Dynamic Web Service Discovery

by

Atousa Pahlevan
M.Math., University of Waterloo, 2006

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

Dynamic Web Service Discovery

by

Atousa Pahlevan
M.Math., University of Waterloo, 2006

Supervisory Committee

Dr. Hausi A. Müller, Co-supervisor
(Department of Computer Science)

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Issa Traoré, Outside Member
(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. Hausi A. Müller, Co-supervisor
(Department of Computer Science)

Dr. Alex Thomo, Co-supervisor
(Department of Computer Science)

Dr. Issa Traoré, Outside Member
(Department of Electrical and Computer Engineering)

ABSTRACT

Existing methods used for service discovery assume that the world is static, considering a predetermined set of attributes. As a result, current discovery techniques return many results that are irrelevant. Our approach to high quality service discovery improves the results' relevancy by considering dynamic attributes with values changing over time. Using this approach, we reveal structure from the data to satisfy the consumers' experiences.

Web service quality is a set of dynamic attributes used to rank services with similar functionalities. When picking a service to execute financial transactions efficiently, we might consider availability, reliability, response time, and transaction cost as quality indicators. Supporting dynamic attributes is a feature critical to providing exceptional quality service discovery. In addition, effective service discovery requires detailed context models that describe both static and dynamic features. The context takes into consideration the situation of the service, the operating environment, the users' circumstances, and their preferences. For instance, latency is an important issue in stock trading services with direct impact on revenue.

One of the main challenges in enabling dynamic service discovery is developing techniques and models to handle the novel aspects of the web service paradigm. This

challenge leads to a variety of research questions related to measuring, monitoring, or querying of dynamic attributes, while guaranteeing integrity and validity.

We outline an architecture framework called Static Discovery Dynamic Selection (SDDS) to gather and manage dynamic attributes considering both context and domain information at discovery time—augmenting static mechanisms. The architecture of SDDS defines individual components that collectively satisfy flexible and accurate service selection with a robust resource management approach capable of considering high-frequency data. Moreover, we devised a multi-criteria decision making algorithm that considers the knowledge domain and the user context, and accordingly, the algorithm returns a small set of accurate and reliable results.

As part of the SDDS framework, autonomic computing adds self-adaptability by taking highly dynamic context information into account. The impact of our method is demonstrated in an implementation of the model. We demonstrate that increasing the adaptability of the web service discovery by including context information provides a noticeable reduction in the number of results returned compared to static web service discovery methods.

We extend the proposed infrastructure to ascertain whether a particular service satisfies, at execution time, specific security properties. We introduce the notion of certified web service assurance, characterizing how consumers of the service can specify the set of security properties that a service should satisfy. In addition, we illustrate a mechanism to re-check security properties when the execution context changes. To this end, we introduce the concept of a context-aware certificate and describe a dynamic, context-aware service discovery environment.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank my PhD advisor, Dr. Hausi Müller, for his support, patience, and encouragement during my studies at University of Victoria. Thank you Hausi, for being my supervisor and for your advice and lessons. I will carry these with me throughout my life.

Special thanks are due to Dr. Alex Thomo, my co-supervisior. He has provided valuable advice for this research, enabling me to develop a better understanding of the subject.

I am grateful for the support from the Rigi group. You challenged me every step of the way and helped me to develop new ideas in this research.

I would like to thank many others who assisted me in different ways during the completion of my PhD, particularly the secretaries and staff who keep the Department of Computer Science running. A special mention to Wendy Beggs, and Nancy Chan.

I am indebted to my friends who made Victoria a comfortable place to work and live.

I offer my regards, and love to my family, in particular, my husband Jean-Luc. Over the course of my dissertation, our friendship blossomed to an engagement, and now you are my husband. Without the guidance and love of my family, it would have been impossible to pursue my work.

For my family

Chapter 1

Introduction

1.1 Overview

With the increasing proliferation of web services, selecting the best one for a particular need can be an overwhelming task. As such, it is natural to expect some automation of the process. A promising automatic service selection method for service discovery is to evaluate the suitability of each service with respect to extra attributes, in particular, attributes whose values change over time. This is quite challenging in practice, because dynamic attributes can be difficult to measure, may constantly fluctuate, be context-sensitive, and depend on environmental factors such as network availability.

According to World Wide Web Consortium (W3C): "A Web Service is a software application identified by a URI [universal resource identifier], whose interfaces and bindings are capable of being defined, described and discovered by XML artifacts and support direct interactions with other software applications using XML based messages via Internet-based protocols" [Sch02]. To ensure both interoperability and reduced costs, the focus of any operation needs to allow for discovery, integration, and communication inter- or intra-organizationally. Web service discovery is an eminently useful mechanism to facilitate such processes over the web. Moreover, web service standards are recognized as key enablers for interoperability between different systems.

Web service discovery is "the act of locating a machine-processable description of a web service-related resource that may have been previously unknown and that meets certain criteria. It involves matching a set of functional and other criteria with a set of resource descriptions. The goal is to find an appropriate web service-related

resource" [MvH04].

Service discovery has many applications, from finding devices such as printers on a local area network (LAN), to finding trading services such as e-commerce providers on a wide area network (WAN). To discover services, one can choose from different mechanisms, ranging from a network address or URI, using a decentralized peer-to-peer (P2P) based registry to a centralized registry.

The benefits of service discovery are: (1) encouraging reuse of existing capabilities; (2) promoting loosely coupled systems; (3) minimizing the cost and effort of programming through the reuse of standardized components and interfaces; (4) supporting heterogeneous environments.

Several factors are crucial for web service discovery. First, the discovery method needs to be aware of both the syntax and semantics of web services, since the service advertiser and requester might have different knowledge and perspectives of the same service. To resolve the lack of clarity with respect to service recognition, a semantic level of service is desired; that is, a computer-interpretable description of the service and the means by which it can be accessed [MSZ01]. The lack of semantics yields poor results, necessitating human intervention. In addition, assessing the quality of web services (QWS) is a primary step towards obtaining high-quality results. QWS is a set of non-functional, domain-specific attributes that contain both application-level (e.g., throughput or latency) and network-level properties (e.g., availability). The quality of a web service at discovery time depends on the overall quality of mentioned attributes. Current discovery methods, however, do not focus on QWS. When many web services deliver similar functionality in the same domain, the QWS attributes are key features in distinguishing among services with similar functionality. In those situations a web service should be selected to satisfy the functionality required through the desired QWS attributes. Thus, we evaluate and measure service quality at both registration and discovery time.

To enable high-quality discovery, standards are required to integrate quality attributes. Moreover, to provide relevant services to fulfill a consumer's request, we must be able to discover and select a service considering its operating environment, and its user's situation and preferences. We refer to the above mentioned attributes as dynamic service attributes. The goal of studying these attributes is to improve the accuracy and integrity of service discovery. We conclude that the involvement of these aspects in the discovery process is a vital factor in providing a high quality of experience (QoE) to the user.

The following section presents a scenario used throughout the dissertation to illustrate the need for monitoring QWS attributes.

1.2 A Service Selection Scenario with Dynamic Attributes

We illustrate the crucial role of dynamic attributes for finding suitable services by considering the following hypothetical scenario. There are dozens of telescopes accessible through web services around the world. To a user interested in photographing a particular star, only a subset of these services are of particular interest, based on several functional attributes such as which telescopes may be oriented towards the star. Furthermore, the user may not have a service-level agreement that permits her to use a specific telescope. By considering all such attributes, one can reduce the entire set of telescopes to a small subset which is functionally equivalent (e.g., acceptable mirror size). This is referred to as static, functional selection. After functional selection, the elements of the subset need to be ranked and possibly further pruned with respect to dynamic attributes. The values of all these attributes fluctuate over time, potentially dramatically, which complicates the matter of selecting the web service most likely to deliver the highest quality photography.

Given four possible telescope web services as depicted in Table 1.1, the goal is to select the best candidate for the purpose of taking a picture of a specific star. An important dynamic attribute specific to the domain of telescope services is the visibility of the sky, which varies throughout the day (sometimes drastically as in the case of T3), depending on the weather and season, and would likely be measured using a separate data source, such as the observatory's weather station. Another important dynamic attribute for telescopes, one of varying interest to different users, is composition: the number of objects captured in a photograph. On certain days, particularly those in which the availability of the telescope is limited due to, say, maintenance, it may be impossible to determine a value for this attribute. For example, perhaps no photographs have been taken, or the quality of the photographs may be too poor to calculate their composition reliably.

This scenario highlights the necessity of dynamic attributes to improve web service discovery. Consider the consequences if one ignores that the parameter values can fluctuate and merely assumes that all functionally-equivalent services are equally

Table 1.1: An example set of telescope web services with multiple dynamic QoS attributes, as measured on four consecutive days.

		Days			
Telescope	Context/QoS Attributes	1	2	3	4
T1	functional	Y	Y	Y	Y
	availability(%)	20	40	50	60
	user preference/contrast	8	9	8	5
	time/dayLight	8	9		5
	user preference/composition	8			5
	latency(ms)	6	2		3
	weather/visibility(%)	70	90	90	80
	geographical position/viewing angle	8	8	4	5
T2	functional	Y	Y	Y	Y
	availability	85	93	65	89
	user preference/contrast	8	9	4	5
	time/dayLight	8	9	5	5
	composition	8	9	9	5
	latency	9	8		9
	weather/visibility	90	80	80	90
	geographical position/viewing angle	8	6		8
T3	functional	Y	Y	Y	Y
	availability	80	70	70	40
	user preference/contrast	8	9	4	5
	time/dayLight	8	9	4	5
	composition	8	9		5
	latency	6	7	6	8
	weather/visibility	10	10	70	90
	geographical position/viewing angle	1	1	3	7
T4	functional	Y	Y	Y	Y
	availability	30	60	30	60
	user preference/contrast	8	9	4	1
	time/dayLight	8	9		1
	composition	8			3
	latency	5	7	4	1
	weather/visibility	70		70	50
	geographical position/viewing angle	8	9	9	9

valuable. The user must manually filter the results, even extreme ones as in the case of T2 and T3 in Table 1.1, where one service outperforms or matches another on every measurement for every metric. This leads to user frustration and limits the adoption of the system.

Despite this obvious need, the existing literature does not adequately address this problem. Techniques come in three flavours: those that do not consider dynamic attributes at all, those that measure the dynamic value in advance and reuse them during service discovery, or those that take a snapshot of a value. In the former cases, the real potential in varying attribute values is squandered, and they are treated as static attributes. Current mechanisms are primarily based on string similarity. These mechanisms are of limited use because the service descriptions on which the similarity is calculated on are short, often ambiguous, and sometimes syntactically erroneous. In the latter case, services are misrepresented by temporary fluctuations in values or harshly penalized because the mechanism for obtaining those values is unsuccessful. For instance, the value of the service availability of T3 on day 4 is low; however, the overall availability of the service was high on days one, two, and three. To rate T3 at day 4, we need to consider the quality of T3 during the previous days to avoid misinterpreting. This can also assist in evaluating a service when the values are not immediately available, such as T1 latency on day three. Similar situations will inevitably arise in a practical implementation.

As demonstrated, to satisfy the user's need for timely, well-chosen web services, new ideas are needed. We present a novel technique for monitoring, integrating, and establishing reliable values for dynamic service attributes that can withstand the fluctuations and properly satisfy user requirements.

So far, we have introduced different types of attributes and their importance in web service discovery through a practical example. Now we study these attributes and review the principles of web service technologies to provide the reader with a better understanding of the contents herein. We give an overview of the history of services and the way in which services can improve customers' experiences as well as stabilize businesses. We also introduce the evolution of enterprise applications and the problems that could be solved using web services.

1.3 Service Attributes

The nature of a web service is identified through a set of attributes that characterize the service's ability to carry out a specific task. They can be non-quantitative and expressed in textual terms, or qualitative represented by values. We distinguish between static and dynamic service attributes. Static attributes describe the basic capabilities (termed functionality) of a service. Generally, the values of such attributes are fixed unless they are upgraded periodically. Non-functional attributes determine the quality of the service and these values are updated frequently. Stock trading is an example of a service that performs stock transitions, and a "waitingTime" attribute describes the quality of the service.

Research in dynamic attributes remains a demanding discipline wherein the service quality is all that matters, because of the challenging nature of measuring and monitoring these types of attributes. A service that fails to perform the functionality is identified as a *malfunctioning service*, whereas low values of the non-functional attributes represent a *poor quality service*.

Each domain has its own set of attributes; however, some of the attributes are cross-domain. For example, in both the retail and flight reservation domains, the "price" attribute is identical whereas the "type of flight" (e.g., non-stop or direct) is specific to flight reservations. Distinguishing the attributes that apply to a domain is nontrivial and should be decided by a community of experts in each domain. Consider the telescope example from Section 1.2. Figure 1.1 depicts an abstract classification of service attributes for telescope services. We classify the service attributes into two major groups in our "service attribute taxonomy": static and dynamic. Static or fixed value attributes express the principal capabilities of the service such as "mirror size" in our telescope example. The size of the mirror is almost always fixed unless it is upgraded. Dynamic attributes or range-valued attributes are those that can change over time, such as weather conditions in the same example. Dynamic attributes are classified into two sub-categories: measurable and unmeasurable. These subcategories are associated with a set of entities to describe details such as their units and instances. For example, in the telescope attribute taxonomy (cf. Figure 1.1), WeatherInfo belongs to the telescope domain and is categorized as a dynamic attribute. It has instance properties (hasInstance: Y) such as visibility, and it is measurable (isMeasurable: Y). This taxonomy requires concrete knowledge of all possible attributes in each domain. The weather instances can be different for each domain or have more

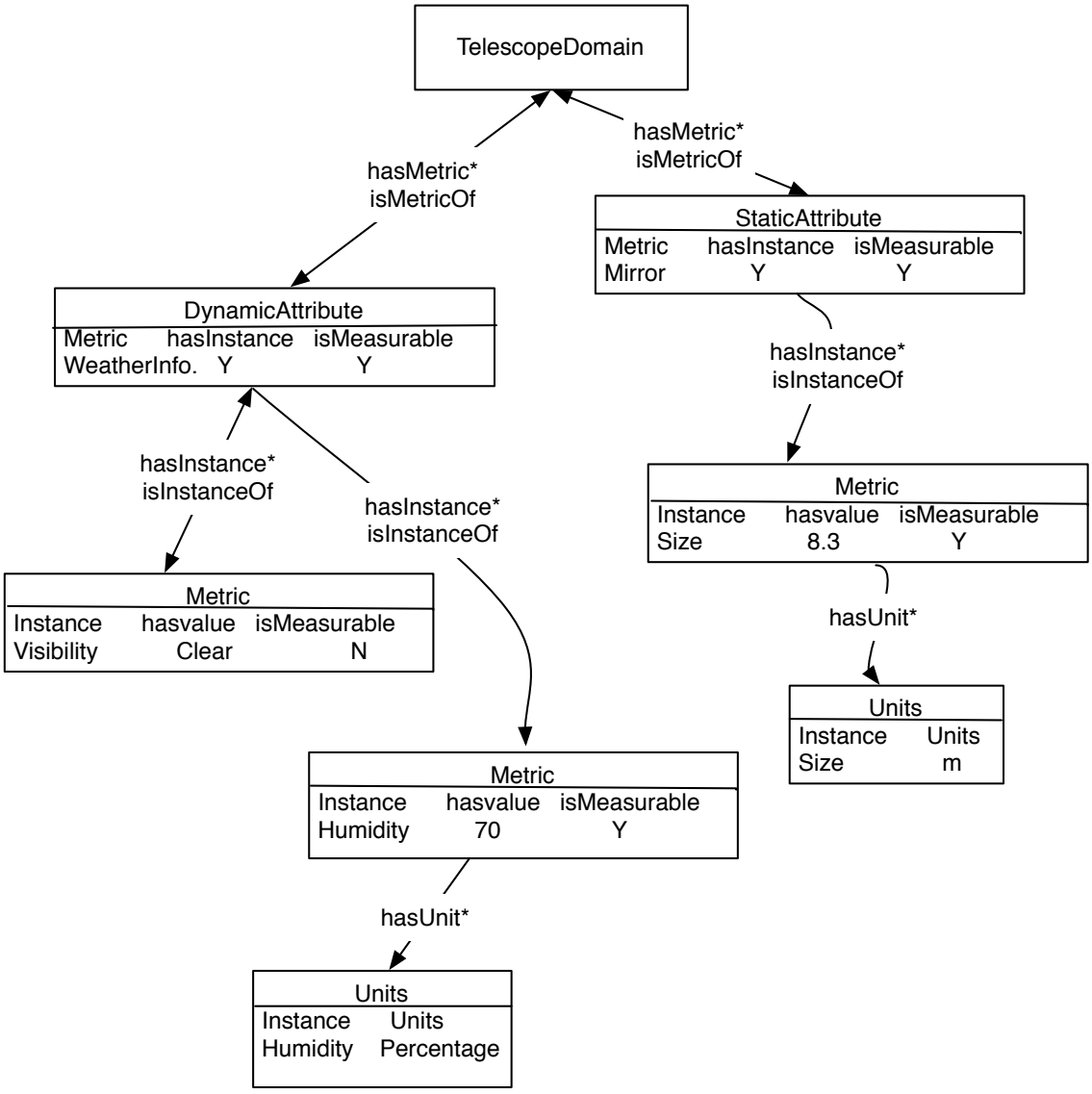


Figure 1.1: Basic telescope attribute taxonomy

instances such as windspeed. To avoid ambiguities in service attributes descriptions, the attribute taxonomy is designed to document semantic descriptions of web service attributes for all domains.

