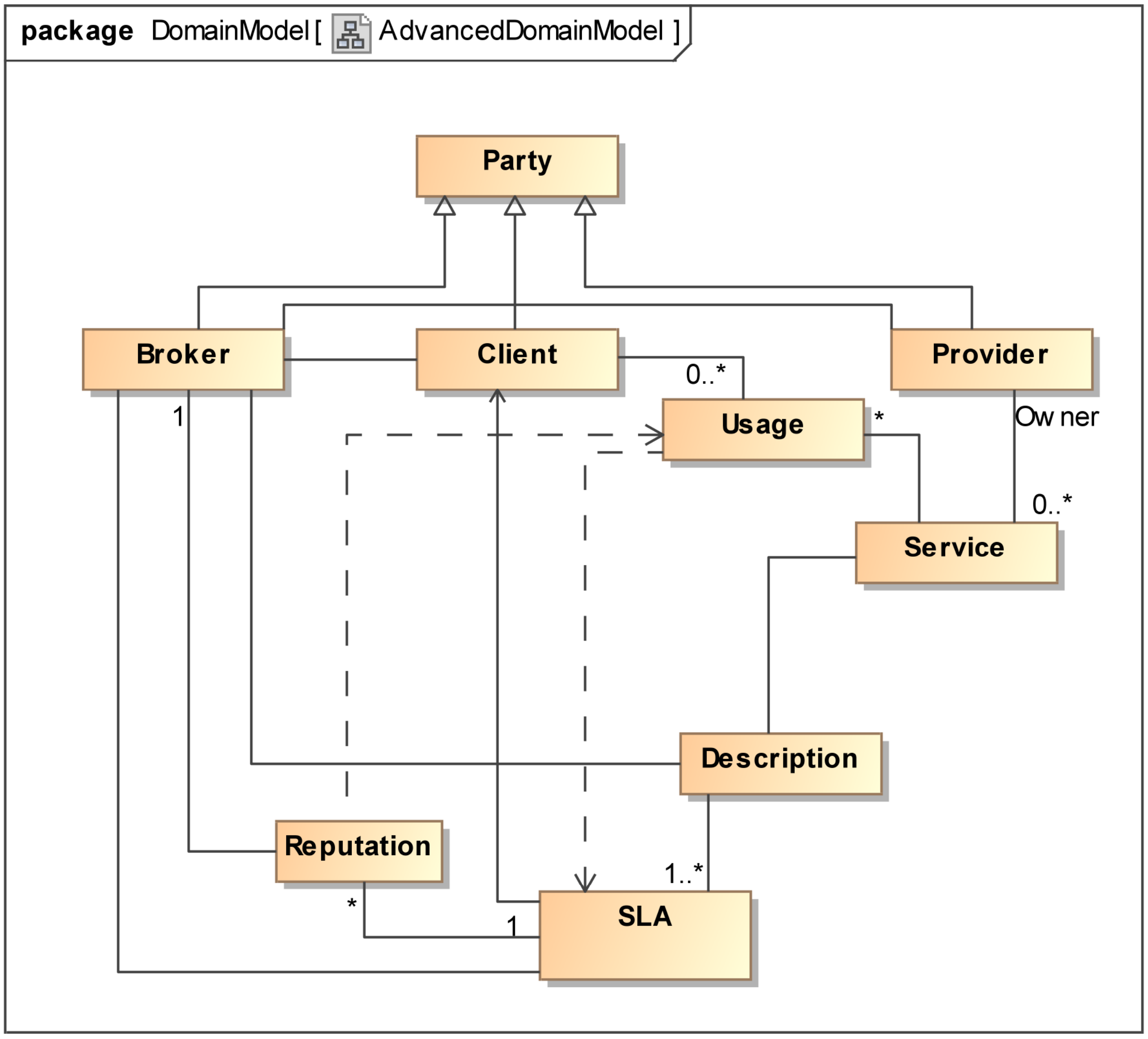

Figure 3.2: Advanced Broker-based Interaction Model

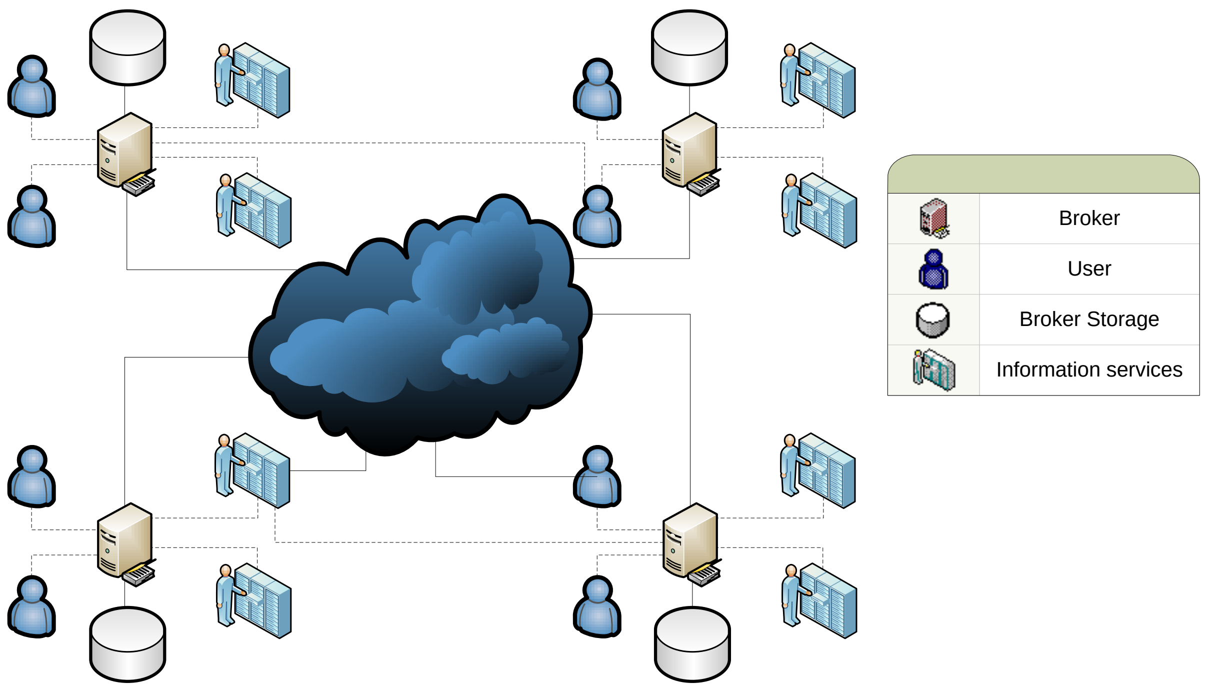

Figure 3.3: High-level Architecture

The broker relies on objective reputation information that reflect accurate service performance in its service recommendation function. The current broker architecture includes an internal reputation system that expects to receive feedbacks about previous service invocations from independent and trusted service monitoring agents [48]. However, the broker should easily adapt to obtain reputation from external reputation systems.

As previously mentioned, brokers are autonomous entities and may in fact compete with each other to attract more providers and/or customers. Moreover, providers can choose to advertise and customers can search with more than one broker. Brokers can opt to facilitate inter-broker search by participating in an inter-domain service discovery system such as OSDA [28].

3.5 Design Goals

The proposed broker system includes several components, as described briefly in Section 3.4. In this section, we elaborate on the design requirements for dirSaaS. These requirements serve as guidelines for the detailed design of the system, which is described in Chapter 4.

3.5.1 Directory: Scalable Architecture

The architecture of a service directory ranges from fully centralized to fully distributed [4]. Since the system is targeted to Internet scale, it is critical for the broker's directory to scale with growing demand from users. The broker should be scalable to millions of users and several thousands of providers and services. Scalability provides a high-quality experience for both providers and users under variable loads. A low-quality experience with the broker system will reflect on the low expectations from the functionality of the broker itself, and consequently from the advertised services as well. The chosen directory architecture directly affects the scalability of the system.

3.5.2 Discovery: Effectiveness

Discovery is an integral part of our system. The most important evaluation criterion for discovery in our system is how effective it is in discovering services [4]. Effectiveness in this context comprises two complementary criteria: completeness and correctness. Completeness (recall) is the ability of a service-discovery process to return all matching service instances registered in a service directory, whether the directory is distributed or centralized. Correctness (precision) measures how closely a discovered service matches the user's request [4].

3.5.3 Data Models: Expressiveness

The purpose of service description and quality models in our system is to facilitate discovery and ranking/recommendation, rather than being complete behavioral or functional specifications of services. Thus, data models need to be expressive enough to serve the needs of both providers and clients when advertising or searching services. The level of service details should be balanced; too many details distract the user and complicate the search process, whereas too few details make the discovery and ranking almost useless. At the same time, it should be easy and efficient to process those data models at the machine level.

3.5.4 Recommendation: Objectiveness and Personalization

As explained earlier, there is a strong need for feedback and reputation systems in online environments. While many such systems have been introduced for various online content and activities, the major component of these systems is the reliance on human end-users to evaluate and provide their feedback on the items or services that they had experienced. The use of such subjective feedbacks arguably causes many problems to the effectiveness of those reputation systems [26]. Therefore, in our system, we require the use of an alternative method for obtaining those feedbacks. The feedbacks obtained using this method should be objective and reflect more accurately the perceived user view of the provisioned service, without any manual intervention from the user.

Service clients may have different perceptions and priorities when accessing services. They might be looking for the best quality at any cost, or the most trusted provider within a certain budget, or other constraints. Thus, the system should not offer the same service rankings and recommendations for all users. Instead, those rankings and recommendations should be tailored to the individual user preferences and constraints.

3.6 Usage Scenarios

In this section we will elaborate on the main usage scenarios of the system. These include service advertisement, discovery, ranking and administration, and feedback. The use case diagram in figure 3.4 illustrates all the use cases, their actors, and their relationships. Each use case is described below.

3.6.1 Service Advertisement

While search engines usually crawl and index the web on their own without any action from content provider web sites, it would be difficult to create a structured service index from plain text descriptions and SLA terms on service provider web sites. Thus, advertisements from service providers are the main method for populating the directory of our broker system. Providers can advertise their services with the broker as follows: follows:

1. Select or search for an appropriate service type. Services of the same type have different descriptions and SLAs but share the same structure of those descriptions and SLAs.

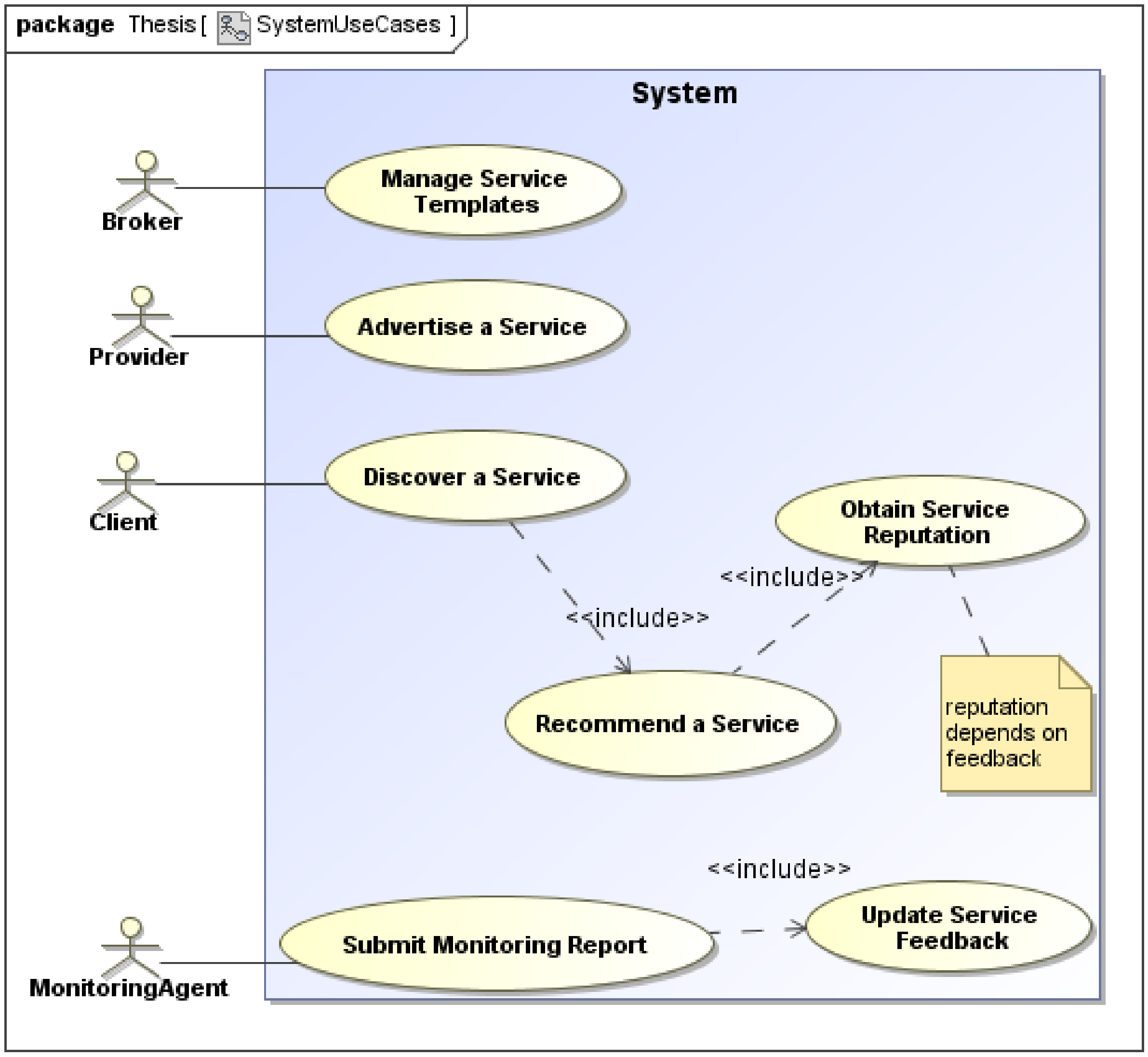

Figure 3.4: System Use Case Diagram

2. Provide service description details for that type.
3. Submit one or more SLA offers.

3.6.2 Service Discovery

Discovery is the process which allows clients to specify their requirements for searching the service directory and obtain the list of service matches. Those matches are not necessarily ordered or arranged in any specific order. A client could discover services as follows:

1. The client selects or searches for an appropriate service type.
2. The client specifies the desired features or capabilities.
3. The system processes the discovery request and return a list of available service matches, not in any particular order.

3.6.3 Service Ranking and Recommendation

To grasp a better picture about the merits of each offer in the available service matches, the broker evaluates service rankings and sends a personalized recommendation to the user. Once the discovery is done and the list of service matches is available, the user can get his/her personalized recommendations as follows:

1. The user specifies constraints and weights (importance) on individual quality attributes and overall cost accepted in the SLA offers.
2. The user specifies the weights (importance) of overall cost and quality.
3. The system processes those inputs with the raw discovery matches from the discovery use case (Section 3.6.2), removes SLAs that do not satisfy the constraints in the first step and ranks, influenced by the user-supplied weights in step 2, the remaining SLA offers. The details of this process is described in the next chapter.
4. The user chooses one of those offers to view more details or to order the service from the provider directly (outside the system).

3.6.4 Feedback

To support the internal reputation system, the broker needs to receive service quality reports from authorized monitoring agents. Once these reports have been submitted, the broker processes them to derive service feedback.

This use case proceeds as follows:

1. The monitoring agent submits service quality reports for that cover a predefined monitoring interval for a specific SLA.
2. The reputation system processes these reports to calculate service feedback.
3. The calculated feedback is added to the feedback history of that specific SLA.

3.7 Summary

In this chapter, we modeled a basic two-party service provisioning scenario using only clients and providers and described the difficulty to use it in modern environments. We then added a third-party - a broker - to that scenario and proposed a high level architecture of service provisioning using this advanced model. We described the important features of such an architecture to help guide its design, and enumerated the major use case scenarios of using it for effective service provisioning.

In the next chapter, we will describe the detailed design of such an architecture - the main components, their functions and their interrelatedness.

Chapter 4

System Design Specifications

4.1 Introduction

The reference model used by dirSaaS was described in the last chapter. To summarize, that model assumes the existence of service clients and providers that interact first indirectly through a broker and later directly between each other. In this chapter, we elaborate on the design details of the proposed system architecture that will fulfill the role of the broker entity. We describe all the components of the broker's core services and storage system. For each component, we describe its purpose and functionality, as well as its interface and interactions with other components. Then we describe the design of the system data model and data structures, including the messages between different components. Finally, we describe the user interface design of the system.

The architecture of dirSaaS builds on the main concepts of OSDA. The dirSaaS broker is composed of the Broker Core Services and the Broker Storage, and an optional System Interface. Providers and requesters interact with the broker system directly using the exposed API of the core services or through the optional system interface. The system interface is mainly targeted for illustrating the basic usage and to facilitate human interaction with the system. It can also be used in the administration task of the broker system itself.

4.2 Chapter Organization

The remainder of this chapter is organized as follows. In 4.3, we describe the main usage scenarios of the system using a black box (abstract) view of the system and its components. The broker components are then described in Sections 4.4 (Broker Core Services), 4.5 (Broker Storage), and 4.6 (System Interface). A detailed elaboration of the data models and messages used in the system is presented in 4.7. Section 4.8 summarizes the chapter.

4.3 Supported Interaction Scenarios

The system supports three main usage scenarios that are initiated by the external stakeholders of the system. Those stakeholders are service providers, service requesters and monitoring agents. Each of them initiate Service Advertisement, Service Discovery, or Service Feedback scenarios, respectively.

4.3.1 Advertisement Use Case

The service advertisement scenario is described in the sequence diagram in figure 4.1. It illustrates the design details of the use case described in Section 3.6.1. Further details about the Advertisement Component is described in Section 4.4.1.

4.3.2 Discovery and Recommendation Use Case

The service discovery and recommendation scenario is described in the sequence diagram in figure 4.2. It illustrates the combined design of the use cases of Service Discovery and Service Ranking and Recommendation described in Sections 3.6.2 and 3.6.3. Further details about the Discovery Component and its related components is presented in Section 4.4.2.

4.3.3 Feedback Use Case

The service feedback scenario is described in the sequence diagram in figure 4.3. It illustrates the design details of processing monitoring quality reports to obtain service feedback. Further details about the Reporting and Feedback Components is presented in Sections 4.4.3 and 4.4.4.

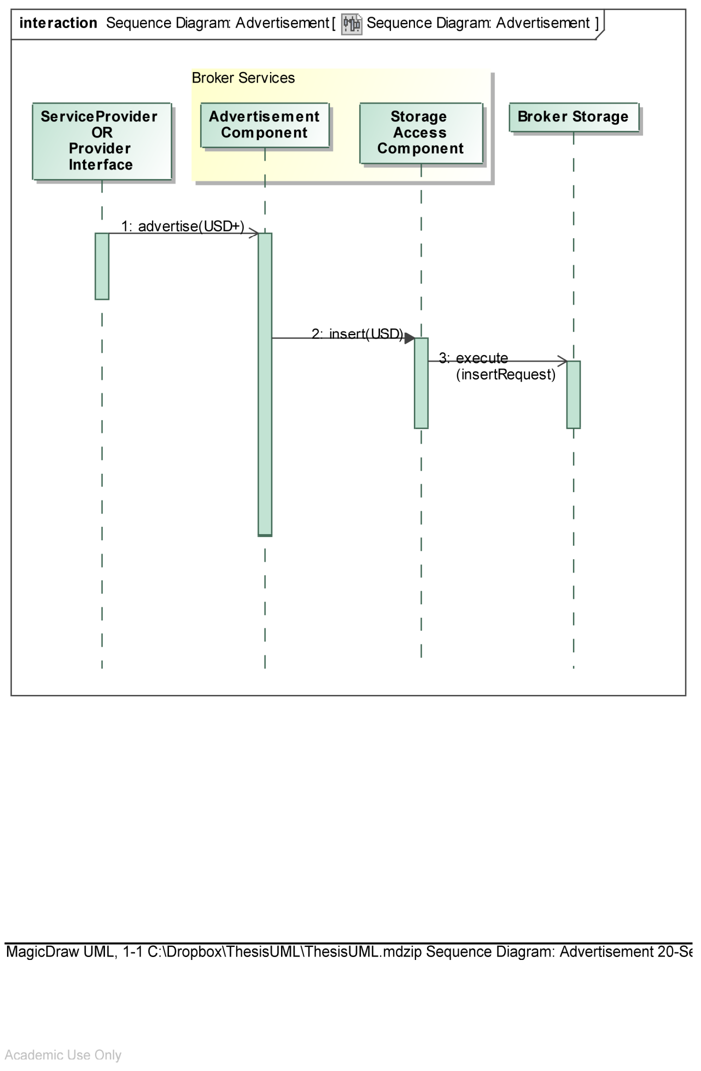
interaction Sequence Diagram: Advertisement [Sequence Diagram: Advertisement]
Broker Services
ServiceProvider OR Provider Interface
Advertisement Component
Storage Access Component
Broker Storage
1: advertise(USD+)
2: insert(USD)
3: execute (insertRequest)
MagicDraw UML, 1-1 C:\Dropbox\ThesisUML\ThesisUML.mdzip Sequence Diagram: Advertisement 20-Se
Academic Use Only