In the following sections, we explore the advantages of implementing web services for both developers and consumers to show why this method has great potential.

1.4 Siloed vs. Service-Oriented Architecture

The main motivations behind implementing web service discovery are enabling reusability to improve the effectiveness of a software system and providing interoperability in a heterogenous environment [FP12]. To clarify, we review the two main architectures used in developing a distributed software system.

"Software silo" is a common technology wherein several subsystems of an application exist behind a single application program interface (API) that can access each others' data—applications implemented by Amazon in 1995. Then in 2002, Amazon developed a software system with added features where components acted as interoperable services. The published interface is the only permitted communication mechanism and is exposable to the outside world; today we call this a service-oriented architecture (SOA). SOA is significant because of its data availability through an API and its interoperability. Although initial implementation of SOA requires more effort compared to a siloed services, the payoff comes in the form of reusability [FP12] .

To give a concrete example, suppose we want to provide a supply chain service both as a silo and an SOA. The service includes the following systems: order, shipment, finance, user, and procurement. Figure 1.2 shows the silo architecture where all subsystems are inside a single external API (supply chain service), and can internally access all the data. In the SOA version of the same service, the interactions among the internal subsystems (services) are only through the published APIs. Simply, the SOA supply chain service is the composition of the services that can be reused for different purposes (cf. Figure 1.3).

SOA was a software revolution that has been utilized by many software companies, including Facebook in 2007, when the Facebook platform was launched. Using this architecture, developers can create applications (e.g., New York Times) that interact with the main features of Facebook (e.g., which articles your friends were reading).

Up to this point, we highlighted how dynamic attributes can improve user experiences (i.e., QoE). We have also considered specific advantages of SOA over siloed architectures to facilitate interoperability as well as reusability across organizations. The following sections provide support from literature in other fields for utilizing the service concept in computer science. Specifically, services have already shifted from a manufacturing application to one that applies to business in computer science. The sections that follow expound this argument in a series of discussions.

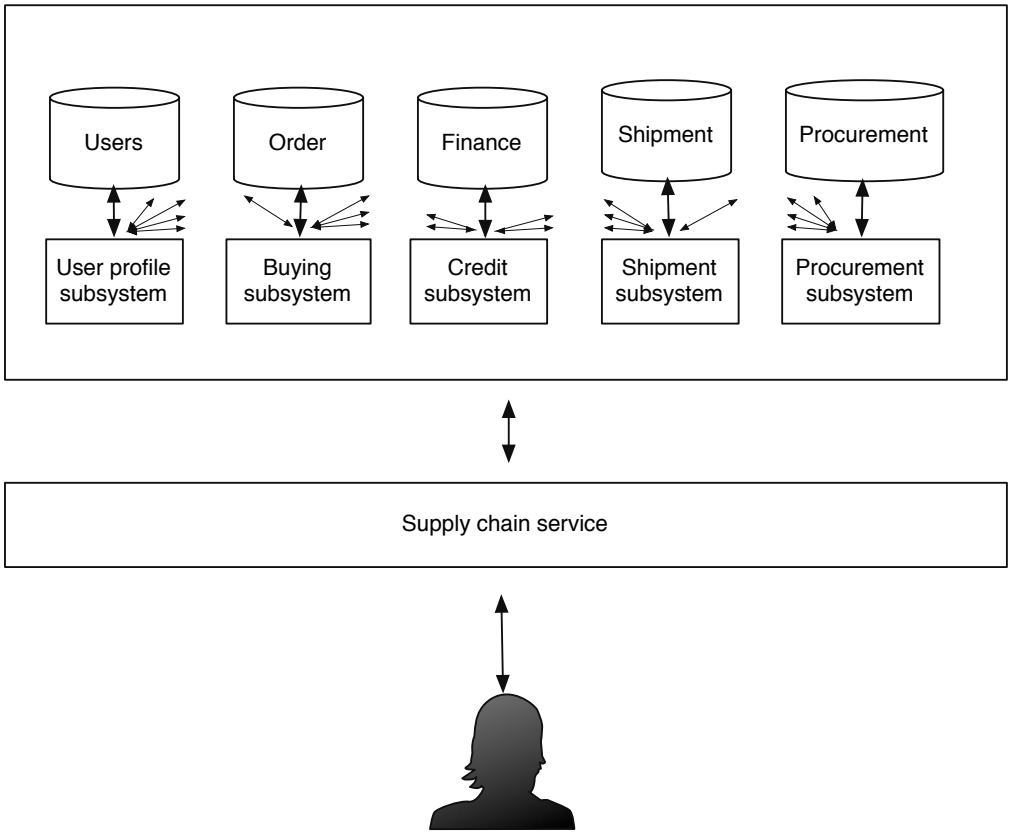


Figure 1.2: Silo of a sample supply chain service

1.5 A History of Services

Service is a broad term, and until recently referred to providing value to customers in contexts outside of computer science, such as maintenance, repair, or insurance. The word "service" comes from the latin *servitum* which means "slavery". The benefits of services were discussed by Vandermerwe and Rada in 1988 where "servitization" is defined as the process of creating value by adding services to products [VR88]. A combination of globalization, technology, and distribution in industry has motivated traditional manufacturing companies to move towards a servitization approach. Currently, the shift from manufacturing to providing services can be found in business-to-business (e.g., IBM), business-to-consumer (e.g., Apple), and whole industries (e.g., software-as-a-service, UPS). Simply, by providing a coherent bundle of goods, services, support, knowledge, and self-service, companies can foster strong customer centricity, increasing revenue and keeping businesses afloat. "Create wealth by creating value" has become the modus operandi for businesses.

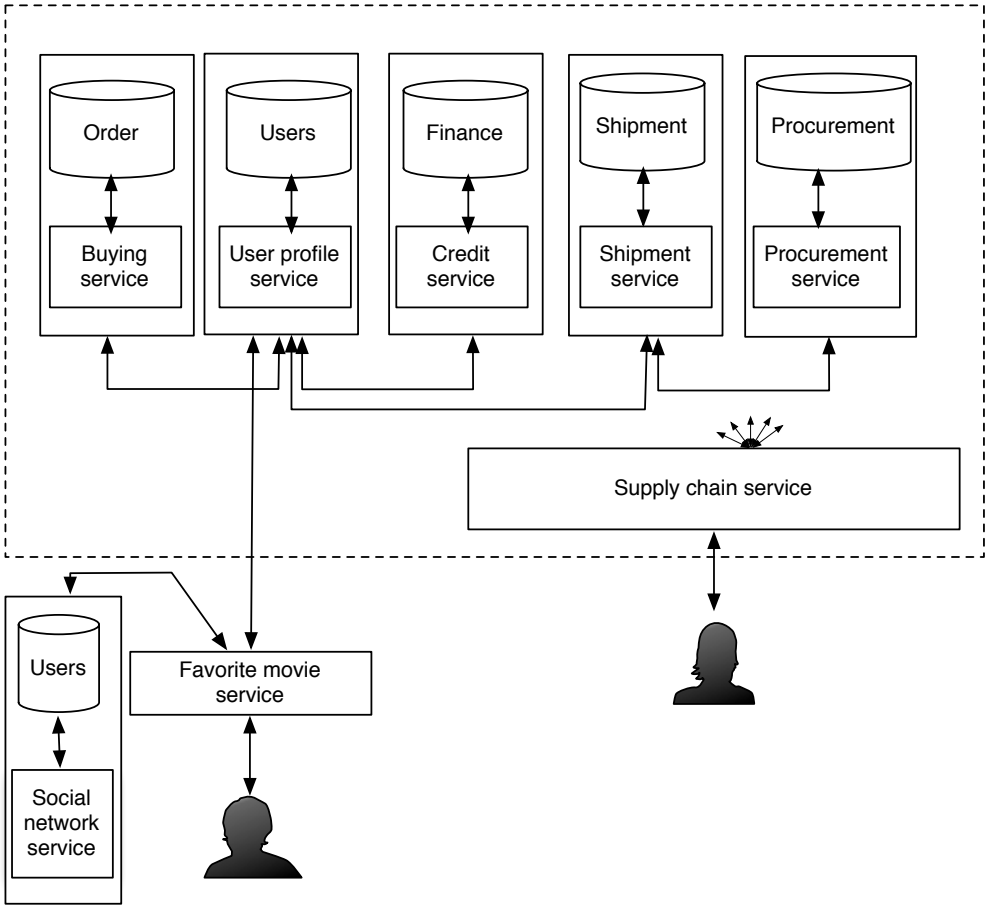


Figure 1.3: SOA of a sample supply chain service

Servitization leverages benefits in four aspects: customer loyalty, differentiation, stability, and corporate image [Mal06]. Firstly, by developing services in addition to physical products, a long-term relationship can be established between consumers and service providers. For instance, trusted suppliers can use their influence to offer new services to their customers. Secondly, companies can retain existing customers or encourage new ones by providing value-added services. This is referred to as product differentiation. Thirdly, offering services such as maintenance can generate stable income and increase companies' turnover and margins. Finally, providing services can associate a company's image with high quality products.

Malleret has done several studies to analyze service profitability by interviewing a number of firms that operate business-to-business (B2B) services [Mal06]. His reports indicate a higher profit margin from service revenue in some sectors, and more stable revenue from services in all sectors [Mal06]. They studied the cost, price,

and value of services provided by these firms. The results demonstrate that firms that learn how to sell, deliver, and bill for services can generate regular income. More specifically, this study details the aspects (e.g., actions and conditions) that affect service profitability. When firms offer services to their customers, if these create value for customers, they also create value for those businesses. Thus customer experience is essential to service profitability (*customer satisfaction*). Two important aspects need to be taken into account: first, a business can differentiate itself by producing the service at a competitive level of quality (*service quality*). Second, businesses must recognize that various customers may have different priorities and preferences (*customer preferences*).

Vargo et al. discussed two mindsets to motivate the transition from goods to services [VL08]. In the service dominated approach, authors introduce the service conceptualization as the most critical distinction between the goods- and service-oriented approaches. They represent the differences as shifting from "operand resources—tangible, and static resources to operant resources—usually intangible, dynamic resources". Accordingly, services should be considered as a product in their own right, due to their dynamic nature, rather than be thought of as inferior to the tangible goods developed by a company (*service dynamic nature*).

Furthermore, in complex and specialized markets such as the high-tech and pharmaceutical industries, third parties have been recognized as crucial consultants that know suppliers and can alleviate traditional business barriers [VR88]. The role of an *intermediary* to facilitate the interaction between companies (*service provider*) and customers (*service consumer*) for highly specialized markets, known as know-how [VR88], has made it possible to promote servitization.

1.6 The Focus of the Thesis

Without the ability to distinguish between available services using dynamic quality metrics, all services with the same functional attributes are considered to be equal. In the context of web service discovery, the research questions are *why, what, when*, and *where*, but the dynamic attributes also need to be included in service search and selection. Dynamic attributes have been discussed and categorized in several previous studies in the web service discovery domain. Following the W3C guidelines, our research is mainly focused on metrics in the categories represented in Figure 1.4 [LJL+03]. Here, the quality of service attributes fall into five main groups,

performance, dependability, application-specific, transaction support, and *security* attributes. Each group contains several metrics [KP09]. Here, only common and critical metrics are illustrated.

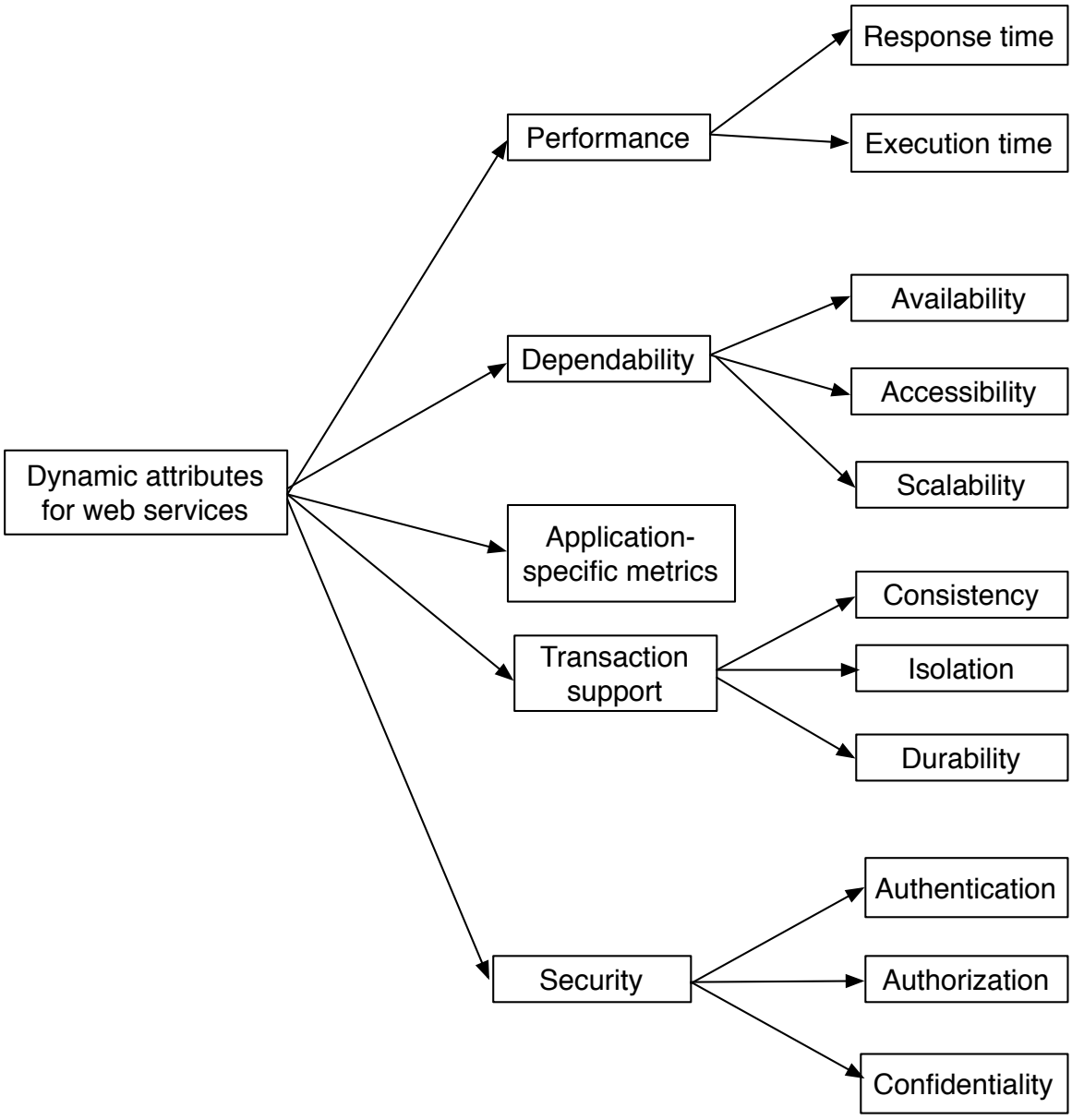


Figure 1.4: Criteria of dynamic attribute

The performance category includes *response time*, *latency*, and *execution time*. Execution time is the time taken for a service to process a request. Latency is identified as a composite attribute computed from the execution time and *queue delay*

time. Queue delay time is the time required to execute a service request. Response time is the overall time required to complete a service request. It is identified as a composite attribute since it is computed from latency and network delay.