Figure 4.1: Advertisement Use Case: Sequence Diagram

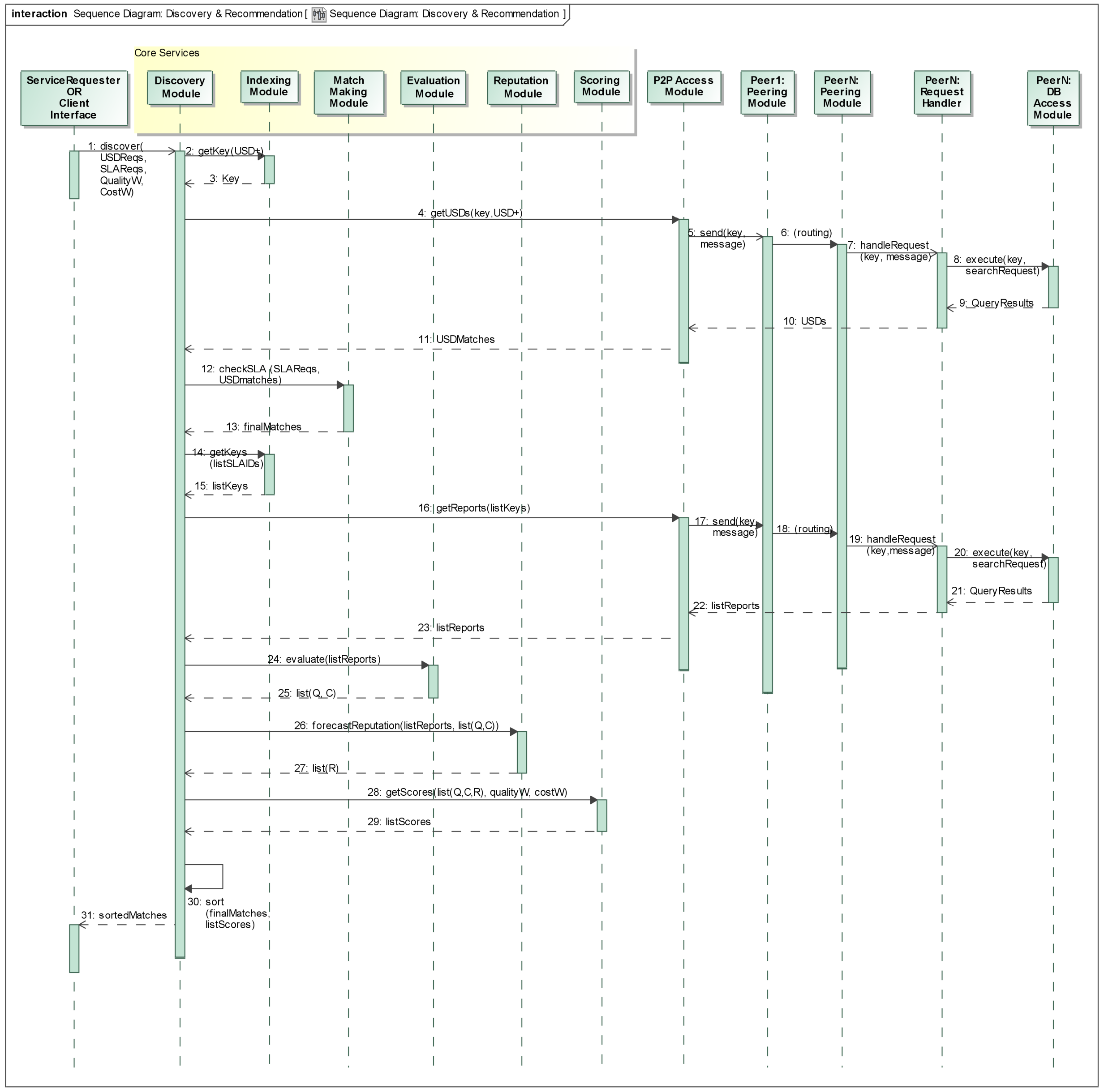

Figure 4.2: Service Discovery and Recommendation Use Case: Sequence Diagram

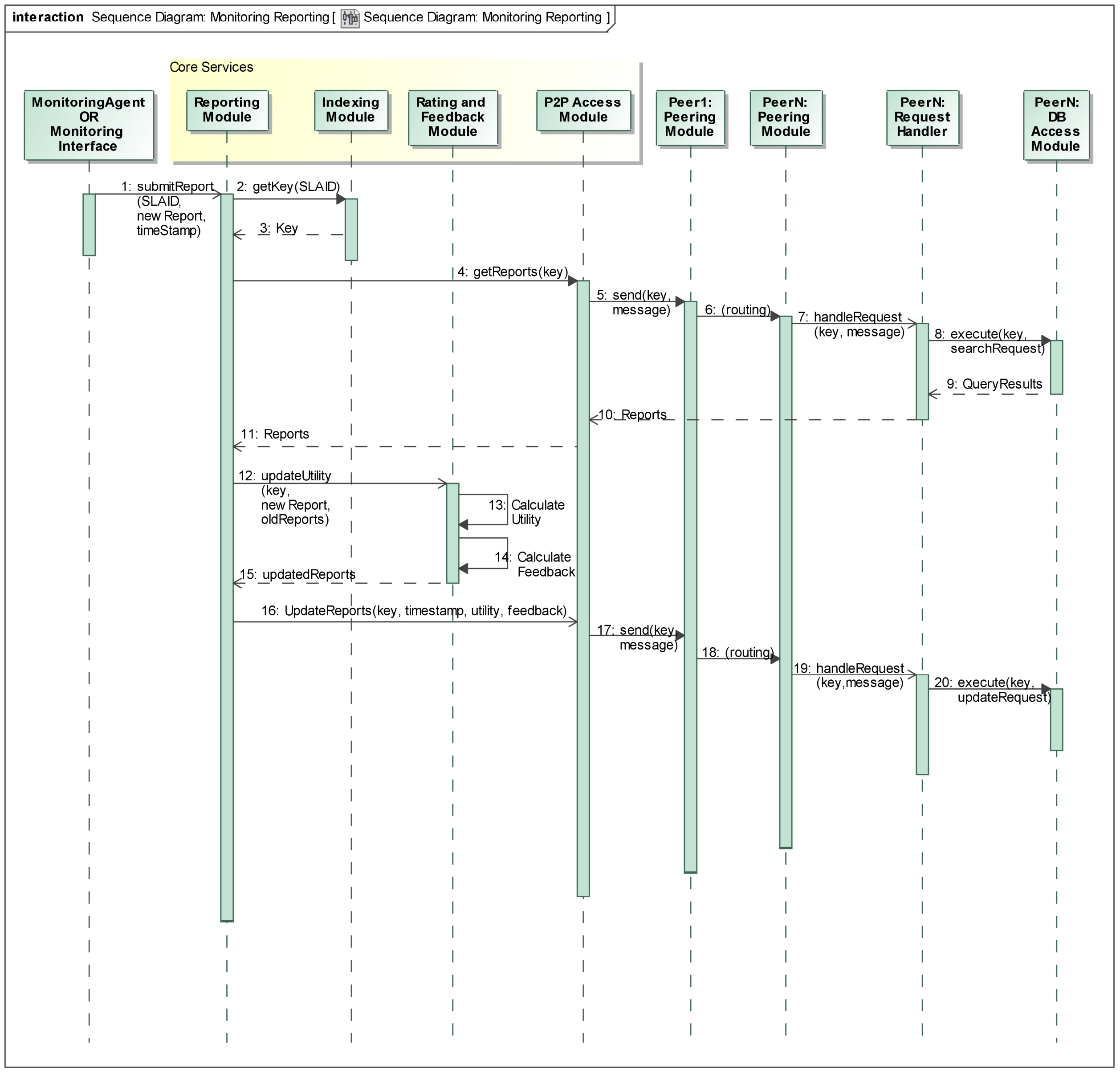
interaction Sequence Diagram: Monitoring Reporting [Sequence Diagram: Monitoring Reporting]
Core Services
MonitoringAgent OR Monitoring Interface
Reporting Module
Indexing Module
Rating and Feedback Module
P2P Access Module
Peer1: Peering Module
PeerN: Peering Module
PeerN: Request Handler
PeerN: DB Access Module
1: submitReport (SLAID, new Report, timeStamp)
2: getKey(SLAID)
3: Key
4: getReports(key)
5: send(key, message)
6: (routing)
7: handleRequest (key, message)
8: execute(key, searchRequest)
9: QueryResults
10: Reports
11: Reports
12: updateUtility (key, new Report, oldReports)
13: Calculate Utility
14: Calculate Feedback
15: updatedReports
16: UpdateReports(key, timestamp, utility, feedback)
17: send(key, message)
18: (routing)
19: handleRequest (key,message)
20: execute(key, updateRequest)

Figure 4.3: Feedback Use Case: Sequence Diagram

4.4 Broker Core Services

To achieve modularity and ease of development, the Broker Core Services are designed as loosely coupled components that interact together using well defined interfaces. We then proceed to describe the design details of each component.

4.4.1 Advertisement Component

This component is responsible for accepting a service description document in the USD+ format (Section 4.7.1) and then forwarding it to the Storage Access Component (Section 4.4.9), which then handles its persistence in the Broker Storage.

This component is one of the three publicly exposed interfaces of the system. It can be accessed directly from a service provider system, or through its wrapper in the Provider Interface (Section 4.6.3). Its public API is simply:

```
void advertise(USD+ serviceDescription, List<SLA> SLAs);
```

For simplicity, this call is one-way and does not return any value to the requesting party. Alternatively, this call can be asynchronous to allow the provider to receive a confirmation or failure message as a callback. However, this option would complicate the realization of the design.

4.4.2 Discovery Component

This component is responsible for accepting a query about the service description document again using the USD+ format (Section 4.7.1). As shown in figure 4.2, it acts as the main interface and controller for the discovery and ranking scenario.

This component is one of the three publicly exposed interfaces of the system. It can be accessed directly from a service requester system, or through its wrapper in the Client Interface (Section 4.6.2). Its public API is:

```
List<USD+> discover(USD+ USDRequirements,SLA SLARequirements,
              double qualityWeight, double costWeight);
```

Where:

- *USDRequirements* specifies each attribute of the description content in USD+ (4.7.1) that the requester wants to search. The difference from the advertisement USD+ is that not all attributes need to be defined in the query. The absent attributes would be considered as wildcards and will not restrict the returned matches. The content of each attribute should be the exact value required or a subset match of it.

- *SLARequirements* specifies each quality parameter of the SLA that is to be restricted by this query. Similarly to USDRequirements above, not all SLA quality parameters need to be specified. However, depending on the semantics of quality parameters (whether more-is-better or less-is-better), the specified values in this query act as an upper or lower bound on the returned matches.