Dependability is another group of QoS attributes containing the attributes *availability*, *accessibility*, and *scalability*. Availability is the probability that a service is up whereas accessibility is the availability of a service during a particular time period. Scalability is the ability of a service to handle more operations during a given period.

Application specific metrics refers to a set of attributes that apply to all services of the same domain. For example, a telescope service will involve attributes such as the *telescope mirror size* which is specific to telescope service, and the *weather domain* which may apply to other types of web services. Thus, certain attributes will be cross-domain, such as weather domain attributes. In addition, attributes belonging to different applications can have different meanings in separate domains. Determining application-specific attributes that apply to a particular domain is nontrivial, and needs be decided by domain experts.

Transaction support is composed of attributes that together guarantee to maintain the integrity of the data being manipulated in a process (i.e., service discovery process). For instance, the integrity of data is identified as *consistency*, individual transaction is specified as *isolation*, and data persistency is defined as *durability*. This group of QoS attributes is crucial in distributed transactions in web service based environments.

Security is critical when services are delivered over the public Internet. Organization or person verification, access control evaluation, unauthorized information dissemination assessment, and malicious access to data protection are specified as *authentication*, *authorization*, *confidentiality*, and *encryption*, respectively. Managing these types of attributes is different from the rest due to required techniques to evaluate and assure them.

1.7 Research Objectives

The main objectives for our study are:

- Investigate how to improve the accuracy of web service discovery using dynamic attributes

- Develop a taxonomy of dynamic attributes for the purpose of web service discovery
- Develop an integrated framework for static and dynamic web service discovery
- Develop self-adaptive web service discovery mechanisms
- Develop algorithms for computing attribute values in the presence of fluctuations and missing samples
- Tailor the framework for specific dynamic attributes such as security.

1.8 Research Questions

Our main research questions are:

- Why do web services have to be searched based on "what they do" and "how they do it"? Would it be inadequate to search for web services only by a keyword or by lexicographically matching service descriptions?
- Why does the suitability of each service need to be determined with respect to extra attributes—dynamic attributes? Is there a way to describe web services in a more quantitative manner?
- What type of dynamic attributes can be monitored and how are they distinguished at runtime? Is it possible to monitor dynamic attributes at runtime during service discovery?
- When and how can these attributes be evaluated in the web service discovery process? How can we specify the evaluation techniques for these attributes, in different contexts, at runtime?
- Why is evaluating web services based on dynamic attributes challenging? Are there any aspects that need to be taken into account for appropriate service selection?
- With regards to the above-mentioned research questions, what is the appropriate matching algorithm for quality web service discovery? Since these attributes potentially fluctuate and are context-sensitive, how we can avoid misinterpretation because of temporary fluctuations of these attributes?

- With enabling on-the-fly service selection, and exposing the service functionalities over the Internet, how can we avoid the security and reliability risk compared to off-the-shelf software systems? How can we consolidate the security in web service discovery, and how can we alleviate the security risk at runtime, in a SOA environment?

In this dissertation, our main contribution is service selection based on dynamic attributes to improve QoE. In the process of web service discovery, QWS information should be used since the user is vulnerable to inaccurate service selection. Regardless of whether the web service is being used for business or consumer interactions, and which dynamic attributes are being identified, an enhanced process that accommodates efficient monitoring and measurement mechanisms based on QWS are eminently useful. Figure 1.5 represents an abstract quality web service discovery process.

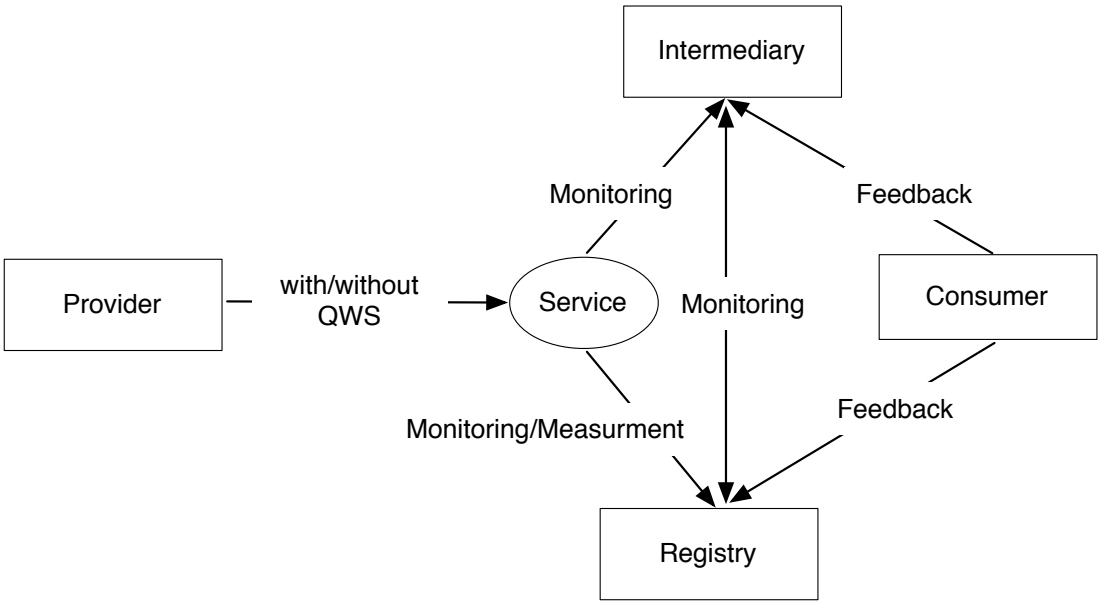


Figure 1.5: Quality web service discovery process

To achieve this, we propose a dynamic web discovery framework that can act on these types of service attributes during the process of publishing, searching, and locating web services on the network. The objective is that the process could handle multiple non-functional attributes simultaneously. These attributes are context-sensitive and need to be recognized and measured accordingly. Therefore, there is a need to develop an attribute recognition process along with the service selection that can decide whether a service is relevant for a given context. Considering these crucial

factors in our research, we show how to improve the quality of service discovery and subsequently QoE.

Moreover, security properties should be used as constraints over service selection. The effort of modelling, measuring, monitoring, and assessing these properties is often perceived as superfluous due to the contextual nature of their interpretation. Services returned from discovery mechanisms that ignore these aspects are often unreliable, a major problem that devalues the search results. We introduce a security assurance mechanism that enables service consumers to specify the desired set of security preferences that services must satisfy, thereby providing a mechanism to re-certify them at runtime when the context changes.

1.9 Dissertation Outline

This dissertation is divided into nine chapters.

Chapter 2 introduces the general techniques and concepts in the field of web service discovery. We also review web service discovery architectures.

Chapter 3 explains our research methodology.

Chapter 4 presents our approach to improve the quality of dynamic web service discovery regarding "transaction support" metrics. We introduce an architectural framework called static discovery dynamic selection (SDDS). We evaluate the notation of data integrity in SDDS where the transactions involve more than one component in a web services environment.

The SDDS approach and mechanism to examine the application-specific attributes are described in Chapter 5. We demonstrate how we can automatically monitor and measure these attributes in SDDS.

Chapters 6 and 7 document the implementation of our proposed approach by demonstrating the impact of using these attributes to increase the quality of web service discovery. In particular, we introduce our proposed algorithm—dynamis—to evaluate the web services at runtime.

Chapter 8 represents the mechanism to monitor and evaluate the "security" properties during service discovery.

Finally, Chapter 9 provides a summary, conclusions, and future work

Chapter 2

Background

2.1 Towards SOA with Web Services

Services can be wrapped into a form that can be utilized to promote servitization of software systems over the Internet, namely web services. A web service is a software system designed to support interoperable machine-to-machine interactions over a network. Web services have been adopted by the majority of Fortune 500 businesses, and have become a critical concept for business people [GDD+05]. In business, the web service approach is about integrating application functionality within or across organizations. This integration helps to reduce costs and increase quality without affiliating with any specific business partner. Web services have become ubiquitous as integration infrastructure was upheld by a majority of organizations due to ease of integration, yielding cost savings. Furthermore, when a better service is offered (based on quality, reliability, or price) from other suppliers, it can be substituted. This process is called dynamic business integration, which ultimately provides higher consumer retention and overall service quality.

Web service technology has also been developed to enable service outsourcing over the web. For instance, a system could be composed with outsourced services across multiple organizations using software-as-a-service (SaaS) and service-oriented architecture (SOA) paradigms [KR04]. The practice of outsourcing to make use of high quality services automatically is less common and needs significant effort.

We argue that the financial risks associated with outsourcing services using conventional web service technologies are high due to ignorance of vital aspects. A low quality choice of outsourced services degrades the quality of services and generally

lessens consumer trust. Here, one of the emerging research areas is dynamic web service discovery to enable service consumers to locate and select appropriate web services, based on both static and dynamic attributes and consumers' preferences, as well as improving service outsourcing and migration towards SOA. Primarily, the UDDI [CHvRR04] web service registry has been promoted by companies such as IBM, Microsoft and SAP to register and expose service functionalities to consumers. Unfortunately, there is no mechanism in place to evaluate and periodically review the correctness of web service information. Studies indicate that between 63% [KCO05] and 67% [FK05] of the registries are not valid due to incomplete, incorrect, duplicated, or outdated web service descriptions (WSDL) [CMRW07]. It has been reported that more than 52% of services have less than 20 words in their text description. On average, each operation has been described in 10 words [FK05], which exhibits that web services are not well-documented.

Moreover, there is no mechanism to verify the non-functional attributes of a web service such as availability and response-time, due to limited keyword search mechanisms. The statistics from two locations, IBM and KAIST, on UDDI determine that about 16% of the valid web services (63%-67%) are down and 96% of live web services respond in under two seconds. Thus approximately 45% of services are operative [KCO05], based on the principal requirements of availability and response time. Here, one observation is that even if text description and text retrieval were sufficient for service discovery, the web service registry is not able to accomplish more than a standard search engine [FK05].

Before we delve into details in this chapter, we include fundamental mechanisms for web service discovery (e.g., architectures and standards) that are useful for understanding the concepts underlying the technology. To make these mechanisms more understandable, we will show how the Telescope example (cf. Chapter 1.2) uses technologies like WSDL, UDDI, and SOAP.

2.2 B2B Integration with Web Service

Before web services, Business to Business (B2B) integration had been introduced to integrate heterogeneous systems and automate business process across organizations [ACKM04]. A successful example of B2B integration is Ariba and similar broker companies that facilitate binding and routing messages among the services provided by different companies. The broker tackles the integration of heterogeneous systems

that make the conventional middleware mechanism inappropriate for B2B exchange. Lack of support from software vendors for the broker's defined format and communication protocol have limited this solution. Notably, trust-related issues between interacting components that undermine the middleware system for B2B integration remain a significant hurdle. Another example of B2B integration is the EDIFACT standard that is being used by US retailer WalMart. Such mechanisms are rarely used due to the lack of standards, suitable infrastructure, and intense hardware and communication requirements that make the system difficult to maintain, reproduce, and adopt. For easy and inexpensive application integration, web services are broadly accepted by organizations [GDD+05]. We explain the contribution of web services to resolve the limitations of traditional mechanisms (e.g. middleware), and to allow flexible application integration as follows.

2.2.1 Service-Oriented Infrastructure

In service oriented infrastructure, companies expose their functionalities through a published interface that can be invoked by other applications. Here, the difference between web services and the middleware service is the feasible invocation of services on the web and across organizations. Thus, the web service concept has become popular and applications can be built as a loosely coupled set of services that can be published, discovered, integrated, and communicated with over the web. Early in Chapter One we introduced a pattern known as SOA. SOA is an architecture in which all business tasks and processes are designed as services which can be invoked through the service's interfaces. In this paradigm, the focus is on building services with well-defined interfaces rather than building a one-purpose program. The services are integrated at interface level in multiple business contexts, not at implementation level, to guarantee flexibility and reusability.

The SOA architecture contains three components: a service provider, a service requester, and a service registry. The service provider creates a service description (i.e., WSDL) to publish in service registries (i.e., UDDI). The created service is deployed in the network, and becomes accessible. The service requestor finds a service description published in the registries. Finally, the service registry advertises the published service description, and allows the service requestor search on its directory of published service descriptions. Figure 2.1 depicts the SOA components for the telescope example.

In this Figure, Atousa is a service requestor who is interested in finding the observatories that provide the Celestial Photography Service in order to acquire a photo of Mars. The service registry already has Mauna Kea in its directory of service descriptions that provide such a service (i.e., functionality). Mauna Kea's Celestial Photography Service is invoked by Atousa through three operations among the SOA components: publish, find, and bind. The publish operation is the contract between service registry and service provider (e.g., Mauna Kea) wherein the service descriptions move to the registry directory structure. The find operation is the contract between service requestor (e.g., Atousa) and service registry where the requestor states her search criteria and the registry matches the criteria against its directory. Mauna Kea is the service that has been found based on Atousa's requested criteria (i.e., functionality). A naive approach for the find operation is a simple HTTP GET request that returns all services published in the registry. However, a favourable find operation is to discover more relevant results. Finally, the bind operation allows Atousa to invoke Mauna Kea's Celestial Photography Service. So far, the key feature of all SOA operations is the service description that publishes, retrieves, and invokes. Notably, most existing mechanisms provide a service description based on just the functionality of the service—what does the service do? Accordingly the implemented SOA operations are designed to handle publish, find, and bind based on the static description.

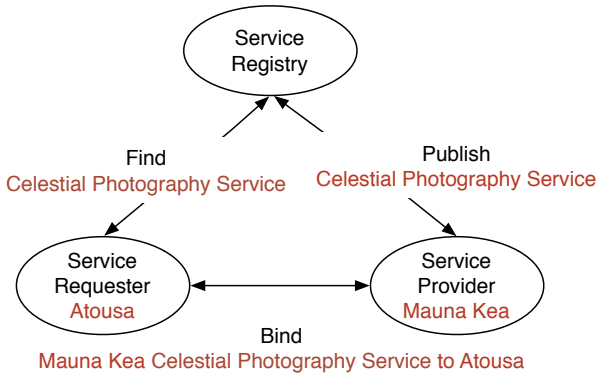


Figure 2.1: Service-oriented architecture

2.2.2 Communication Protocols

Another aspect to motivate B2B integration with web services is to address the issues in interacting with new organizations in a global business environment where cost,

scalability, and flexibility become hurdles. A service-oriented web-based approach (i.e. web services) eliminates these issues through offering services with a specific protocol over the Internet [Gan]. The adoption of the Internet as a communication tool already accessible to all businesses is encouraging.

2.2.3 Standardization

Although having service-oriented architecture and appropriate communication protocols are essential towards heterogeneous application integration, they are not sufficient. The attempt to standardize languages and interaction protocols in web services is a vital aspect that has been initiated by small companies and is adopted and improved upon by consortia such as OASIS and W3C. The technologies that aid SOA systems in achieving application integration over the Internet include WSDL, UDDI, and SOAP which we cover in detail in this chapter.

2.3 Reviewing the Telescope Example: Celestial Photography Service

Web service technologies and standards, we review the telescope example throughout this chapter. Celestial Photography Service is a new service that has been offered by several observatories as part of the functional attribute (cf. Table 1.2). The objective is to provide a mechanism to people ordering photos of celestial objects and events. To develop on this idea, the observatories utilized distributed computing and networking technologies. For instance, Mauna Kea created its own website so that users can make orders and stay up to date. Now, Mauna Kea is happy with its electronic Celestial Photography Service business; however, they recognize as the number of customers grow, manual tasks will limit their success. In this case, they recognized that application integration allows time and cost saving for such processes as answering inquiries, and shipment requests. They contacted Atousa, a contractor from the RIGI lab at the University of Victoria. RIGI specializes in web services solutions. After several meeting between Mauna Kea and RIGI, they were convinced to hire Atousa.

In the rest of this chapter, we explain how observatories (i.e., Mauna Kea) use web services technologies to increase efficiency, and subsequently improve QoE.

2.4 Web Services Technologies

A majority of the current web service discovery mechanisms are based on three main components: service providers, service consumers, and service registries. This section focuses on these components by explaining web service technologies for service description (WSDL and OWL), discovery (UDDI), and interactions (SOAP). We also outline some of the extended versions of these technologies which have been implemented to address new inquiries.

2.4.1 WSDL: Web Service Description Language

WSDL was created by Microsoft, IBM, and Ariba and plays a major role in web service architecture by confirming interoperability of the service description. It is a generic service description that can be used as a wrapper over existing ones. This is necessary because existing B2B standards (e.g., EDI, SWIFT) supported with expensive infrastructures cannot be quickly discarded and replaced.

WSDL is an XML-based language employed to describe services, in particular, their interfaces [CMRW07]. The service specifications are used for describing the abstract functionalities and concrete details of a service; enabling us to separate the concrete details of service description from the abstract functionalities offered by a service. The concrete details describe *how* and *where* certain functionalities are offered, while abstract functionalities explain *what* the service offers. WSDL describes web services through the following major units [CMRW07], which are more thoroughly expanded in Appendix A:

- *Types*: a type mechanism to aid the correct interpretation of the exchanged data over communication. The type system basically is an XML Schema and allows the elaboration of a specific type system.

- *Messages*: abstract definition of the transmitted data that is exchanged between the service provider and requestor. A message consists of logical parts; each part is associated with a type.

- *Operations*: exchange of messages are described as operations; the interactions can either be synchronous or asynchronous.

- *Port types*: collection of abstract operations; each refers to an input and an

output message. Yet, the lack of a concrete specification of a service that implements the port types makes the WSDL abstract.

The rest of the WSDL's description provides a concrete level of specification.

- *Binding*: concrete protocol and data format specification of all operations and messages for the specified port type.
- *Ports*: an implementation of port type, which contains all of the concrete details needed to interact with a service, such as the URI through which the implementation of the port type can be accessed.
- *Services*: collection of ports. Ports that are available at the same address are generally grouped in the same port. Another way of grouping ports is storing a different binding for the same port.

The advantage of the separation of abstract and concrete specifications in WSDL is the usability of the abstract interface, since different services can use different bindings available at different addresses. That being said, there are improvements needed to WSDL to support more aspects, such as describing QWS attributes [KP09].

2.4.2 OWL-S: Web Ontology Language for Services

OWL is designed to specify ontologies that form classes, properties, and their relationships for a specific application domain. A given application can use these ontologies to discover implicit and unknown facts through parsing the ontology and create a list of axioms that are expressed as a resource description framework (RDF) statement. The objective behind this method comes from the field of knowledge representation (KR) in artificial intelligence (AI). OWL is an extended version of the RDF Schema with enhanced expressiveness.

OWL-S, OWL for services, is an ontology that describes the semantic web services that are expressed in OWL. The semantic description is a machine interpretable description of a web service that facilitates the automatic discovery, composition, invocation ,and interoperation of services. Semantics turn the programming interface description into a content-oriented representation of the service. Since the semantic aspects are not a fundamental part of the web service description, they must be added or linked to other semantic descriptions (cf. Appendix A.7).

The ontology-based description provides a common vocabulary for describing a domain. The standard vocabulary of a domain provides the means for interoperability between services. An ontology adds semantics to web services through detailed information about the service. Generally, OWL-S describes a service in three types of disciplines:

What does the service provide for clients? The answer to this question is filed under *profile*. How does the service work in terms of a process model? The answer to this question can be found in the *service model*. How does one service interact with another? The answer to this question can be found under *grounding*. Grounding supplies the details of how to embody those interactions in real messages to and from the service. Each service profile may be thought of as a summary of salient aspects of the process model plus additional information suitable for the purposes of advertising and selection.

An OWL-S profile is used to advertise a service as a function of three types of information: 1) service provider information, such as web URL, physical address name, and E-mail address. 2) service functional description, including inputs/outputs, preconditions, and effects. 3) service functional properties such as category and geographical availability. The process model informs clients how to request the service, and describes step by step the processes carried out by the service.

This description can be used when seeking a service in three different ways: 1) when performing a strength analysis of whether the service meets the request, 2) when composing service descriptions from multiple services to perform a specific task, 3) when monitoring the execution of the service.

2.4.3 UDDI: Universal Description Discovery and Integration

UDDI is offered as a public XML-based registry of web services available on the Internet [CHvRR04]. UDDI allows businesses to publish their services by defining APIs and data structures and discovery of services by querying the published descriptions. The UDDI APIs are specified in WSDL, and the registry itself can be accessed as a web service. The UDDI business registry is a logically centralized, physically distributed service with multiple root nodes that replicate data among one another on a regular basis. Once a business registers for the first time with the business registry service, the data is automatically shared with other UDDI root nodes and becomes widely available to anyone who wants to learn what web services are used by a given

business [CHvRR04].

The UDDI registry is divided into four components: 1) *businessEntity* includes the name and description of the business and is a catalogue of organizations and their information such as address and contact information; 2) *businessServices* provides a classification of the service or business, businessEntity, based on standard taxonomies; 3) *bindingTemplate* describes how to access a web service, including technical information on the service bindings and its address; 4) *tModel* consists of a general technical specification that can be written in any language, technical model of service property, interface, protocol, and any classification schema. This information helps to program client applications that can bind to the corresponding service.

Although UDDI is a flexible registry for syntactic interoperability of web services, it fails to publish and enable search on the dynamic properties of services. Enhanced UDDI is an enhanced registry that alleviates these limitations and obtains more accurate search results where two identical XML descriptions may have different meanings. Capability-based (semantic) descriptions articulate the functionalities of the web service in terms of inputs, preconditions and outputs. Since the registry is a major storage space for web service information, it is the best resource for maintaining the service's semantic description. An extension of the UDDI registry that can be used to add a capability search functionality is addressed via developments in OWL-S and UDDI [SPS04]. Such an architecture brings these technologies towards web service selection by embedding an OWL-S profile description to UDDI. For processing OWL-S profile information, the UDDI registry is augmented with an OWL-S matchmaking component. To search for web services based on their capability a *capability port* is added to the UDDI registry. Upon receiving an advertisement through the *publish port*, the *OWL-S/UDDI matchmaker* processes the UDDI advertisement. If the advertisement includes the OWL-S profile description, it then forwards the advertisement to the matchmaking component. The matchmaker classifies the advertisement based on the semantic information it contains. The degree of the match between services depends on the match between the concepts that are represented by them. The matching between concepts is not syntactic, instead it is based on the service concepts in their OWL ontologies [SPS04]. Moreover, for the embedding of OWL-S profile information into a UDDI registry, an OWL-S/UDDI mapping mechanism is implemented [SPS04]. The mechanism is a one-to-one mapping of OWL-S profile elements to corresponding UDDI elements.

2.4.4 SOAP: Simple Object Access Protocol

Interactions among web services, SOAP is designed for message negotiation and transmission on top of HTTP in an XML format [SOA07]. This communication protocol is one-way and asynchronous. It can be evolved with an underlying protocol or middleware. SOAP exchanges information in the form of messages in an envelope with two main parts: An optional header and a mandatory body, both of which can have multiple sub-parts. SOAP messages can be transported from sender to receiver while they are processed by a number of intermediaries. The header includes all information that can be accessed and processed by intermediaries, and the body conveys a message for the receiver. The web service intermediaries can access the header information according to some indication provided in a SOAP message. For example, the *none* indication means that the block can be read but not processed by nodes. The *next* indication allows the precedence of the block by a node. The header carries the requester information. Another type of interaction that can be made through SOAP is RPC-style in which the requester makes a method call, for example a PhotoOrder-Request method call. In this case, the procedure name and input parameters are specified in the body. It should be noted that data structures are encoded into *SOAP encoding*, an XML schema representation of data types such as integer, string, and array. The downside is that binary data can not be transformed through SOAP, but can be transported as attachments, URIs, or through other protocols (e.g., DIME).

2.4.5 Web Service Discovery

To use available services, consumers need the ability to search and access appropriate services. Automatic service discovery is a promising process to make the searching and accessing process dynamic, without requiring user intervention. For instance, a user on a mobile device should not need to memorize the IP address of a desired service, nor should they be required to download device drivers manually. At its most basic, for automatic service discovery in ad hoc networks, services expose both their capabilities and access information, then consumers locate a service through its service type, and finally select the appropriate service among the many discovered services.

Some of the relevant Service Discovery Protocols (SDP) are Service Location Protocol (SLP), Universal Plug and Play (UPnP), Multicast Domain Name System (mDNS-SD), Universal Discovery and Description (UDDI), and Web Service Dynamic

Discovery (WS-Discovery). We give a basic overview for each of these below.

The Service Location Protocol (SLP) is used for local area networks. SLP was developed by the IETF SvrLoc working group, and consists of three main components: a User Agent (UA), a Service Agent (SA), and an optional Directory Agent (DA) [GP99]. The UA initiates service discovery on behalf of the user. A query to retrieve service information is then sent to the SA through multicasting or to a DA via unicast. SAs are associated with services or devices that process work on behalf of a service to advertise itself. A DA is a centralized information repository that collects service advertisements and makes itself known by multicasting a message about its availability. When a UA issues a request to a DA via unicast on behalf of the user application, the DA checks its database for matching entries for a given query and returns a URI for each service found. It also allows the UA to send its request to the SA via multicast so that all of the registered SAs can know about the request cast. SLP is renowned for its scalability and flexibility as it can be used for networks of different size without any pre-configuration (with or without a DA).

UPnP has been promoted by Microsoft for connectivity between PCs, intelligent appliances, and wireless devices. It includes a set of protocols that were originally designed for small networks where a peer-to-peer mechanism for automated device re-configuration or service control is possible [UPN12]. UPnP consists of services, devices, and control points. Periodically, devices announce their presence to the control point by providing descriptions of themselves and the services they offer. Clients who need to discover a service run such a control point. The control point then waits for advertisements from devices or searches for devices via a multicast message specifying the desired service. A client can retrieve an XML description of the device that includes the attribute values of the associated service, or if a device has a URI it can retrieve a page. Therefore a user can control or check the status of the device.

mDNS/DNS-SD was developed by Apple as an extension to Domain Name Services to operate over a multicast protocol [CK11]. Having multicast capabilities, it is no longer necessary to maintain a single unicast DNS server on a local network. Network services provided on local devices (e.g., printer, document sharing, music sharing) can advertise and be located through DNS service discovery. When a consumer needs to use a previously discovered service, it queries for the SRV (port number) and TXT (additional information about service) of the service.

UDDI is an XML-based registry sponsored by OASIS, providing facilities to allow

businesses to publish, discover, and interact over the Internet. It is a platform-independent way to locate web service applications. It consists of three important components: White Pages, Yellow Pages, and Green Pages; that is, the name and description of the business, the standard service or business category, and service technical information, respectively. The information is exchanged between a service requester and provider through Simple Object Access Protocol (SOAP) [SOA07]. The required service information (functionality of web service) is described by Web Services Description Language (WSDL) that is a machine-readable, XML-based format [CMRW07]. UDDI provides the capability to search for specific service types characterized by attributes in a limited way.

2.4.6 A Review of Architectures and Techniques for Web Service Discovery

To enable dynamic web service discovery, several architectures have been studied. The architectures fall into two main classifications: centralized and distributed [DA10]. Centralized architecture enables saving the web service descriptions in a central repository that can be accessed through an API (e.g, UDDI) or at web portals (e.g. XMethods). Here, the registry is not only a repository of web services, but it also manages both query and discovery processes (cf. Section 2.4.3), and the service descriptions (e.g. WSDL) are accessible through web portals. The central approaches to serve service discovery suffer from a number of drawbacks, such as exposing a single point of failure, scalability problems, and performance issues when information is not available at one site. To alleviate these issues, the distributed approaches enable accessing the service description at multiple sites by virtue of different designs. The web service description can be accessed through a web crawler; a software agent to further process the query and find the appropriate services accordingly. It could also be accessible through a shared node in which the web service information is distributed among peers to increase fault tolerance.

Depending on the method of web service discovery, a web service description can be specified on functional and non-functional attributes. The service discovery methods based on the functional attributes are mainly based on keyword or category matching. Interface matching is another type of functional service discovery in which the similarity of the services' interfaces, (e.g., input and output parameters) are identified in order to find the most relevant services. To bring a level of semantics to

functional service discovery, ontology methods are introduced. Herein the functional terms of web services and their relationships in different domains are specified and utilized to explore the desired web services. Other information can also be provided in the form of an ontology for functional web service discovery. Functional service discovery methods are 1) UDDI-based, 2) designed to rely on semantic matching, and 3) focused on user preferences to fulfill expectations.