- *qualityWeight* and *costWeight* are the relative weights (importance) of aggregate service quality and aggregate service cost, respectively. They correspond to ω_q and ω_c respectively in [26].

This call is synchronous (it waits for the response) and returns the ranked list of service matches.

4.4.3 Reporting Component

The reporting component acts as the link or interface between independent trusted monitoring agents [48] and the broker system. It is responsible for accepting a monitoring report (Section 4.7.3) submission on a particular SLA rather than a particular service or provider. This fine granularity is needed as the cost and quality parameters are different for each SLA of the same service, thus each service SLA is effectively an independent service instance by itself. Furthermore, the report itself is rather detailed. It lists each quality parameter of the SLA along with the value at the time of the measurement. A predefined number of reports for a predefined monitoring interval are submitted by the monitoring agent to this component.

It is assumed the SLA cost is fixed and does not necessarily vary during the validity period of each SLA. More advanced SLA cost modeling may account for compensation credit to the service user when the service level falls below the SLA specification, thereby reducing the cost of the service. However, the purpose of including SLA information and monitoring in this work is to facilitate the discovery of services by future users using objective feedbacks. Therefore, the variable cost incurred by the previous users has no effect on the decision of the new users. The quality variation, that led to the cost variation in the first place, is important in the decision making process and this is what we intend to capture from the monitoring streams of reports.

This component is one of the three publicly exposed interfaces of the system. It can and should be accessed directly from a monitoring agent system, but could also be used through the Monitoring Interface in the portal (Section 4.6.4), but this use is mostly for prototyping and testing.

This interface component has the following public API:

```
void submitReport(ID SLAID ,List<Report> periodReports,
                    Date reportingPeriod);
```

Where:

- `SLAID` is the ID of the reported SLA. This will be needed to locate the report database of this particular SLA.
- `List<Report> periodReports` is a list of monitoring reports where each report is an attribute-value list of all the SLA quality parameters and their measured values.
- `reportingPeriod` is the monitoring period covered by the submitted reports, not necessarily the time of submitting this report to the broker system.

The Reporting Component acts as the controller of the reporting scenario (figure 4.3). It sends the list of reports to the Feedback Component to perform the service feedback calculation. It then pushes this feedback value to the Broker Storage to be amended to the feedback series.

4.4.4 Feedback Component

This is an internal component that does not interact with outside of the Broker Services. Its main function is to calculate service utility and feedback based on the reports that were supplied from monitoring. This component is initiated when the reporting component forwards monitoring reports covering a monitoring interval. The feedback is calculated and then sent back.

A summary of the steps needed for these calculations is provided below. The derivation details are provided in [26].

Utility Calculation

Calculating the overall service utility ν is summarized as follows:

1. For each quality parameter in the SLA, if the monitored value is equal or better (either more or less depending on the type of the parameter), then the $Accept$ value for this parameter is 1, otherwise it is 0. This value for each individual parameter is stored in an OutcomesHistory (table) structure.
2. The probability for each quality parameter value to meet its expected value (the SLA value) is estimated as the ratio of successful previous measurements in the overall measurements. This probability is used as the individual utility of the particular QoS parameter. Thus, this value for each parameter will be updated when a new report is submitted.
3. The overall service utility is the multiplication of all the individual parameter utilities.

Feedback Calculation

Following utility calculation, feedback calculation is done using the following formula (See [26]):

$$Feedback = -\frac{\mu}{6}\nu^3 + \frac{\mu U_0}{2}\nu^2 + (1 + \mu(\frac{1}{6} - \frac{U_0}{2})\nu \tag{4.1}$$

Where

- U_0 is a constant that depends on the cost of the particular SLA. Assuming there is some limit $MAXCOST$ on the cost of the SLA, then U_0 is the ratio of the current SLA cost to $MAXCOST$ i.e. $U_0 = \frac{SLACOST}{MAXCOST}$
- μ is a constant that is calculated based on U_0 as follows:

$$\mu = \begin{cases} \frac{6}{2-3U_0} \text{ if } U_0 \in [0, \frac{1}{2}] \\ \frac{6}{3U_0-1} \text{ if } U_0 \in [\frac{1}{2}, 1] \end{cases} \tag{4.2}$$

- ν is the current overall service utility as calculated previously.

4.4.5 Match Making Component

This component checks a list of service descriptions (including a list of SLAs for each description) if any of those SLAs match the SLA requirements. Thus, this component receives this information as input and will remove the non-matching SLAs from the original list. If a particular service description has no single SLA match, then the whole service description is pruned. The component will return the list of matching descriptions with at least one or more SLAs that match the requirements.

The requirements are expressed in terms of cost and quality constraints. Services with higher cost than the cost constraint are filtered out. Quality constraints are expressed as the thresholds of accepted values of the SLA QoS parameters. Thus, the component has to check each parameter and decide from the parameter type if the threshold is an upper bound or a lower bound and accordingly filter the list of SLAs.

Match Making is an internal phase during the scenario of service discovery. Thus, it does not have any interactions with other broker parts or external components.

4.4.6 Evaluation Component

This component evaluates a list of SLA offers against each other. To achieve a meaningful comparison, we need to derive a scalar normalized value for cost (we will call it C) and quality (Q) that will allow each SLA to have a rank in those categories. These values will be used to derive the final scalar score value in the Scoring Component (4.4.7).

We will now describe a summary of the quality and cost normalization procedures as described in [26].

Quality Evaluation

Since quality is not a single value but rather a list of quality parameters, those quality parameters need to be normalized (scaled) individually, and later the normalized values are combined in a single Q value. This is done in the following steps:

1. For each quality parameter Q_i, two parameters $(q_i)_{max}$ and $(q_i)_{min}$ are defined to represent the maximum and minimum value of the parameter, respectively. If the parameter follows the *more-is-better* type (e.g. availability), then $(q_i)_{max}$ is the maximum offered value in all SLA offers and the $(q_i)_{min}$ is the requester constraint on this parameter. The reverse holds true for parameters of the *less-is-better* type (e.g. response time).

2. Each value q_i of a quality parameter Q_i is scaled to a value in $[0,1]$ as follows:

$$Scal(q_i) = \begin{cases} \frac{q_i - (q_i)_{min}}{(q_i)_{max} - (q_i)_{min}} & \text{if } Q_i \in QOS^+ \\ \frac{(q_i)_{max} - q_i}{(q_i)_{max} - (q_i)_{min}} & \text{if } Q_i \in QOS^- \\ 1 & \text{if } (q_i)_{max} - (q_i)_{min} = 0 \end{cases} \tag{4.3}$$

   Where QOS^+ and QOS^- refer to the *more-is-better* and *less-is-better* types of parameters.

3. Finally, the overall scaled quality value Q of the particular SLA is calculated from the vector of $Scal(q_i)$ using the following formula:

$$Q = \sqrt{\sum_{i=1}^{N} Scal(q_i^2)} \tag{4.4}$$

Cost Evaluation

Taking all SLAs together, the maximum (C_{max}) and minimum (C_{min}) is calculated and the cost for each SLA (c) is normalized (to C) as follows:

$$C = \frac{C_{max} - c}{C_{max} - C_{min}} \tag{4.5}$$

Therefore, this component will return the list of SLA offers, along with the normalized Q and C values for each offer.

4.4.7 Scoring Component

This component will produce the final scalar score value for each of the SLA offers it receives. As input, it takes the SLA offers to score, the normalized cost and quality values C and Q (4.4.6), the importance (weight) of cost (ω_q) and quality (ω_q)to the requester, and the forecasted Reputation (R) of each SLA (4.4.8). The score of each SLA is calculated using the following formula [26]:

$$Score(SLA) = e^{\lambda(R-1)} + e^{-\lambda}(\omega_q Q + \omega_c C - 1) \tag{4.6}$$

The component will return the list of the scores of each SLA offer it received.

4.4.8 Reputation Component

Reputation of a particular SLA depends on the previous performance history of that SLA, rather than what the user thinks (subjectively) of the service. This objective method of reputation calculation removes the potential bias that users may have when they evaluate the services they use. Depending on the user input to derive an aggregate service reputation has many documented problems in the literature [26] despite its apparent appeal and widespread use in current information systems deployed on the Internet.

The idea that we depend on in this work is that past service performance as quantified in service feedback (4.4.4) gives a strong indicator of future performance, although not at all times. Thus, the forecasting of the reputation needs to balance the past short-term fluctuations with long-term consistent behaviour and adjust (according to some criteria) the reputation value accordingly. Out of many possible statistical forecasting and smoothing techniques (like Moving Average, Weighted Moving Average, or Simple Exponential Smoothing (SES)), we will implement the reputation calculation using the SES technique due to its simplicity and reasonable performance as described in [26]. Thus the reputation time series $R(t)$ is computed using the feedback series $f(t)$ as follows:

$$R(t) = \alpha f(t) + (1 - \alpha)R(t - 1) \tag{4.7}$$

Where α is a constant smoothing factor in $[0, 1]$. α acts as the weight of the most recent feedback as well as an aging factor for all other feedbacks. The current (t) reputation is rather a weighted average of the recent (t) feedback and the last ($t-1$) feedback. Higher α values will give more weight to the most recent feedbacks but less for old ones.[26].

4.4.9 Storage Access Component

This component acts as the abstraction over the details of accessing the Broker Storage. It provides interfaces for the other components in the Broker Services to interact with the Broker Storage without being too coupled with the its choice of implementation.

The current choice of the Broker Storage is Chord [44], as in OSDA[28]. Thus, we need application-specific keys to act as the routing keys for Chord. The Indexing Component (4.4.10) is used for this task for all the functions performed by this component.

According to the current needs of the other components in the Broker Services, this component provides the following functions:

1. Inserting a service description and its associated SLAs. Nothing is returned by this function.

2. Retrieving and returning service description matches based on a description query.
3. Retrieving and returning service feedback history of an SLA based on its identification (Provider, Service, SLA).
4. Updating a particular SLA feedback history with an new value. Nothing is returned by this function.

4.4.10 Indexing Component

This component is dependent on the implementation of the Broker Storage. Its function is to facilitate the mapping from system data to the mechanism used by the Broker Storage to store this data. For example, when Chord is used for P2P routing, this component will generate the necessary keys to map the application data to the Chord key namespace. Thus, the design of this component plays an important role in achieving effective and efficient load balancing. Improper design here might lead to skewed storage and/or query load on the peer nodes.

4.5 Broker Storage

Our proposed system should scale to millions of users and many thousands of services. Thus its back-end storage should be distributed to offer graceful scalability. For this reason, we designed the back-end storage as a P2P network of cooperating nodes to support the storage needs of the broker system. It consists of three main components: a Peering Component, a Request Handler Component and a Database Access Component.

4.5.1 Peering Component

The selected structure for this component is a Chord [44] ring. Chord is a very popular structured peer-to-peer network that is based on the Distributed Hash Table and consistent hashing [23] concepts. It is known for its fault tolerance and self-stabilization properties [43]. The use of Chord guarantees an upper bound of $O(logN)$ on the number of hops for each routed request.

Each participating node in the broker storage implements this component to listen for and route messages based on their application keys. The final routing destination will pass the message to the Request Handler Component for parsing it and executing the relevant action.

4.5.2 Request Handler Component

Based on the type of the message received from the Peering Component, this component will interact with the Database Access Component to execute the required functions. Therefore, this component will parse the system-specific requests (e.g. retrieving service descriptions) to a generic database request that is sent to the Database Access Component.

This component returns the results (if any) directly to the requesting Storage Access Component. The contact information of the Storage Access Component should be used from the original request coming to the Request Handler Component. Therefore, there is no need to to route through the Broker Storage again for returning the response.

4.5.3 Database Access Component

This component is an abstraction over the database chosen to ultimately store and retrieve all required system information. The generic requests it can receive include $Insert$, $Update$, and $Retrieve$. Deleting information from the database is mainly used to automatically delete outdated records. Future system design enhancement might require the ability to explicitly delete specific information.

4.6 System User Interface

The System User Interface is mainly designed to illustrate simple use cases of the system. It supports manual human input and interaction with the main system. We divide it into four components: Administration Component, Client Interface Component, Provider Interface Component and a Monitoring Interface Component.

4.6.1 Administration Component

The main functionality of this component is to initialize and tune the system to be used by the other components and components. Some of the possible functions is to manage authorized monitoring agents, assign agents to specific providers or services, managing service description and SLA templates, and so on.

This last function requires further elaboration. The system relies on receiving messages and information in a proper format. This format or structure might or should depend on the particular service category or type used. We cannot assume that all service types should use the same attributes to describe their services or to constraint its SLA performance since some of these attributes or constraints may be meaningless in the domain of some services.

However, we assume that all services of a particular type should adhere to that type's template. This is necessary for the proper evaluation and ranking of offered services to eventually give a meaningful comparison to potential service users.

The source of these templates is a point worth discussing as well. Practically, service providers in a particular domain should agree to some approved or de facto standard to describe their services. If such standard exists, our broker system should use it as the service template. When the new services and service types emerge, however, the system administration should try to come up with such representation and potentially drive its standardization process along with help from partner service providers.

4.6.2 Client Interface Component

This interface part allows service requesters to interact with the system to browse and search available service types as well as service descriptions and SLAs.

4.6.3 Provider Interface Component

This interface part allows service providers to interact with the system to browse and search available service types as well as providing service descriptions and SLAs of the services they want to advertise.

4.6.4 Monitoring Interface Component

This interface part allows monitoring system administrators to input service monitoring reports of a specific SLA that they are contracted to monitor.

4.7 Data Model and Main Data Structures

In this section, we will describe the service information that our broker system relies on to achieve its purpose. In particular, we describe how services are described, how SLA information is represented and reported, what particular messages the system uses.

In dirSaaS, services are organized by a service type or category classification. A service type or category is a general classification to group related service offerings together. Examples include productivity software services (e.g. Google Docs or Zoho), Customer Relation Management - CRM (e.g. Salesforce) and Platform-as-a-Service (e.g. force.com or Google App Engine). Each service type is associated with a service template that provides the structure for describing the functionality as well as the quality of the service. One can think of a template as an XML schema or a database schema. The template just describes the structure of the service record. The broker manages this system schema, which then is used by providers to populate the database with conforming records or documents. The proposed system attempts to enforce standard service type names, standard parameters (and their names, data types, allowed values,...etc.).

Neither the clients nor the providers are supposed to create their own templates to describe advertisements or discovery requests. There may be a standardization body or an industry consortium to standardize those templates. For example, SLP templates and service types are IETF standards. Ontologies can be introduced to compare between heterogeneous templates that describe the same service type. Unified templates at least within one broker system are necessary for the correct behavior of the broker. There is no advantage for letting each provider define their own terms and definitions in their templates. In fact, that would introduce difficulties to detect the semantic differences between them. The broker could utilize an already standardized or widely accepted template to describe a particular service type. So, when a broker wants to allow a new service type in its market, and no standard template exists for that type, a provider could submit its template but the broker has to approve it to become visible. The broker may also make its own templates or solicit a provider or a group of providers to develop such a template. Instead of burdening providers to develop a new template each time they advertise a new service type, the broker can implement this task (probably with inputs from providers) and then all providers can just use the developed template to describe their services. The broker in this case is promoting standard mechanisms for describing services. This is in contrast to the real situation on the web today, where each provider is free to describe its services using its own terms (usually in unstructured plain text) and each provider claims their offerings are superior to other providers' offerings.

Because of its expressive power, platform independence and acceptance in the Internet, we are going to use XML as the basis of all data formats. To describe the used structures more formally and less ambiguously, we will use the XML Schema language to describe those structures and formats.

4.7.1 Service Description Model

The description model we use (USD+) is an extension of the Unified Description Model (USD) that was used in OSDA [28]. As shown in figure 4.4, the description schema consists of three parts that are enclosed in the overall document: meta information about the description such as its expiry time and the service type; the description content which specifies the general capabilities of the service; and a list of pointers to SLAs that the service supports.

4.7.2 Service Level Agreement Model

SLA Information could be described in a variety of proposed methods in the literature (e.g. WSLA [24], WSOL [45], SLAng [42]). At their core, any of these frameworks should be usable with our system. What we effectively need from an SLA is shown in figure 4.5. SLA is presented as a simple list of quality parameters, their definitions, and contracted thresholds along with an abstract cost value. Almost all available SLA description languages offer this basic information albeit with much more information than we need for the effective ranking of services based on their SLA performance.

4.7.3 Monitoring Report Model

The system uses two report models or types depending on where it is used. The individual report that the monitoring agent submits to the system consists of a list of values of the QoS parameters of the SLA they monitor. Along with the SLA identifier, this information would be enough for the system to retrieve the second report model. This more complete model is shown in figure 4.6. It includes the full SLA information in addition to the historical values of the SLA QoS parameters and the overall measurements of utility and feedback. This more complete information is what gets stored and retrieved by the system when any of that information is required.

4.7.4 System Messages

There are three types of message structures that are used by external components when communicating with the system. These are described below.

Advertisement message: This message is sent by the service provider to the Advertisement Component in the Broker Services. It contains two data structures:

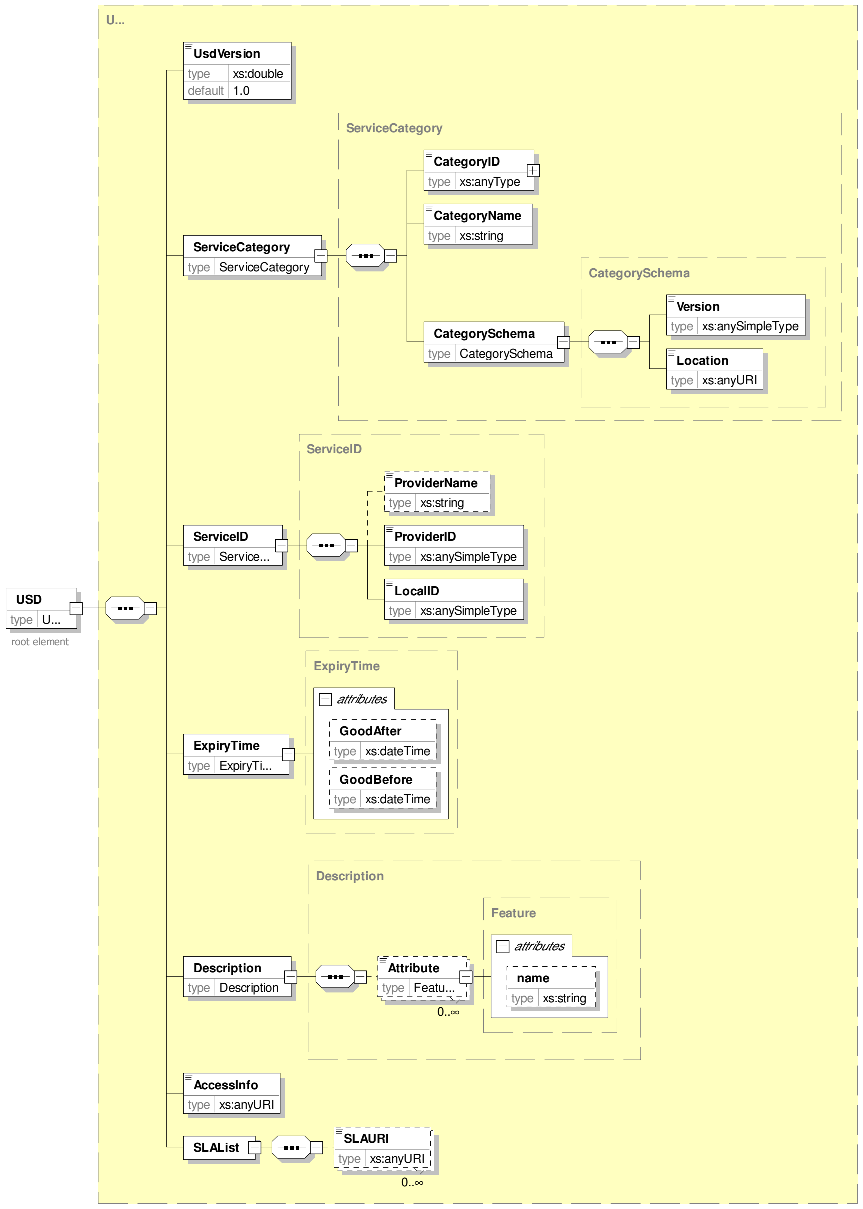

Figure 4.4: Data Model: Extended Unified Service Description (USD+) Schema

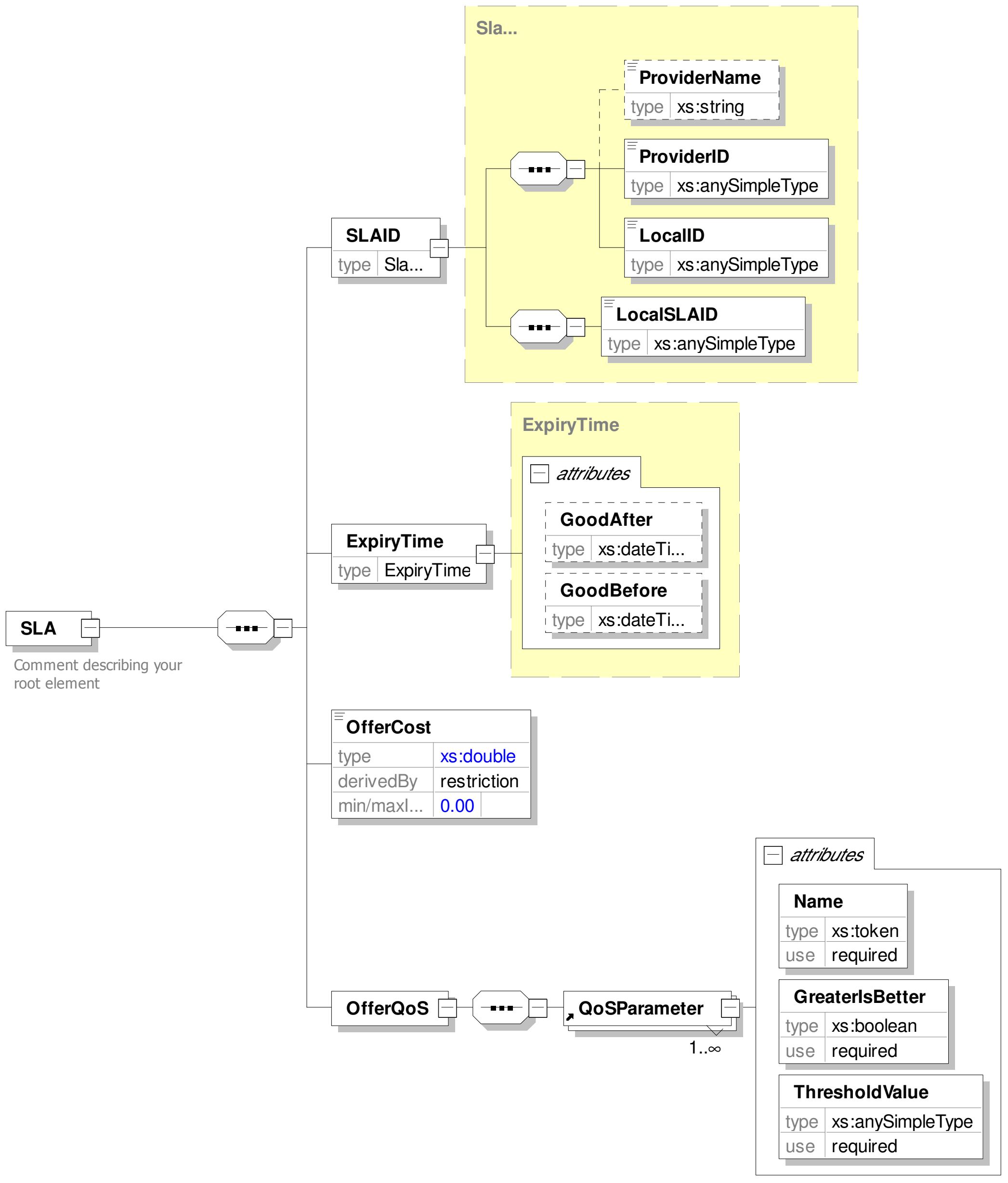

Figure 4.5: Data Model: SLA Simplified Schema

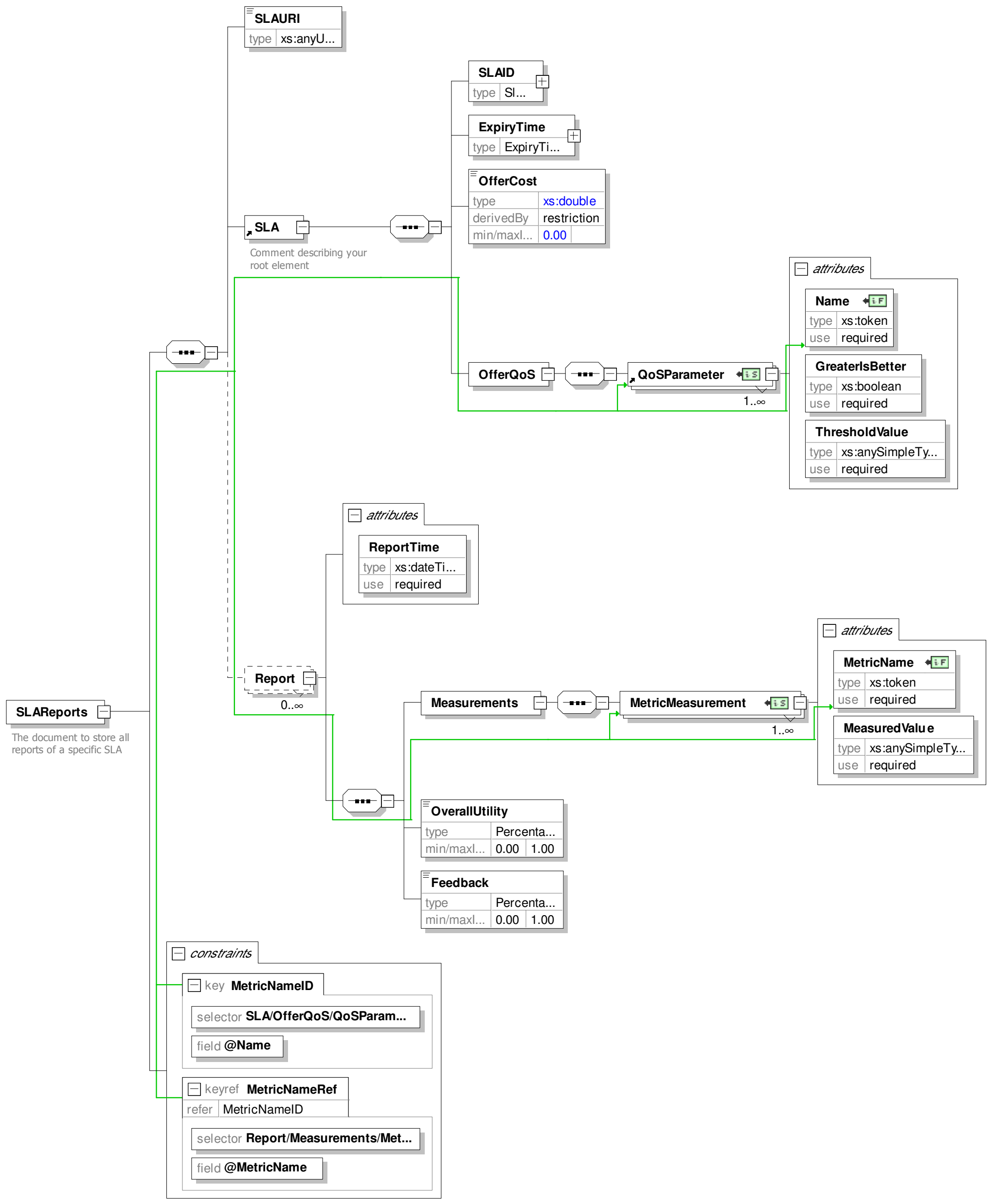

Figure 4.6: Data Model: SLA Complete History

- The service description content as represented in USD+.
- A list of the contents of each SLA of the service.

Discovery message: This message is sent by the service requester to the Discovery Component in the Broker Services. This message contains four parts:

- The service description query based on the same USD+ model but with potential empty attribute values.
- The SLA requirements based on the same SLA model but with potential empty attribute values.
- The importance of quality in ranking discovered services represented as a weight value between 0 and 1.
- The importance of cost in ranking discovered services represented as a weight value between 0 and 1.

Monitoring Report Message: This message contains three parts:

- The identifier of the reported SLA. This includes a Provider ID, a Service ID, and a local SLA ID.
- The report content as represented in the first report model (list of QoS values).
- A timestamp to indicate the date and time when the service data was collected, not the time of the report submission.

4.8 Summary

In this chapter, we provided the detailed design of the proposed broker system and its components. We also described how the system interfaces and messages with external components and entities. In the next chapter, we are going to discuss the steps and technologies used to implement a working prototype of the proposed system based on the design choices in this chapter.

Chapter 5

System Implementation

Having described the design of our broker system in the last chapter, in this chapter we delve into details of our prototype implementation. We describe the motivations and tradeoffs when designing and implementing the various components of the system, and compare the design choices in our system to those of OSDA [28], which is the basis of our work.

5.1 Chapter Organization

In section 5.2 we describe the used technologies and frameworks for implementing the various parts and their components. Then, we illustrate some of the implementation enhancements of dirSaaS over OSDA in section 5.3. Finally, we describe some of the new features of dirSaaS in 5.4 before summarizing in section 5.5.

5.2 Implementation Technologies

As the goal and scope of this system is to be an effective tool when interacting with Internet applications, our choice of implementation technologies was focussed on Internet standards and platform-independence using open-source frameworks and technologies. Because of the need for platform independence, we used Java as the programming language, HTTP as the main transport protocol and XML as the format for all messages and data structures. Web Services fit as a natural choice for such environment. We used Java EE 6 standards, such as EJB, JAX-WS, JAXB, SDO, as the reference APIs. We also relied on open-source components, such as GlassFish, JXTA, Chord, INS/Twine, Berkeley

DB XML. Since we have mainly web-based technologies, the resulting system is highly modular and flexible. The system is distributed by nature and thus all its components can be deployed on separate systems or all on the same machine. This helps dirSaaS scale naturally in the face of increasing load.

We will now describe the data structures used by the system, followed by discussing the implementation details of each part.

5.2.1 Data Structures

As described earlier and also in Section 4.7, XML is the de facto standard used to represent data in web applications and services for its expressive power and platform independence. Multiple tools and frameworks now support XML naturally. Thus, it satisfies the expressiveness design requirement that we described earlier in Section 3.5.

Since Web Services technologies use XML exclusively, then it would be a natural choice for all messages internal and external to the system to be based on XML and to use the latest XML binding technologies (static like JAXB and SDO, or dynamic like SDO). For that choice to work best, our back-end data storage is also a native XML Database (the embedded Berkeley XML DB).

5.2.2 Broker Core Services

The Broker Services functions as a Java EE Server. The components of this part are implemented using stateless session Enterprise Java Beans (EJB) using the latest EJB 3.1 standard. They run on a GlassFish server. These EJBs cooperate with each other to achieve their intended functionality. The components that support external system interactions (Advertisement, Discovery, and Reporting) are deployed as public web services. Those services can be accessed by other system parts mainly using SOAP/HTTP and relying on JAX-WS as the underlying technology of choice for implementing SOAP/HTTP. The input parameters to these services are XML documents that conform to our predefined XML Schemas.

Advertisement Web Service

This web service represents the main access interface for service providers who wish to advertise their services in the system. This web service is implemented as a one-way call that does not return any response to the caller. Once the input advertisement has been received, it is forwarded to the Broker Storage through the Storage Access Component for storage.