So far we have depicted the taxonomy of web service discovery methods by reviewing the literature. We now introduce some of the discovery mechanisms that are based on functional attributes. Existing methods mainly use standard informational retrieval techniques [MRS08] to find relevant service descriptions [SWY75, DHM+04, LCS97]. Broadly, these descriptions are represented as vectors located in a vector space model (VSM) [CFV07]. The query is considered as a vector and the nearest neighbours in a vector space are measured through different mechanisms such as cosine measure [CZC08]. Crasso et al. [CZC08] proposed a VSM-based method, WSQBE, to look for the desired services efficiently. In this method, users are provided with partial information about the service such as a set of words and functional descriptions of the expected interface, referred to as a skeleton. WSQBE then analyzes this skeleton to find similar services in a relevant category. The documents are classified in a set of pre-specified categories such as financial and education, to reduce the search space. The services are categorized based on their textual contents and their method signature using the TF-IDF heuristic.[1] Here, TF-IDF is a word weighting heuristic in web service descriptions to convert the list of the words returned from the text-mining process to the vector space. One of the distinctions of this work over other similar methods is the provisioning of a text-mining process for analyzing WSDLs before mapping them to vectors. This process alleviates the existing issues of service discovery based on WSDL, such as grammatical and spelling mistakes, snippets of abbreviated text, and different coding and naming conventions that lead to similar descriptions treated as non-similar or vice versa. For instance, removing all words that have low levels of usage or usefulness within a given context.

Wang and Stroulia [WS03] combined the IR method, similar to Crasso et al., [CZC08]with a structural matching of service operations. The structure matching is computed over multiple steps. First, the data types that are specified in WSDL are compared in a relaxed matching mechanism to avoid any strict similarity matching. Then, a message and operation structure matching process scores the similarity of source and

[1] http://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

target web services. For instance, they specify a semi-compatible match where the data types that can be adapted to each other (i.e., int and float). The results from this research show the effectiveness of this method over the methods utilizing either the IR mechanism or structure matching only. Notably they evaluate the approach using the IR evaluation metrics (i.e., precision and recall). They also improve their work with semantic matching employing WordNet [SW05]. WordNet is a hierachical semantic relation between words that provides not only lexical matching but also hyponym and homonym matching.[2]

Verma et al. [VSS+05] proposed a UDDI-based web service discovery employing ontologies, DAML+OIL, to add semantic to the WSDL service description (i.e., pre/-post condition). DAML+OIL ontologies provide an interface to UDDI that allows querying based on ontological concepts [CvHH+01] for sharing definitions. To add semantic information to UDDI they add tModels (cf. Section 2.4.3) in the registry that are linked to ontologies. Thus, each operation in WSDL is a key that maps to values that are concepts in ontologies. Similar to this work, Paolucci et al. [PKPS02] maps DAML-S [AJK+03], that is a semantic representation of properties and capabilities of web services rather than WSDL to the UDDI structure.

Balke and Wagner [BW03] proposed a preference-based service selection to coordinate a user's desires with a service discovery process. They describe the importance of web service personalization with a flight booking example, demonstrating how to rule out inadequate services. They gather the user preferences from the user profile or domain knowledge and expand the query with this gathered information. For instance, if a user is looking for a flight, the return result can be optimized with his/her profile information such as Delta, and non-stop.

In the next chapter, we show how our proposed architecture handles non-functional attributes to improve the quality of web service discovery.

2.5 Summary

Multiple technologies have been invented to enable service outsourcing over the web. For instance, B2B integration had been implemented to facilitate integration of heterogenous software systems. Although, the traditional middleware mechanism became less valuable due to trust-related issues, lack of appropriate communication protocol,

[2] `http://wordnet.princeton.edu`

and support from software vendor. Web services contribute to resolve these limitations, and therefore have been adopted by businesses. Web service technologies and standards increase interoperability and service to consumers through the Internet. For instance, to expose service functionalities, service descriptions and interface languages have been developed. These are discoverable by consumers through UDDI mechanisms. SOAP is designed to accommodate interaction among web services. Finally, these services can be integrated through SOA paradigms.

Primarily, efforts have thus far been focused on publishing, searching, and finding web services that fulfill the requested functionality. An emerging line of research demands that we enhance existing mechanisms by considering non-functional attributes that affect service quality.

Chapter 3

Research Approach

3.1 Research Methodology

An aging population is causing significant challenges for the healthcare system. For example, 12.4% of the U.S. population is currently over 65 years of age, a figure expected to rise to 20% by 2030 [Jas11]. Therefore, the use of distance medical expertise becomes valuable by making available the ability to remotely monitor patients (i.e., patient-centric) through connected sensors, hubs, middleware, and a network of medical services linked together as opposed to accommodating patients in hospitals (clinic-centric). In fact, facilitating the patient-centric approach could result in cost reductions and provide better quality of life for patients [Jas11].

To implement a co-operative medical service to support integration and communication inter- or intra-organizationally, service discovery mechanisms have been employed. All participating members in the healthcare system should be able to automatically discover each other and access services and information while preserving confidentiality. Typical examples of such services are Tele-heath and E-health to deliver medical services, expertise, and information over a distance and through the Internet respectively.

Before the invention of web service mechanisms, conventional software systems were fairly rigid and interoperability was almost impossible. In 2000, I worked as a software engineer implementing a system to handle inter-communication transactions at the Asia Medical Center and Hospital in Tehran, Iran. Each application was designed and implemented for a specific department in the hospital independently(e.g., lab, medical image center); reusability and interoperability between these applications

was complicated when at all possible. Lacking web service technology, we managed to enable intercommunication by implementing some tightly-coupled applications. However, a minor change in one application could potentially have a major impact on the rest of the system, and gradually decreases system effectiveness when accommodating the new requirements.

In 2008, I worked as an intern student at IBM on the Panorama project—a large-scale web project implementing electronic public health solutions for the federal government (i.e, E-health). The objective of the project was to create an inventory of the Canadian public health system; this was proposed after the SARS outbreak of 2003, when it became clear that Canada's ability to contain an outbreak was hindered by poor communication between provincial and territorial public health systems. It was clear to me that it was the lack of service-based mechanisms that made this project necessary in the first place.

These experiences motivated me to delve into service search and discovery mechanisms as I found them very beneficial in encouraging reusability, improving interoperability, and reducing resources spent on coding. My goal was to eventually allow service oriented architectures with web service implementations to support interoperable service discovery in all sorts of environments, including the health care industry.

3.2 Literature Review

Universal Description, Discovery and Integration (UDDI) is the primary registry for service search and discovery, and has been promoted by companies such as IBM, Microsoft, and SAP as a way to register and expose service functionality to consumers. Unfortunately, the data in a UDDI database is not curated and there is no mechanism to evaluate the validity of the information. Studies indicate that between 63% and 67% of the registries are not valid due to incomplete, incorrect, duplicated, or outdated web service descriptions [KCO05] [FK05]. As a result, most companies discontinued their public UDDI repositories in 2006. UDDI remains in use on private networks where information about services available on the private network can be curated.

We investigate how to improve the accuracy of web service discovery in a large heterogeneous network. To address this objective, the first research question is "Is it possible to search for web services only by keyword or by lexicographically matching service descriptions?" And if it is, how could we search based on both "what they do" and "how they do it?"

First, we studied approaches that improve the syntactical matching of service operations (i.e., functionalities) using both IR methods and structural matching [WS03] [SW05]. Then we examined service functionality described with a semantic representation, and accordingly used a semantic matching mechanism [VSS+05] [PKPS02]. Although these mechanisms improve the accuracy of service discovery, they still suffer from a major drawback: all these approaches search functionality, ignoring non-functional aspects of services. To remotely monitor a patient's vital signs and provide

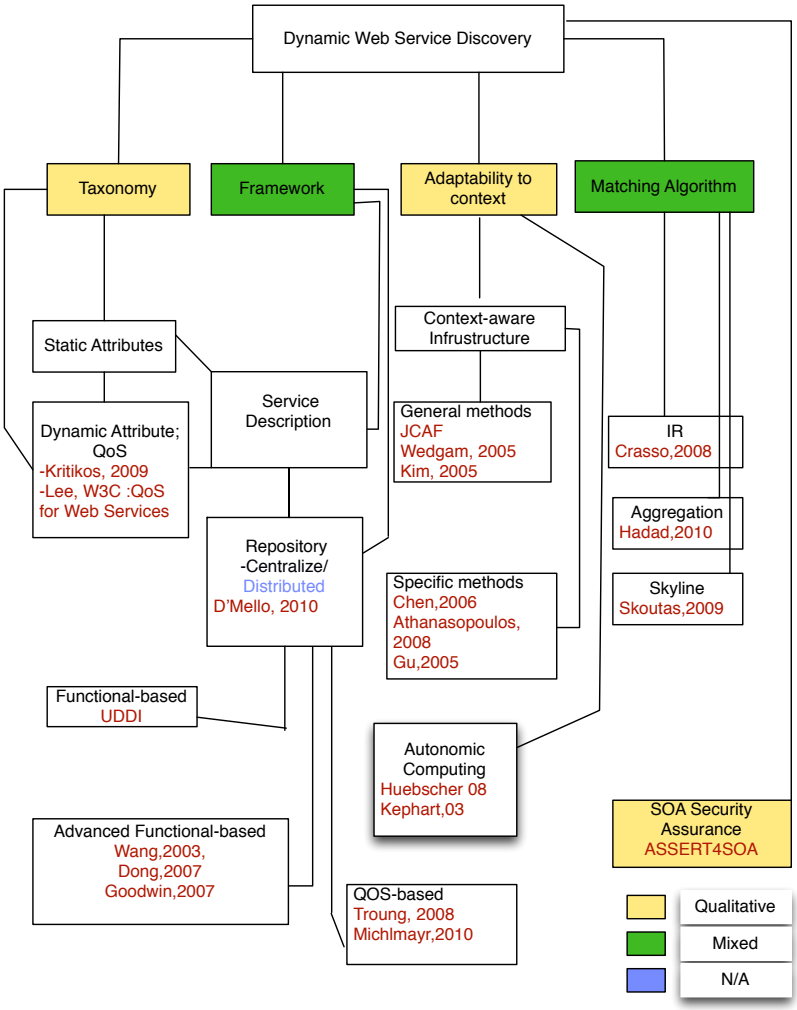


Figure 3.1: Literature map

feedback to the patient and healthcare professionals, we need a reliable, secure, and high performance service [Jas11] [WBvB+09]. These qualities—referred to as Quality of Service (QoS) attributes, or more generally dynamic attributes—enable us to

describe and therefore search for services in a quantitative manner.

This leads to the next research question: "Does the suitability of each service need to be determined with respect to dynamic attributes? Can we describe web services in a more quantitative manner?"

Accordingly, we reviewed the existing approaches that attempt to integrate QoS attributes in service discovery [KP09] [ZBD+03] [MRLD10] [TDB+08] [TRPA06]. In order to perform qualitative research [Cre09], we also examined VRESCO infrastructure [MRLD10] [TDB+08] to build understanding from the requirements for designing and implementing QoS-aware service discovery. All these approaches attempt to integrate QoS attributes to the advantage of both users and providers, allowing a QoS-aware web service selection to address the user's QoS requirements. We performed a qualitative study in order to answer the relevant research questions and understand the strengths and weaknesses in these approaches. We chose to focus on five broad areas: QoS- *attribute taxonomy*, *framework*, *adaptability to context*, *matching algorithm*, and *security*. Based on our knowledge, these are the building blocks for implementing high quality web services (cf. Figure 3.1).

To develop a taxonomy of QoS attributes for the purpose of web service discovery we conducted qualitative research [Cre09], studing several related works [TRPA06] [KP09] [ZBD+03] [LJL+03] (cf. Figure 3.1). Our objective is to identify ubiquitous categories of QoS attributes. We explored the question "What type of dynamic attributes could be monitored and how are they distinguished at runtime?" We are aiming for an abstract taxonomy of attributes that contribute to the discovery process. Our taxonomy should be generic and reusable in most domains and only support the important metrics that can be efficiently evaluated during the service discovery process. This factor is our contribution to existing QoS taxonomy in the abovementioned mechanisms (cf. Figures 1.1 and 1.4) .

We developed a framework for dynamic web service discovery that integrates with existing static discovery. This framework measures or computes the QoS attributes identified in the taxonomy and monitors them at runtime. We first conducted a qualitative study to explore the pros and cons of implementing such a framework. For example, we studied VRESCO, an advanced runtime environment for supporting QoS attributes [MRLD10] [TDB+08](cf. Figure 3.1). While QoS-based approaches do support the QoS attributes, they did not integrate with the existing static discovery mechanisms, one of our main objectives. Furthermore, they measure the QoS attributes in advance and use them during the web service discovery process, the values

of these attributes being potentially stale at discovery time. To test the effectiveness of our framework, we did a quantitative study (cf. Figure 3.1) by comparing the number of results returned from our implemented framework with the conventional service discovery repositories such as UDDI or Seekda. The details of our mixed methods research will be explained in Chapter 4.

After generating a taxonomy of QoS attributes and building an appropriate framework that integrates with conventional service search and discovery, we examined the following research questions: "When and how should these attributes be evaluated in the web service discovery process? How can we specify the evaluation techniques for these attributes, in different contexts, at runtime?"

Context-aware service discovery approaches use contextual information to achieve better system performance and user experience [HJSY08] [OVS+06] [HIMB05] [TD09] [PTC07]. In our qualitative research, in most of the literature, contexts such as location and time are taken into account to refine service discovery results. We concentrated on these contexts as a fundamental aspect to filter out irrelevant services. Researchers have implemented context-aware systems by identifying a set of predefined QoS attributes and contexts, but this methods tie the system to a specific set of attributes. In a time-critical service such as telemedicine, response time, medical facilities, and location would be important attributes and contexts in an emergency situation. However, under normal circumstances different attributes such as the diagnosing practitioner would be more important for patients. To distinguish important attributes at runtime rather than examine a static set, we leveraged the knowledge and experience in a self-adaptive system in the RIGI group; we chose to employ an autonomic control loop (i.e., MAPE-K) to develop a self-adaptive web service discovery mechanism. Such an implementation is able to select a specific set of QoS attributes for a given context at runtime. Based on our research, autonomic control loop is a viable approach to implementing self-adaptive service discovery. This gives us a framework that can be easily extended to support a wide variety of context in many different domains. The details of our qualitative study is presented in Chapter 5.

Most of the QoS-based mechanisms in web service discovery employ aggregation [EHMR10] or IR mechanisms [CZC08] to score and rank services(cf. Figure 3.1). The IR mechanism are based on human judgment, and generally the accuracy of the results is subjective due to unreliability in human judgment [KKR09] [KKR08]. Aggregation also suffers from several drawbacks: first the reduction of several values to one single score reduces the accuracy of the results, but also modeling a weight vector

containing threshold values for QoS attributes is not a trivial task. To avoid these issues, Skyline is often proposed [SSS+09]; however we could not find any quantitative evaluation that determines the accuracy of this approach in the service discovery area. In addition, the algorithms that have been implemented in service selection ignore the challenges for computing the QoS attributes at runtime. We wondered "Why is evaluating web services based on dynamic attributes challenging? Are there any aspects that need to be taken into account for appropriate service selection?" In fact, the values of QoS attributes fluctuate over time, potentially dramatically, and may sometimes not be available at all. Our objective is to develop algorithms for computing attribute values in the presence of fluctuations and missing samples to improve the accuracy of the service discovery result. This led us to ask the question "what is the appropriate matching algorithm for quality web service discovery? Since these attributes potentially fluctuate and are context-sensitive, how we can avoid misinterpretation due to temporary fluctuations of these attributes?" Our contribution is a set of advanced matching algorithms—Dynamis aggregation and Dynamis skyline—that support temporary fluctuations and missing values. To determine the accuracy of service selection results, we have performed a quantitative evaluation to compare the following algorithms; aggregation, skyline, Dynamis aggregation, and Dynamis skyline. The results show a major improvement in accuracy using the Dynamis approaches. Our proposed algorithms are studied in Chapter 6.