Discovery Web Service

This web service represents the main access interface for service requesters who wish to discover available services according to their functional and non-functional needs. Since the service requester is waiting for the results, the natural implementation would be a synchronous web service. However, since the steps required to get the final results evaluated and ranked is complicated and requires a lot of messaging, it would be best to change this web service to an asynchronous call and let the user know of the results using callback or polling for example.

The enterprise bean implementing this web service acts as a controller, handling all data retrieval and update from and to the P2P Broker Storage and forwarding them to (sub) components for further processing. Thus, those sub components (like the evaluation or reputation components) are decoupled from contacting the other components and act as simple functions doing calculations over the passed data. Those other (internal) components are implemented as local EJBs and perform their specialized calculations that were described in the last chapter.

Reporting Web Service

This web service represents the reporting component of the system and its primary function is to accept monitoring reports from monitoring agents and store them appropriately in the P2P Broker Storage after doing initial processing to calculate the utility and feedback of the reported service using the (internal) Feedback Component. The calculations were described in the last chapter.

Storage Access Component

This is the main access point to the Broker Storage from the Broker Core Services. It abstracts the details of communicating with the P2P implementation, and provides those communication facilities as high-level functions that handle the calls of appropriate components.

The other functionality of this component is to receive callbacks from the P2P nodes when they need to return some results to the Core Services. This functionality is implemented as a special web service that will receive the results from the P2P nodes and forward them to the requesting component appropriately.

Indexing Component

This component is an extended version of the Broker Request Handler and the Indexing Component of OSDA [28]. Its main function is to create INS/Twine [8] strands that are then hashed to generate the keys required for routing in the P2P Broker Storage.

Three kinds of keys are required in the new system for advertisements, queries as well as SLA monitoring reports. Advertisement and query keys are generated in the same manner as in OSDA. Multiple advertisement keys are generated based on the service description attributes or capabilities. Advertisements are then sent to all peers associated with those keys to improve redundancy and fault tolerance. Queries are generated from the same capabilities section but they get forwarded to the peer associated with the key of the longest strand.

Keys for monitoring reports are generated differently. They are created by hashing the ID of each SLA (4.7.2), which consists of the service ID and a unique number for each SLA. The reason for this design choice is to distribute the storage of reports independently of the description of their reported services.

5.2.3 Peer To Peer Implementation of the Broker Storage

Each participating peer node implements these components:

Peering Component

The peering component is implemented as a JXTA peer to take advantage of the bootstrapping, authentication and secure communication facilities in the JXTA framework. The peering components in all peers constitute a Chord ring, where Chord is used to route requests between peers. Chord is a very popular structured peer-to-peer network that is based on the Distributed Hash Table and consistent hashing concepts. It is known for its fault tolerance and self-stabilization properties [43]. The use of Chord guarantees an upper bound of $O(logN)$ on the number of hops for each routed request.

Request Handler Component

The Request Handler Component receives the messages routed by the Peering Component and, according to the type of the encapsulated message, calls the relevant database operation appropriate for that type. The supported message types are $insert$, $update$, and $search$ and the syntax of each of those messages is in XPath/XQuery. If the message type is $search$, the enclosing message includes return contact information to allow this component to call the web service at that address, which according to our design is the Storage Access Component in the Broker Services.

Database Access Component

This component handles the necessary API calls to Berkeley DB XML. BDB XML is an embedded native XML database that is built on top of the popular Berkeley DB. Since it is embedded, it can only be instantiated and called using its public API calls. This also means that creating new DB instances is handled automatically when a new peer is initiated and no extra work needs to be done. BDB XML also includes full XPath/XQuery processing, and allows (optional) XML Schema validation of inserted documents for ensuring the integrity of XML records.

5.3 Enhancements to OSDA

5.3.1 System configuration

The OSDA broker implementation uses static XML configuration files to supply environment parameters to the EJB broker components, like broker or peer address. This is against the official EJB specifications (to access the filesystem directly). Furthermore, these files were parsed using a custom XML parser that is internal to the implementation of a specific Jetty server. We have changed that to a dynamic form using JNDI environment entries parameters (with default values in ejb-jar.xml). The JNDI API allows those values to be updated dynamically at runtime without recompiling/repackaging the system.

5.3.2 Broker Implementation using EJBs

The enterprise beans have been updated to reflect the latest specifications (EJB 3.1) of the Java EE 6 standard. This simplifies their development and maintenance. It also allows using some of the latest Java enhancements like resource injection using annotations (that were used for the system configuration described above).

5.3.3 UnifiedQuery model

It was argued [28] that the UnifiedQuery model is suitable as queries could be built from advertisements. While this in theory appears reasonable, its representation in the same XML Schema definition makes validation harder, as it depends on parsing the type of message (Command). Indeed we could describe separate XML messages for each type of request/response pair. An example from industry is the XML Schemas for the OpenTravel Alliance (OTA) where each type of message request or response is described in a separate XML Schema document and each of these schemas reuse inner components that are shared between them. In our case, the inner Description element and its associated Attribute elements could be reused between the two and the fact that a query is a subset of the advertisement can be explicitly mentioned in the schema to ease the validation.

5.3.4 XML Database Choice

The current OSDA implementation uses eXist native XML database to store advertisements at the global P2P level. However, the implementation does not mandate nor facilitate creating a separate instance for each peer to distribute the storage. So, in order to assign a separate DB instance to each peer, manual deployment configuration is needed (this includes manually starting the eXist database and recording its address in a special configuration file that needs to be changed manually each time the address is changed).

So to improve upon the previous OSDA architecture, we describe the following requirements for a new choice of XML database. First, the database should support working in embedded mode so that it is instantiated and used implicitly by each running JXTA peer, alleviating the need to manually create a database instance upon starting a new peer and assigning the reference to it from the new peer. Second, it would also be preferable to use an XML database that supports XML Schema validation - in the sense that each xml document in a collection should validate against the specific schema for that collection. Third, it should provide APIs for Java or alternatively some web services-based or XMLRPC-based APIs. Fourth, supporting XQuery and its associated XQJ standard would be also an advantage. XQJ is to XML data sources what JDBC is to RDBMSs. Thus, using a standardized API to access the database allows to substitute the implementation without changing the code.

As of August 2010, none of the available products in the market support all the previous requirements. We evaluated eXist (currently used in OSDA), Berkeley DB XML, BaseX, Sedna, Mark Logic XML Server, MonetDB/XQuery, and Documentum xDB (formerly XHive). However, none of databases that support XQJ also provide embedded mode, so the disadvantage of a stand-alone execution for our P2P design outweighs this benefit.

We decided to use the Berkeley XML DB (over eXist) for the following main reasons:

- Per-document schema-validation (there is no mention of schemas or validation in eXist).
- Embedded use inside application (although eXist has support for this, its authors do not recommend it whereas BDB was built specifically as an embedded DB inside applications). In addition to the main requirement for the P2P nodes mentioned earlier, this feature eases administration considerably.

5.3.5 New approach for data binding

We propose to use JAXB and SDO instead of String objects that need to be parsed manually. The use of Strings to represent XML content is frowned upon in the community as it needs complete parsing before making use of it. This leaves us with XML data binding approaches which include XMLBeans, Simple XML and others in addition to JAXB and SDO. JAXB is the natural choice for web services as it is implicitly used by JAX-WS tools. The only problem with JAXB is its static nature. XML Schema documents have to be compiled separately before building and deploying the system and there is no way to add new types at runtime (e.g. using a new schema that extends the default one). SDO comes to rescue here as it allows both static binding and dynamic binding which means it is possible to create or extend new types at runtime. We need such flexibility to facilitate creating new service templates dynamically. We can rely on the static strongly-typed JAXB API for accessing the system messages and main structure (USD, Requests, Responses) and use the dynamic, weakly-typed SDO API for the templates of the services.

5.4 New Features in dirSaaS

5.4.1 SLA-related Design Choices

Since importing and using SLA information is a new requirement for our system in comparison to OSDA, we need to review the path this information is taking inside the system and how it is eventually stored and retrieved.

We propose to separate storage of USDs from their associated SLAs. The USD will still keep pointers to the SLAs to facilitate their retrieval based on the retrieval of the USD document itself. The SLA information, however, will be stored separately along with its associated monitoring data. This will facilitate the monitoring implementation since a monitoring report may not include USD information but only SLA information.

The current approach to generate P2P routing keys for USDs is to hash the description content along with the broker URL. Since SLAs are stored with their monitored data, We propose to generate keys for SLAs based on the hash value of the SLA ID only. The ID of an SLA consists of the service ID in addition to a unique SLA number if the service supports multiple SLAs. The ID of each supported SLAs will be stored in the USD itself as we previously mentioned. Thus upon discovery of USD matches, it is an easy task to retrieve their associated SLAs. When a monitoring agent submits a report, it submits the ID of the associated SLA as well. Using this ID, we can retrieve the full SLA history and update it with the new report (and the derived calculations of utility and feedback).

5.5 Summary

In this chapter, we elaborated on the design choices and tradeoffs for implementing the different parts and components of dirSaaS. We showed how the new dirSaaS prototype extends and improves the implementation of OSDA to achieve the new requirements of our system.

Chapter 6

Conclusion and Future Work

6.1 Summary of Contributions

This thesis describes a comprehensive framework for a SaaS service directory that allows personalized search and evaluation of functional and non-functional service parameters. As such, it introduces several features currently missing in current works in service description and discovery: a structured schema for describing functional and non-functional service description and quality parameters; an interface to accept service monitoring reports about previous service execution performance; a scalable directory of service information and feedbacks; and a personalized service recommendation engine that takes previous service performance in consideration.

The core of the service directory is the service description scheme, which uses simple, extensible schemas to describe both functional service capabilities and non-functional service quality guarantees. Along with easier parsing and processing of the service information, this structure allows simple, template-based creation of service description and queries, benefiting both service providers and service requesters.

The structure of the simple service quality schema serves as a basis for the second main contribution of this thesis: an approach for integrating the service management platform with third-party service monitoring agents who are contracted to monitor and submit reports of service performances of current subscribed service users.

Meanwhile, service providers benefit from being able to advertise the existence and features of their service offers using a single, eventually standardized template and to easily compare with other providers using the same template. This same template will be used by service requesters, making search easier and more comprehensible, and providing incentives for providers to outperform their competitors.

Another contribution of this thesis is the use of an innovative approach to storing and indexing service descriptions, as well as SLA guarantees and runtime reports. Each service description is stored using multiple identification keys that are generated by hashing over partial strands of its content. When key ranges are assigned to multiple participating directory (peer) nodes, the information is effectively replicated and can tolerate partial failures in parts of the storage nodes network. When searching, partial information in the query can then be matched to any corresponding key, gaining access to the full advertised record. SLA information is stored in a possibly different location than the corresponding service description, helping to avoid malicious attacks targeted against the sensitive and influential service monitoring information.