Finally, security is an important attribute; for example a patient's medial records must be protected and only disclosed to the intended healthcare provider. Research questions are "With enabling on-the-fly service selection, and exposing the service functionalities over the Internet, how can we avoid the security and reliability risk when compared to off-the-shelf software systems? How can we consolidate security in web service discovery, and how can we alleviate the security risk at runtime, in an SOA environment?" We collaborated on an infrastructure project—ASSERT4SOA—with the University of Milan [ASO10]. This infrastructure provides all the standard mechanisms for service security evaluation and certification. We then integrated our framework with ASSER4SOA to re-certify security attributes at discovery time. Essentially, our framework re-evaluates the security requirements at runtime and a new certification is assigned to each service that is being used for service scoring during service selection process. Our contributions to ASSERT4SOA are explained in Chapter 8 and is represented in Figure 3.2.

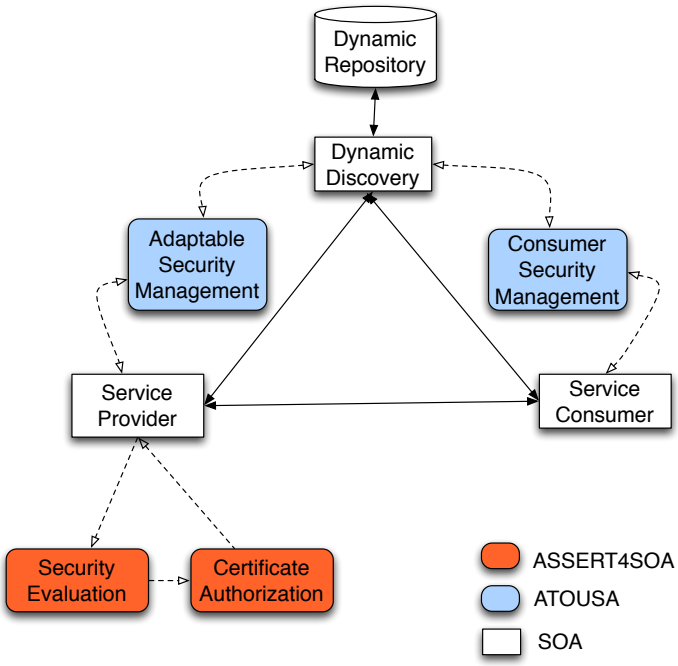


Figure 3.2: Research Plan

3.3 Summary

The invention of web services enables reusability and interoperability in software systems. Working in industry as a software engineer implementing systems to support inter- and intra-organizational transactions, I realized that the service-based mechanisms solve many interoperability problems and reduce costs significantly but also introduce problems such as service discovery. These experiences encouraged me to focus on service search and discovery mechanisms for my PhD research. In my investigations I studied related work and reviewed existing approaches in the literature to identify research gaps and formulate research questions. My research involved qualitative, quantitative, and mixed methods. I designed and implemented a high-quality service discovery framework. My evaluation strategy involved several case studies. In the end I achieved consistently better accuracy in service discovery than existing mechanisms.

Chapter 4

A Dynamic Framework for Quality Web Service Discovery

A service-oriented architecture (SOA) is an architecture for a collection of services that communicate with each other (cf. Chapter 1.4). SOAs are built around loosely integrated service endpoints, enabling interoperability among heterogeneous systems. Because the number of web services has increased, the concept of a discovery mechanism utilizing SOA infrastructure capabilities was introduced so that applications could identify the services they need more effectively.

One of the key phases in web service discovery is service selection—the process of finding a service that is pertinent to a user's request based on both static and dynamic attributes [LW07]. Service providers register their services in a registry, which can then be accessed by the public. A broker helps requesters and service providers find each other. The requesters ask the broker about the services they require. When the broker returns with results, the requesters use the findings to bind themselves to a particular service. Web service discovery standards such as UDDI are based on the syntactic keyword and category mechanisms (cf. Chapter 2.4.3). Such techniques, albeit crucial to finding the services regarding "what the services serve" (functional), do not search based on "how the services serve" (non-functional) (cf. Chapter 2.1). Because of this shortcoming, Microsoft, IBM, and SAP shut down their own UDDI public registry in 2006. IBM developed its own registry, WebSphere Service Registry and Repository (WSRR), and offers a UDDI module to synchronize the contents of WSRR with the UDDI v3 registry. Notably, UDDI has several shortcomings such as lack of interoperability and semantic support; some reference materials for further

reading are [AMH10] [Min08].

One of the challenging issues in SOA is ensuring quality of service (QoS) by enabling service selection in a dynamic environment ("dynamic web service discovery") [SZW+07] [TN08]. Considerable research has been done to support dynamic attributes in SOA. Researchers have suggested the enhancement of the UDDI standard to register dynamic attributes [Don07]. Another suggested solution is using an expanded broker to store and manipulate dynamic attributes [GRQ07]. A hybrid method combines both approaches.

To improve the quality of dynamic web service discovery, we propose an enhanced model to support the dynamic attributes based on SOA principles. We outline an architecture framework called static discovery dynamic selection (SDDS) [PM09] to evaluate the dynamic attributes concerning both context and domain information during discovery. The architecture of SDDS defines individual components that collectively satisfy a flexible and intelligent service selection for realizing SOA. In this chapter we propose and outline:

- A robust resource management approach to collaborate with partners concerning static and dynamic attributes.
- An advanced SOA model to register, collect, store, and measure the dynamic attributes.

One of the main challenges of enabling dynamic service discovery is to develop techniques and models to handle the novel aspects of the web services paradigm (i.e., updated static and dynamic attributes). This challenge leads to a variety of research questions:

- What is the best way to model web services using static and dynamic attributes?
- How does one query them to support dynamic attributes?
- How can data management be incorporated into current web service standards?

To address these questions, more attention needs to be paid to the methods used to identify qualified services. In particular, it is worthwhile to understand the key elements in the design process and to identify the activities that support the required properties. For example, orchestration between new components and existing service

discovery components is critical for satisfying properties such as data validity and liveness in shared databases.

The rest of this chapter is organized as follows. Sections 4.1 introduces our framework. The detailed analysis and validation of SDDS using finite state automata is discussed in Section 4.2. Finally, Section 4.3 draws some conclusions.

4.1 Static Discovery Dynamic Selection (SDDS) Conceptual Architecture

The overall system architecture of SDDS includes two major processes, as depicted in Fig. 4.1. Static discovery (SD) is a standard service search and discovery mechanism, compliant with UDDI. This process suffers from several limitations in finding qualified services. To improve the SD process, dynamic selection (DS) is introduced to select the convenient services among the discovered ones. Here, the required service information is provided to DS by standard service discovery sensors. DS contains two main components: the user dynamic manager and service dynamic manager. Both facilitate static and dynamic data collection, analysis, planning, and storage. These components collaborate to select the appropriate services for a given request.

4.1.1 SDDS: The Proposed Architecture Model

Our SDDS architecture and experimental set-up for enhanced service discovery is based on SOA [PM09] [PCM10] and depicted in Fig. 4.2. It supports static and dynamic attributes using the following components:

- **Service Consumer:** A consumer of the services provided by others, who makes a request for a specific service.
- **Service Provider:** A provider of services to others, who explicitly registers a service with a web service registry.
- **Proxy:** A web service broker that deals with passing and parsing messages between components. Proxies are used for publishing and querying relevant dynamic attributes. The proxy forwards standard service descriptions to the standard service discovery repository (e.g., UDDI) and returns the unique identifier of the published service. Additional dynamic attributes are stored with

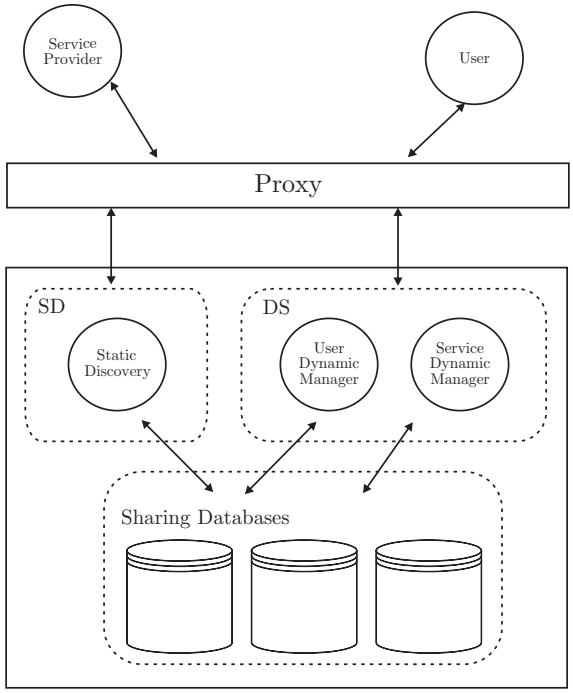


Figure 4.1: SDDS conceptual architecture

the dynamic repository by the service dynamic manger. The proxy also accepts both standard and dynamic queries.

- **Static Repository:** A typical repository in which to store the standard service descriptions, which are static values.
- **Dynamic Repository:** A database in which to store high-quality services that are checked for dynamic attributes. To identify the qualified services, several quality aspects are taken into account.
- **Knowledge-base Repository:** A database in which to store knowledge for domain processing, in addition to information on how the processing is actually performed.
- **Web Service Profile:** A history of the selected web services.
- **Static Service Discovery:** The standard service discovery and selection mechanism, based on static attributes.

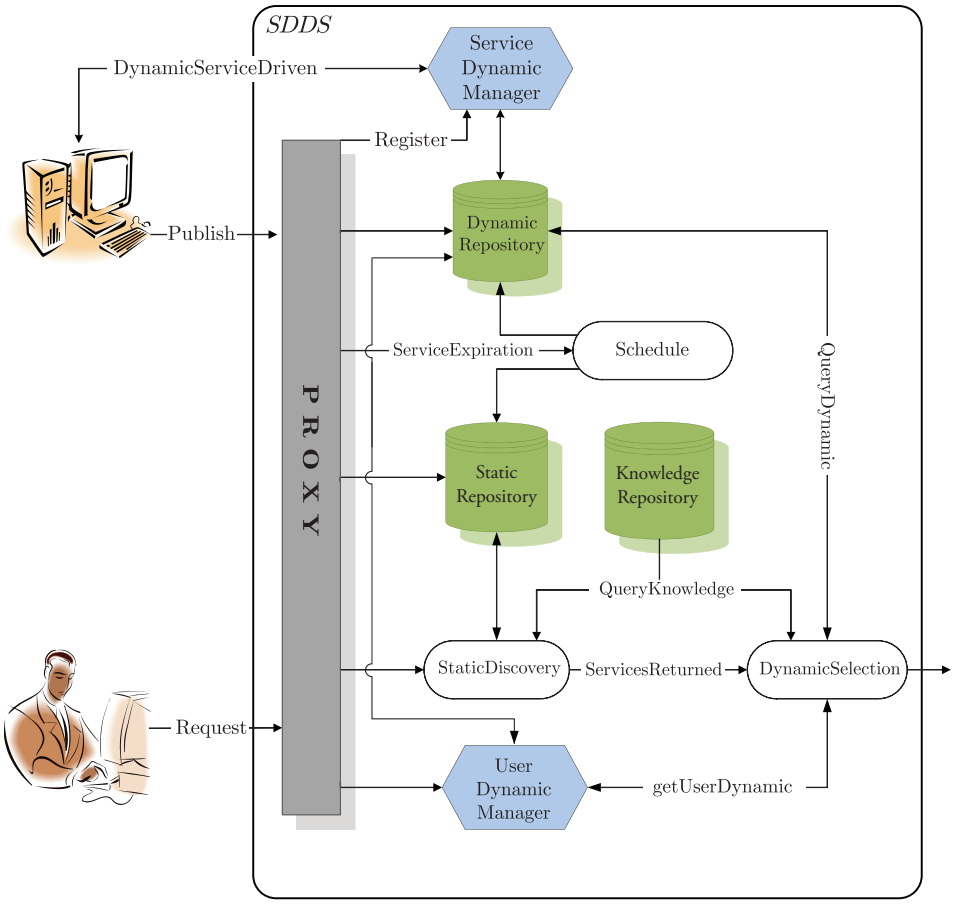


Figure 4.2: SDDS architecture

- **Service Dynamic Manager:** A manager to address the non-functional attributes of selected services in order to best identify service properties. It checks the quality of the services in the standard static repository and stores the qualified services in the dynamic repository. To determine the high-quality services, aspects such as user context, service context, domain knowledge, and user feedback are taken into account.

- **User Dynamic Manager:** A manager to deal with the dynamic values on the user side. It extracts the dynamic attributes from queries and stores and retrieves users' and web services' profiles on demand.

Two kinds of dynamic attributes can be specified in SDDS: service and user dynamic attributes. We assume that each service is equipped with a mechanism to capture its dynamic values from relevant sources and provide the gathered informa-

tion to the SDDS. Our architecture for high-quality web service discovery comprises two phases: Static discovery (SD) and dynamic selection (DS). First, the SD process resembles an index method that enables service discovery within a collection of services classified by their static attributes. Second, the DS process is initiated for service retrieval based on dynamic attributes.

The processes involved in the SDDS architecture include:

Static Discovery (SD): Handles standard service publishing and querying. It discovers the appropriate services by employing a scalable structured index. It employs a classification technique to group similar services together based on service type and to maintain those services in a tree structure for efficient service discovery [CHvRR04]. The service type is identified based on the static attributes of the web. As a result, service discovery costs are reduced due to a more efficient search with an indexing structure. The candidates resulting from this step are forwarded to the next step for further processing and filtering.

Dynamic Selection (DS): Handles the evaluation of dynamic attributes of web services and filters out services that are not relevant to the user based on dynamic values gathered from different sources. It is initiated by Static Discovery to select the highest-quality service by evaluating the dynamic attributes of discovered services.

Domain Handling: For most domains, there are no quality criteria for web services. Determining the quality of service requires domain-specific knowledge that can be given by experts to represent the priorities and preferences of the specific domain. These quality criteria can be defined by a quality ontology, and stored in Ontology Knowledge Databases to enable the automated web service selection process. However, based on each application, application-specific metrics are specified and assigned a value based on the quality properties of services. Those values are stored and updated as default weight vectors in the Dynamic Repository. The default weight vectors contain threshold values for quality parameters used in the domains that are time-critical. Then, in the selection process, the services which have a value less than the threshold are eliminated. Hence, it creates a trust-based QoS-aware service discovery model by eliminating services that are inconsistent with the default weight vector in the same service type, and subsequently creates a web service discovery mechanism with implicit QoS filtering.

Personalization: SDDS provides personalization by allowing the User Dynamic Manager to keep track of selections made by users and their web service history. Thus, the next time a user requests a service of the same type without identifying

their preferences, DS is able to look at the history of a user's previous selections and web service history to make a selection based on that data.

Request management: The importance of a dynamic attribute for a specific service may differ for different users. For example, for one user service availability might be critical, while to another, a faster service response time might be more important. SDDS provides request management as a means to allow users to specify different preferences for dynamic attributes.