The main contribution of this thesis is the integration of the monitoring-based feedback and reputation mechanisms as the core for evaluating and selecting services. The generated recommendations are thus more objective and accurate than any user-based feedback system.

6.2 Future Directions

As with all large frameworks, there are a few major difficulties in the design and implementation of this service directory. We will discuss them here in the context of potential future work that would help resolve these difficulties.

One of the difficulties is that the service description and SLA schemas would require service providers to agree on common terminology and templates for their advertised offers to be compatible with each other. Once a comprehensive collection of service templates becomes available or standardized, developing service descriptions and SLAs would become much simpler, but the initial creation of such a collection would require a significant effort. Since the creation of such standards is far beyond the scope of this work, future work should study this problem and whether it can be solved in an efficient manner.

The design of the SLA information and monitoring storage introduces another difficulty. The current approach requires that all submitted quality reports should be stored to obtain all the necessary information for evaluation and recommendation. The SLA guarantees should be fully parsed and understood by the system to process this information. Future work should study if only summaries or abstracted information can be stored to reduce both storage and processing overhead. This problem is exacerbated when the frequency of monitoring reports is high or when a very large number of services exist. Solving the previous issue might also allow the system to be agnostic to the format or language used to represent SLA information, thus supporting more advanced SLA languages like the one proposed in [42].

The current proposed system assumes that service monitoring agents exist and can access the system to submit their reports. Thus, future work should investigate service and SLA monitoring and compare the existing works on this subject.

Finally, it would be worthwhile to address support for more sophisticated queries in the service directory, with a good analysis of what types of queries should or should not be handled in the global index or passed to the back-end database.

Bibliography

[1] Karl Aberer. P-grid: A self-organizing access structure for p2p information systems. In Carlo Batini, Fausto Giunchiglia, Paolo Giorgini, and Massimo Mecella, editors, *CoopIS: International Conference on Cooperative Information Systems*, volume 2172 of *Lecture Notes in Computer Science*, pages 179–194. Springer, September 4-7 2001. 17

[2] Omotunde Adebayo, John Neilson, and Dorina Petriu. A performance study of client-broker-server systems. In *CASCON: IBM Centre for Advanced Studies Conference*, page 1. IBM Press, 1997. 7

[3] Vikas Agarwal, Girish Chafle, Koustuv Dasgupta, Neeran M. Karnik, Arun Kumar, Sumit Mittal, and Biplav Srivastava. Synthy: A system for end to end composition of web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(4):311–339, 2005. 13

[4] R. Ahmed, N. Limam, J. Xiao, Y. Iraqi, and R. Boutaba. Resource and service discovery in large-scale multi-domain networks. *IEEE Communications Surveys and Tutorials*, 9(4):2 –30, quarter 2007. 11, 12, 17, 25

[5] J. Allard, V. Chinta, S. Gundala, and III Richard, G.G. Jini meets upnp: an architecture for jini/upnp interoperability. In *SAINT: International Symposium on Applications and the Internet*, pages 268 – 275, jan. 2003. 11

[6] Gostavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services - Concepts, Architecture, and Applications*. Springer, 2004. 7

[7] Danilo Ardagna and Barbara Pernici. Global and local qos guarantee in web service selection. In Christoph Bussler and Armin Haller, editors, *BPS: Workshop on Business Processes and Services*, volume 3812, pages 32–46, Sep. 5 2005. 13

[8] Magdalena Balazinska, Hari Balakrishnan, and David R. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Pervasive: International Conference on Pervasive Computing*, pages 195–210, 2002. 54

[9] Sean Bechhofer, Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl 2 web ontology language, 2009. 11

[10] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: service discovery and binding in structured peer-to-peer overlay networks. In *ACM SIGOPS European Workshop*, pages 140–145. ACM, 2002. 18

[11] Elizabeth Chang, Tharam Dillon, and Farookh K. Hussain. *Trust and Reputation for Service-Oriented Environments*. Wiley, 2005. 6

[12] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1. Technical report, W3C, 2001. 11

[13] Edith Cohen and Scott Shenker. Replication strategies in unstructured peer-to-peer networks. In *SIGCOMM: ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 177–190, Pittsburgh, Pennsylvania, USA, 2002. ACM. 17

[14] Andrew Conry-Murray. There's no such thing as a private cloud, 2009. 16

[15] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard P. King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer, and Alaa Youssef. Web services on demand: Wsla-driven automated management. *IBM Systems Journal*, 43(1):136–158, 2004. 11

[16] John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *IPTPS: International workshop on Peer-To-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, Mar. 7-8 2002. 3

[17] John Garofalakis, Yannis Panagis, Evangelos Sakkopoulos, and Athanasios Tsakalidis. Web service discovery mechanisms: Looking for a needle in a haystack? In *International Workshop on Web Engineering, in Conference Hypertext 2004*, 2004. 18

[18] Roger Grimes. Cloud computing is more secure than you think, May 2010. 17

[19] E. Guttman. Service location protocol: automatic discovery of ip network services. *IEEE Internet Computing*, 3(4):71 –80, jul/aug 1999. 11

[20] Bernhard Haslhofer and Wolfgang Klas. A survey of techniques for achieving metadata interoperability. *ACM Computing Surveys*, 42(2):1–37, 2010. 10

[21] Ching-Lai Hwang and Kwangsun Yoon. *Multiple Attribute Decision Making, Methods and Applications - A State-of-the-Art Survey*. Springer-Verlag, 1981. 13

[22] Audun Jøsang, Roslan Ismail, and Colin Boyd. A survey of trust and reputation systems for online service provision. *Decision Support Systems*, 43(2):618–644, 2007. 13

[23] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web. In *STOC: ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, United States, 1997. ACM. 42

[24] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003. 11, 47

[25] K. Kritikos and D. Plexousakis. A semantic qos-based web service discovery algorithm for over-constrained demands. In *NWeSP: International Conference on Next-generation Web Services Practices*, pages 49 –54, oct. 2007. 11

[26] N Limam and R Boutaba. Assessing software service quality and trustworthiness at selection time. *IEEE Transactions on Software Engineering*, 36(4):559 – 574, July-Aug. 2010. 13, 14, 26, 36, 37, 38, 39, 40, 41

[27] Noura Limam and Raouf Boutaba. Qos and reputation-aware service selection. In *NOMS: IEEE/IFIP Network Operations and Management Symposium*, pages 403–410. IEEE, Apr. 7-11 2008. 13, 14

[28] Noura Limam, Joanna Ziembicki, Reaz Ahmed, Youssef Iraqi, Tianshu Li, Raouf Boutaba, and Fernando Cuervo. Osda: Open service discovery architecture for efficient cross-domain service provisioning. *Computer Communications*, 30(3):546–563, 2007. 11, 12, 18, 24, 41, 45, 51, 54, 55

[29] Stephen Paul Marsh. *Formalizing Trust as a Computational Concept*. PhD thesis, University of Stirling, April 1994. 13

[30] E. Michael Maximilien and Munindar P. Singh. Conceptual model of web service reputation. *ACM SIGMOD Record*, 31(4):36–41, 2002. 14

[31] E. Michael Maximilien and Munindar P. Singh. Toward autonomic web services trust and selection. In *ICSOC: International Conference On Service Oriented Computing*, pages 212–221. ACM, 2004. 14

[32] Peter Mell and Tim Grance. The nist definition of cloud computing, 2009. 14

[33] Lik Mui, Mojdeh Mohtashemi, and Ari Halberstadt. A computational model of trust and reputation. In *HICSS: Hawaii International Conference on System Sciences*, pages 2431–2439, 2002. 13

[34] OASIS. Uddi version 3.0.2, 2004. 11

[35] OASIS. Reference model for service oriented architecture 1.0, 2006. 7

[36] Justin O'Sullivan, David Edmond, and Arthur H. M. ter Hofstede. What's in a service? *Distributed and Parallel Databases*, 12(2/3):117–133, 2002. 11

[37] M. P. Papazoglou and D. Georgakopoulos. Service oriented computing: Introduction. *Communications of the ACM*, 46(10):24–28, 2003. 7

[38] Franco Raimondi, James Skene, and Wolfgang Emmerich. Efficient online monitoring of web-service slas. In *SIGSOFT FSE: ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 170–180, Atlanta, Georgia, 2008. ACM. 13

[39] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *SIGCOMM: ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 161–172, San Diego, California, United States, 2001. ACM. 18

[40] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware: ACM/IFIP/USENIX International Middleware Conference*, pages 329–350, 2001. 18

[41] Brahmananda Sapkota, Laurentiu Vasiliu, Ioan Toma, Dumitru Roman, and Christoph Bussler. Peer-to-peer technology usage in web service discovery and matchmaking. In *WISE: International Conference on Web Information Systems Engineering*, volume 3806 of *Lecture Notes in Computer Science*, pages 418–425. Springer, 2005. 11

[42] James Skene, Franco Raimondi, and Wolfgang Emmerich. Service-level agreements for electronic services. *IEEE Transactions on Software Engineering*, 36(2):288 –304, march-april 2010. 11, 47, 60

[43] I. Stoica, R. Morris, D. Liben-Nowell, D.R. Karger, M.F. Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking*, 11(1):17 – 32, feb 2003. 42, 54

[44] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM: ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 149–160, San Diego, California, United States, 2001. ACM. 17, 18, 41, 42

[45] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. Wsol - web service offerings language. In *WES: Web Services, E-Business, and the Semantic Web - CAiSE Workshops*, pages 57–67, 2002. 11, 47

[46] Aphrodite Tsalgatidou and Thomi Pilioura. An overview of standards and related technology in web services. *Distributed and Parallel Databases*, 12(2/3):135–162, 2002. xi, 8

[47] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *Information Technology and Management*, 6:17–39, 2005. 18

[48] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer. Qos-based service selection and ranking with trust and reputation management. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *CoopIS: International Conference on Cooperative Information Systems*, volume 3760 of *Lecture Notes in Computer Science*, pages 466–483. Springer, Oct. 31 - Nov. 4, 2005. 14, 24, 36

[49] W3C. Web services architecture. Technical report, W3C Working Group Note, 2004. xi, 7, 9

[50] Jim Waldo. The jini architecture for network-centric computing. *Communications of the ACM*, 42(7):76–82, 1999. 11

[51] Giorgos Zacharia and Pattie Maes. Trust management through reputation mechanisms. *Applied Artificial Intelligence*, 14(9):881–907, 2000. 13, 14

[52] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z. Sheng. Quality driven web services composition. In *WWW: International Conference on World Wide Web*, pages 411–421. ACM, 2003. 14

[53] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004. 13, 14

[54] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41 – 53, jan. 2004. 18