4.2 Design of SDDS

Our SDDS design is mainly component-based—that is, combining existing methods with new software to provide new functionality. Thus the "transaction support" related QoS attributes (cf. Section 1.6) are crucial to guarantee the integrity of data during the service discovery process [KP09]. We propose a constrained finite state automata formalism to model the interaction among SDDS components. We capture the I/O behavior of each component with an automaton. Input actions are used to model the methods that can be called, while the output actions are used to model the method calls between components. Using automata, we formalize the behavior of the SDDS architecture by modeling its critical constituents: service dynamic manager, user dynamic manager, and standard static service discovery. We use an automata-based language to capture the order in which the method of the components are called, and the order in which the components call external methods. A composite automaton is constructed from the product of the component automata. When the component automata are composed the result may contain invalid transitions, such as transitions that jeopardize data integrity in databases. To guarantee atomicity, durability, and consistency in databases (i.e., transaction support related QoS attributes), we attempt to identify and avoid such invalid transitions. This formalism for modeling the behavioral aspects of components can be used both during design and for validating the system [dAH01] [LZ09].

In the following sections, we introduce fundamental concepts and then discuss SDDS component automata and their composite automaton.

4.2.1 Defining Constrain Automata for SDDS

Primarily, automata are semantic models which can be used to validate the operational constraints of a system (e.g., being dead-lock free). The use of finite state automata was inspired in engineering disciplines to allow for robust mathematical analyses. In an automata model, the architecture is represented as a set of states and a set of transitions. Automata states denote the possible configurations of the system; automata transitions demonstrate the possible actions and their effects on those states. First, we introduce the terminology used in subsequent sections [Fok07] [SER$^+$10].

Definition 1

An SDDS automaton is a tuple: SDDS $= (Q, \Sigma, \Delta, q_0, M, V)$, where:

- Q represents a finite set of states;
- Σ denotes a set of actions;
- $\Delta : Q \times N \times g \to Q$, is the transition relation for SDDS where $N \subseteq \Sigma$;
- g is a set of constraints imposed on the transition relation;

  For simplicity, we use the transition $q \xrightarrow{N,g} p$ instead of the more complete $(q, N, g, p) \in \Delta$, where N is the action subset and g is the guard of the transition. For every transition $q \xrightarrow{N,g} p$, we assume that $N \neq \emptyset$; that is, automata transitions can only fire if an action occurs.
- q_0 denotes a set of initial states where the automaton starts;
- M presents a set of global memory cell for states;
- V for each $q \in Q$ is the value function $V_q : M \to \mathit{Value}$ defined when the automaton is in the state q. The set V_{q_0} includes the initial value functions of V_q, each of which gives the initial values for the memory cells of its corresponding state $q \in Q$.

Let $q_0 \rightsquigarrow^* q_n$ be a finite path as follows:

$$[V_{q_0}], \quad q_0 \xrightarrow{N_1,g_1} q_1 \xrightarrow{N_2,g_2} q_2 \ \cdots \ q_{j-1} \xrightarrow{N_j,g_j} q_j.$$

With every transition $q_{j-1} \xrightarrow{N_j,g_j} q_j$ in $q_0 \rightsquigarrow^* q_n$, we associate a descriptor $\delta : (In, M_j - 1) \mapsto (O, M_j)$ where In and O are input and output values, respectively.

4.2.2 Composing SDDS Automata

To compose two SDDS automata, we use the cross-product of two automata as follows.

Definition 2

Let $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{0_1}, M_1)$ and $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{0_2}, M_2)$ be two SDDS automata. Then the composition of these two automata is:

$$A_1 \bowtie A_2 \ = (Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \Delta, q_{0_1} \times q_{0_2}, M_1 \cup M_2)$$

where Δ is defined by the following rules:

$$\frac{q_1 \xrightarrow{N_1, g_1} p_1, q_2 \xrightarrow{N_2, g_2} p_2, N_1 \cap \Sigma_2 = N_2 \cap \Sigma_1}{\langle q_1, q_2 \rangle \xrightarrow{N_1 \cup N_2, g_1 \wedge g_2} \langle p_1, p_2 \rangle}$$

This rule implies a new transition whenever both automaton A_1 and A_2 want to change their states simultaneously. The new transition gets $N_1 \cup N_2$ as its actions if $g_1 \wedge g_2$ is true.

$$\frac{q_1 \xrightarrow{N, g} p_1, N1 \cap \Sigma_2 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle p_1, q_2 \rangle}$$

This rule implies a new transition whenever automaton A_1 wants to move from state q_1 to p_1 and automaton A_2 remains in state q_2 and is in idle mode.

$$\frac{q_2 \xrightarrow{N, g} p_2, N2 \cap \Sigma_1 = \emptyset}{\langle q_1, q_2 \rangle \xrightarrow{N, g} \langle q_1, p_2 \rangle}$$

Finally, this rule implies a new transition whenever automaton A_2 wants to move from state q_2 to p_2 and automaton A_1 remains in state q_1 and is in idle mode.

4.2.3 User Dynamic Manager

In this design, for a given request, the user dynamic manager collects the required user-side dynamic attributes either from the requester (explicitly) or from the requester profile (implicitly). The goal of this is to expand a query to increase the chance of selecting more relevant web services for a user's request. Fig. 4.3 depicts

the SDDS user dynamic manager as an automaton. $Q = \{x, y, z\}$ is the set of states and $\Sigma = \{\text{Add}, \text{Find}, \text{Expand}\}$ is the set of actions.

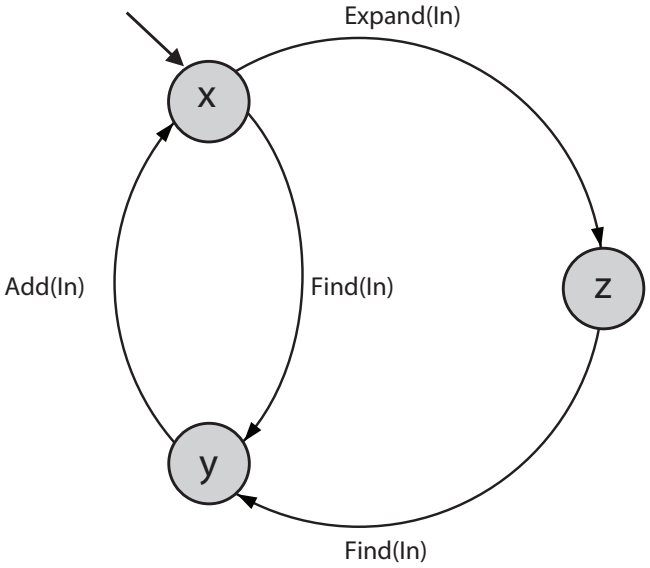


Figure 4.3: User dynamic manager automaton

$q_0 = x$ is the initial state. The memory cells for the state are:

$$M = \left\{ \begin{array}{c} (P : \{p_1, p_2, ..., p_\imath\}; \\ D : \{d_1, d_2, ..., d_\jmath\}; \\ S : \{s_1, s_2, ..., s_\ell\}) \end{array} \right\}$$

where P is a set of web service profiles, D is a set of services in the dynamic repository, S is a set of services in the static repository, and $D \subseteq S$. Each p_i and d_i has locked status. The value function V for initial state is defined as

$$V = \{P = \emptyset;\ D = \emptyset;\ S = \text{StaticRepositoryBrowser}()\}\,.$$

The transition relations Δ are as follows:

$$\left\{ \begin{array}{c} y \xrightarrow{\text{Add}(In)} x, \\ x \xrightarrow{\text{Find}(In),(\bigwedge_{k=1}^{\jmath} d_k.lock=false)} y, \\ x \xrightarrow{\text{Expand}(In)} z, \\ z \xrightarrow{\text{Find}(In),(\bigwedge_{k=1}^{\jmath} d_k.lock=false)} y \end{array} \right\}.$$

The descriptor $\delta : (In, M) \mapsto (O, V(M))$ is

$$\delta(In, M) \mapsto \begin{cases} (null, (P \cup \{f \mid \forall w \in D \wedge \mathrm{SAT}(In, w), f = \mathrm{REL}(In, w)\}; D; S)) \\ \qquad \text{if } \delta = \mathrm{Add}, \\ (\{w \mid w \in D \wedge \mathrm{SAT}(In, w)\}, (P; D; S)) \\ \qquad \text{if } \delta = \mathrm{Find}, \\ (\{In = \mathrm{EXPAND}(In)\}, (P; D; S)) \\ \qquad \text{if } \delta = \mathrm{Expand}. \end{cases}$$

We know that the $\mathrm{SAT}(In, w)$ function returns a true value if w, input web service In, is satisfied by the requester parameters. SAT() is defined as $\mathrm{SAT} : (In, w) \mapsto \{true, false\}$ with $w \in S$. The $\mathrm{REL}(In, w)$ function returns a record to insert into the web service profile for those web services that are relevant to input In. The $\mathrm{EXPAND}(In)$ function returns an input request which is expanded by the set of profile parameters P.

4.2.4 Static Discovery

Static discovery uses a standard discovery web service registry to store and discover services. Modeling of static discovery in automata is illustrated in Fig. 4.4, where $Q = \{1, 2, 3\}$ are the states, and the set of actions is $\Sigma = \{\mathrm{UFind}, \mathrm{UAdd}, \mathrm{Acknowledge}\}$.

$q_0 = 1$ and the memory cell set is

$$M = \left\{ \ (S : \{s_1, s_2, ..., s_\ell\}) \ \right\},$$

where S is a set of services in the static repository. The value function for initial state is defined as

$$V = \{S = \mathrm{StaticRepositoryBrowser}()\}.$$

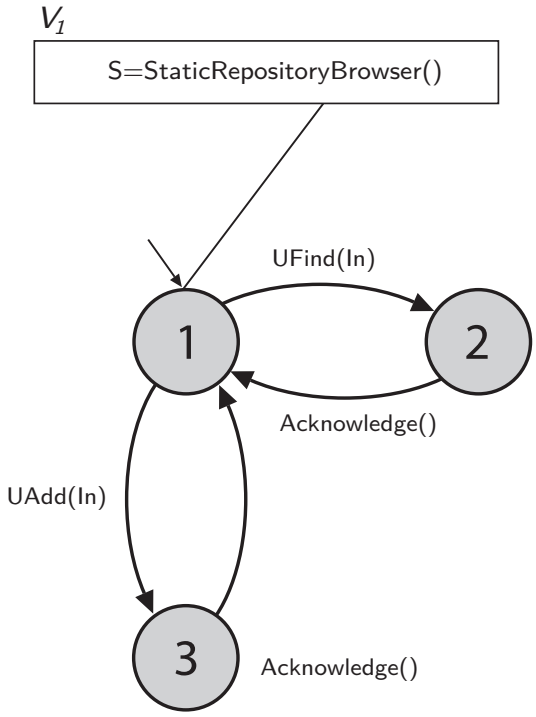


Figure 4.4: Static discovery automaton

The automaton transition relation (Δ) is

$$\left\{ \begin{array}{l} 1 \xrightarrow{\text{UFind}(In)} 2, \\ 1 \xrightarrow{\text{UAdd}(In)} 3, \\ 2 \xrightarrow{\text{Acknowledge}(\ )} 1, \\ 3 \xrightarrow{\text{Acknowledge}(\ )} 1 \end{array} \right\}.$$

The descriptor $\delta : (In, M) \mapsto (O, V(M))$ is

$$\delta(In, M) \mapsto \begin{cases} (\{w \mid w \in S \wedge \text{SAT}(w, In)\}, (S)) \\ \qquad\qquad \text{if } \delta = \text{UFind}, \\ (null, (S \cup \{\text{In}\})) \\ \qquad\qquad \text{if } \delta = \text{UAdd}, \\ (null, (S)) \\ \qquad\qquad \text{if } \delta = \text{Acknowledge}. \end{cases}$$

Here, the SAT(w, In) function returns a true value if w, input web service, is satisfied by the requester parameters In. $SAT(\ )$ is defined formally by: SAT : $(w, In) \mapsto \{true, false\}$ with $w \in S$.

4.2.5 Service Dynamic Manager

The service dynamic manager checks the quality of services in existing public registries based on their dynamic attributes. If the service passes the quality phase, then it is registered in the dynamic repository. The Dynamic repository is updated periodically by the service dynamics manager. Modeling of the service dynamic manager in automata is illustrated in Fig. 4.5, where $Q = \{a, b, c, d, e\}$ is the set of states, and the set of actions is $\Sigma = \{\text{Interval}, \text{Sensor}, \text{AcknowledgeSDM}, \text{UFind}, \text{Update}\}$.

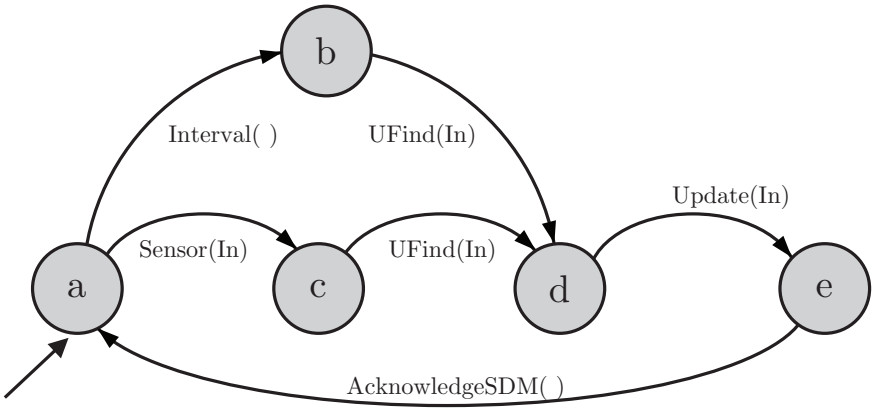


Figure 4.5: Service dynamic manager automaton

For the initial state $q_0 = a$, the memory cell set is

$$M = \left\{ \begin{array}{c} (D : \{d_1, d_2, ..., d_j\}; \\ S : \{s_1, s_2, ..., s_\ell\}) \end{array} \right\},$$

where D is the set of services in the dynamic repository, S is the set of services in the static repository and $D \subseteq S$. Each d_i has locked status. The value function for initial state is defined as:

$$V = \{D = \emptyset,\ S = \text{UDDIBrowser}()\}.$$

The automaton transition relation set (Δ) is:

$$\left\{ \begin{array}{c} a \xrightarrow{\text{Interval(In)}} b, \\ a \xrightarrow{\text{Sensor(In)}} c, \\ b \xrightarrow{\text{UFind(In)}} d, \\ d \xrightarrow{\text{Update(In)},(\bigwedge_{k=1}^{J} d_k.lock=true)} e, \\ e \xrightarrow{\text{AcknowledgeSDM()},(\bigwedge_{k=1}^{J} d_k.lock=false)} a, \\ c \xrightarrow{\text{UFind(In)}} d \end{array} \right\}.$$

The descriptor $\delta : (In, M) \mapsto (O, V(M))$ is

$$\delta(In, M) \mapsto \left\{ \begin{array}{l} (\{w \mid w \in S \wedge \text{SAT}(w, In)\}, (D; S)) \\ \qquad \text{if } \delta = \text{UFind}, \\ (null, (D; S)) \\ \qquad \text{if } \delta = \text{Interval}, \\ (null, (D; S)) \\ \qquad \text{if } \delta = \text{Sensor}, \\ (null, ((D \setminus \{w\}) \cup \{In\}; S) \\ \qquad \text{if } \delta = \text{Update} \wedge \{w \mid w \in S \wedge \text{SAT}(In, R)\}, \\ (null, (D \setminus \{w\}); S) \\ \qquad \text{if } \delta = \text{Update} \wedge \{w \mid w \in S \wedge !\text{SAT}(In, R)\}, \\ (null, (D; S)) \\ \qquad \text{if } \delta = \text{AcknowledgeSDM}. \end{array} \right.$$

Here, the SAT(In, R) function returns a true value if In, input web service, is satisfied by requester parameters R. SAT( ) is defined formally by SAT : $(In, R) \mapsto \{true, false\}$ with $w \in S$.

4.2.6 SDDS Goal Properties

We have shown a user dynamic manager with $A_1 = (Q_1, \Sigma_1, \Delta_1, q_{0_1}, M_1)$ in Fig. 4.3. Static discovery is shown with $A_2 = (Q_2, \Sigma_2, \Delta_2, q_{0_2}, M_2)$ in Fig. 4.4 and a service dynamic manager is shown with $A_3 = (Q_3, \Sigma_3, \Delta_3, q_{0_3}, M_3)$ in Fig. 4.5. SDDS uses

database sharing between these components. Composing A_1, A_2, and A_3 result in $B = (Q_B, \Sigma_B, \Delta_B, q_{0_B}, M_B, V_B)$:

$$B = A_1 \bowtie A_2 \bowtie A_3.$$

The automaton B—that is, the composition of A1, A2, and A3—should fulfill SDDS goals. However, sharing databases with multiple components may lead to data inconsistencies and invalid results. The typical example is that a record may be simultaneously read and written in our repositories. One resolution is to use mutual exclusion in the product algorithm so that databases cannot be accessed by more than one component at one time, ensuring correctness, consistency, and fairness. For instance, static discovery—Update(In)—running with service dynamic manager—UFind(In)—may lead to an invalid result.

$$d1x \xrightarrow{Update(In)\,UFind(In)} e1y \quad d1x, e1y \in Q_B$$

Furthermore, two different functions in a component cannot run on a shared memory set. Accordingly, this goal can be shown as follows.

$$\begin{aligned}
\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists a_1 \in \Sigma_1, a_1 \in N \Rightarrow \\
\forall a \in N, a \neq a_1, (a \notin \Sigma_1), \\
\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists a_1 \in \Sigma_2, a_1 \in N \Rightarrow \\
\forall a \in N, a \neq a_1, (a \notin \Sigma_2), \\
\text{For each } p_1 \xrightarrow{N,g} p_2 \in \Delta, \exists a_1 \in \Sigma_3, a_1 \in N \Rightarrow \\
\forall a \in N, a \neq a_1, (a \notin \Sigma_3).
\end{aligned}$$

An example of such a transition is shown in Fig. 4.6, where both UFind(In) and UAdd(In) are functions of UDDI discovery working on the static repository.

$$b1x \xrightarrow{UFind(In)\,UAdd(In),false} d1z \quad b1x, d1z \in Q_B$$

Enforcing mutual exclusion during the composition of components guarantees data validity; however, this solution does not scale well and is slow to accommodate service discovery when speed and flexibility have become competitive advantages. Several transitions that may cause deadlock/livelock in SDDS can be trimmed from the B

automaton; some of these transitions are shown by dotted lines in Fig. 4.6. Thus, the B automaton shows that the composed system does not satisfy the goal of efficiency due to constraints and invalid results. An example of one of these transitions is shown below:

$$d3x \xrightarrow{Update(In)Find(In),(In=d_k \wedge d_k.lock=true)\wedge(\bigwedge_{k=1}^{j} d_k.lock=false)} e2x$$

$$d3x, e2x \in Q_B$$

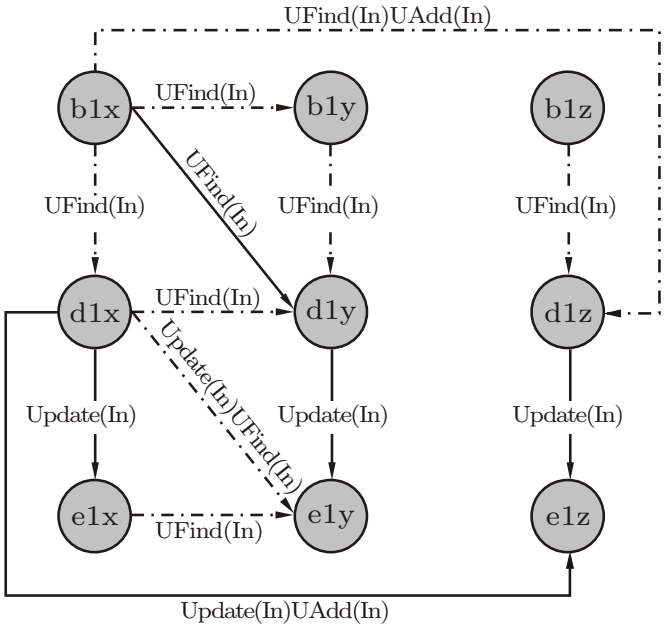


Figure 4.6: Partial view of SDDS automaton

Guard of this transition is false because $d_k.lock = true \wedge k = [1..j]$ and $(\bigwedge_{k=1}^{j} d_k.lock = false)$. However, this transition achieves high efficiency if $Find()$ action searches all dynamic repository elements except for the member which Sensor() is working on. In other words, we are changing the granularity of mutual exclusion to the level of work sets (records of sharing resources). Thus, to achieve high efficiency while preserving data validity, the system must be permitted to run when two actions want to access a common resource with different work sets.

If $\Gamma : 2^M \mapsto 2^M$ is a function that returns a subset of a component work set which

will be changed by another component, then we have the following,

$$(\Gamma(M_1) \cap \Gamma(M_3) = \emptyset).$$

We enrich our framework with a controller that guarantees not only validity but also satisfactory performance. The controller is a web-based synchronization component which enforces valid communications while supervising more transitions in the composed system, thereby improving scalability [WR88] [SL98]. Thus, we enforce transaction support QoS attributes during the service discovery process, such as *consistency*, *isolation*, and *durability* while achieving *dependability* QoS attributes such as scalability. After creating a framework that can manipulate the dynamic attributes for efficient service discovery, the next step is to study the type of attributes and when these attributes need to be monitored. These dynamic attributes vary in different domains and under different situations; that is they are context-sensitive. Chapter 4 demonstrates how SDDS can effectively recognize and monitor these attributes in a given context in order to select an appropriate service.

4.3 Summary

To achieve high-quality dynamic web service discovery, QoS attributes must be taken into account. Since the value of QoS attributes varies in different domains and in different situations, we proposed a model to select appropriate services considering QoS attributes. Our model, static discovery dynamic selection (SDDS), is an effective technology that measures the relevance of services to a particular context. In this chapter, to enable use of *transaction support*-related QoS attributes, we model the behavior of the whole system using constrain automata. This model enabled us to better understand all the transactions among the SDDS components. These transactions need to be considered for efficient implementation. For instance, we identified transactions that caused deadlock/livelock and therefore jeopardized data integrity, consistency, and durability in our databases. To avoid these transactions, we made provisions for a controller that could guarantee the integrity and consistency of data. The controller filters communication among SDDS components to maintain transaction support-related attributes while ensuring efficiency.

Chapter 5

Self-Adaptive Management of SDDS

5.1 Introduction

In the previous chapters, we specified the importance of characterizing services both by the functionality they offer and the quality of the service they provide (e.g., availability or response time). While functionality is typically fixed, quality of service (QoS) changes over time. Existing web service discovery methods assume that the characteristics of a web service are static; therefore these methods provide no support for dynamic attributes [LNZ04]. We define the set of dynamic attributes that accompany the web service discovery request as the *context* of the request. This context information can be used to determine the *application-specific* QoS attributes (cf. Section 1.6) or consists of these QoS attributes. For instance, room temperature is a QoS attribute for a climate-control system, whereas the same room temperature can be context, that is, information as part of physiological context information obtained from your body sensors.

Current web service discovery mechanisms return many irrelevant results because they do not account for context. Our research aims to overcome these limitations by tracking and considering this additional information. We show that these attributes play a crucial role in finding relevant services by enabling the service discovery to dynamically change as new context information becomes available. Our algorithm is based on mechanisms similar to those that benefit autonomic computing. The aim of our approach is to increase the quality of service discovery which should ultimately result in higher consumer satisfaction (i.e., QoE), since many fewer extraneous results are returned. When consumers look for services that meet their requirements,

they also care about the quality of service they will receive. Without assurance of expected quality, the plethora of results returned from a typical registry is not particularly useful [FK05] [KR04]. We therefore contend that QoS is part of the context of the request; its expected value determines the kind of results that should be returned [LRMD08]. In short, the combination of the consumer's network connectivity, requirements, and QoS form the context in which the request is formulated. This dynamic context needs to be evaluated in a timely fashion and be taken into consideration when servicing the discovery request. An extended review of the literature on context taxonomies and classification is explained by Villegas et al. [VM10].

We modify the service discovery protocol to communicate context along with the request in order to guide the web discovery engine towards relevant results. Our context model is extensible, and context information can be specified by the provider, the user, domain experts, or gathered from the sensors. Several context-aware techniques have been developed [TD09], but none of them apply directly to web service discovery. These techniques are designed primarily for tightly coupled systems, and are not suitable for all environments. Our proposed approach to high-quality service discovery aims to overcome these limitations by considering knowledge about the situation in which the request was issued [MRLD10] [KP09].

We employ autonomic computing technologies to manage dynamic context information in the service discovery process. The discovery process adapts to changing conditions automatically by updating the services registered in the repository. In other words, a control loop monitors some resources (i.e., dynamic attributes) and autonomously keeps the repository updated. Besides making our system self-adaptive, autonomic computing aids in identifying the quality selection criteria based on service, user, and domain contexts [ST09a]. Finally, the chosen criteria can be used to filter the relevant services.

As discussed in the previous chapter, the overall objective is a model to register, collect, measure, and aggregate dynamic attributes, enabling storage, as well as continuously detecting changes and identifying anomalies.

Here, our objectives include the following methods and algorithms that aim to advance the current state of the art in QoS management:

- A means for the service provider to publish and update QoS information to reflect the context of service.

- A means for consumers of a web service to add constraints to the search oper-

ation (e.g., preferences) that reflect the context of the request. The user can express their preferences simply.

- An enriched method to automatically evaluate the service and the consumer's context in order to identify the selection criteria automatically, such as type of service.

- A self-adaptive service discovery process that realizes domain specific criteria. New domain specific criteria can be added and utilized to evaluate web services without changing the underlying computational model.

- A filtering mechanism for using dynamic attributes as a secondary criterion for service selection. Chapter 5 explains this algorithm in details.

Now, we first give an overview of a few of the existing context-aware systems, followed by a more detailed explanation of how SDDS uses the context information to evaluate the quality attributes.

5.2 A Review of Context-Aware Systems

Context-aware web services are mostly studied as part of context-aware systems. Dustdar and Truong have performed a comprehensive survey on context-aware systems [TD09]. One example of a context-aware system is the Java Context Awareness Framework (JCAF), a framework and programming API for developing and deploying context-aware applications [Bar05]. They manage context information with separate services, known as context services. The client can publish and retrieve the context information from the context services. All the communication in JCAF is based on Java RMI, and there is no automatic discovery mechanism for finding other context services.

The AWARNESS project is another infrastructure for developing context-aware systems that are targeted at mobile networks, and in particular for the healthcare domain [Weg05]. The architecture is composed of three layers. The network infrastructure layer is responsible for accessing communication networks. For instance, in the case of a medical emergency, more network resources can be allocated or context describing the availability of network resources to an application can be provided. The service infrastructure layer is responsible for the delivery of services required by

the applications or end-users. The mobile application layer provides end-user applications.

CAMUS is a middleware system for acquiring, interpreting, and disseminating context information to create context-aware network-based robots [KCO05]. It provides a means for modelling the environment in which the robot provides services. Another context-aware architecture is SOCAM [GPZ05], a distributed middleware designed to support the acquisition, discovery, and interpretation of various contexts. It is similar to the context gathering mechanism in our approach: context information can be sensed through physical sensors, defined by users, or interpreted through context reasoning. However, the system is not targeted at web service discovery and is implemented based on RMI.

These existing context-awareness mechanisms are either designed for a closed and tightly coupled environment or not based on web service system techniques, and are therefore not directly applicable in the web service domain. A description of some of the context-aware systems designed for web-based environments follows.

Omnipresent is a location-based context-aware system that is modelled based on OWL. In this project, several services are developed to manage geographical information. Users can register themselves in the system and include their location in their profile through a web form. This information is sent to the location-based service, and the user may interact with the service. The location-based service function calls are implemented using the SUN JAX-RPC package that is designed for mobile devices with limited capacity [dAdSBdS$^+$06].

Another approach for context-aware service oriented architecture is CA-SOA [CYZ06]. This model is aimed towards ubiquitous service discovery that can support contexts via different components: service agent, user agent, and broker agent. This approach stands out because it is equipped with a real-time context acquisition method and comes with a rule-based algorithm for context matching. However, the matching algorithm is predefined and cannot change according to context.

CoWSAMI [AZI$^+$08] is a middleware infrastructure that enables context awareness in open ambient intelligent environments with the objective of integrating loosely-coupled context sources and aggregators at runtime. CoWSAMI provides a means to export interfaces that comply with the standard web service architecture. For dynamic discovery of context sources, a distributed mechanism is provided to maintain data freshness of context information as context sources change. These context sources can be associated with different discovery policies.

We attempt to extend existing context-aware service discovery and increase adaptability into web services. We encourage the use of autonomic computing (AC) in context-aware service discovery [KC03], to improve context reasoning. Thus, our model can be characterized as an adaptive, context-aware, and service-oriented architecture that uses autonomic control loops to determine *application-specific* attributes at runtime.

5.3 Autonomic Computing

Coordinating the interactions among different components at runtime becomes a complicated task where hundreds of devices are connected to the Internet (i.e., pervasive computing). For such systems, self-governing components are beneficial, as they are capable of making timely decisions when changes and conflicts occur at runtime [KC03]. In doing so, autonomic computing is propitious to adjust the system operations to support runtime conflicts and changes. This is referred to as self-management. Thus, the low level decisions can be taken by a computer at runtime, rather than humans. In addition, autonomic systems can reconfigure themselves based on policies that are identified in their high-level configuration. If the business level objectives change, then the system can self-configure in accordance with objectives to accomplish the desires. For instance, if a new component is added to the system then the other components may need to change their own behaviour to use the new ones, this is known as self-configuration. Another important aspect to support runtime changes (i.e., integrate the components in a large system) is the question of performance. Autonomic systems are continuously making decisions to maintain performance and cost at a desired level. For instance, performance may be able to maintain itself by tuning system parameters. This is referred to as self-optimization. Self-healing is another beneficial aspect of automatic systems to diagnose and repair the problems that can come both from hardware or software. Finally, in the face of any malicious attack, automatic systems are self-protected to defend the whole system, and also anticipate problems and take steps to avoid or mitigate them.

Autonomic systems consist of autonomic elements. An autonomic system could include one or more managed elements such as a CPU or a large legacy system. These elements are controlled with a single manager. The autonomic manager mitigates human responsibility by controlling the managed elements. The autonomic manager includes components for monitoring, analysis, planning, and execution, that all these

elements are sharing the knowledge of the environment, service level agreements, and other related information. The managed elements are monitored, and the monitoring component filters the element's data. Then the refined data is processed and analyzed by an analyzer, and a plan is constructed accordingly and executed to achieve the high-level objectives.

An important contribution of IBM is the autonomic manager (MAPE-K loop) as depicted in Figure 5.1. The MAPE-K loop controls the managed element by implementing an intelligent control loop composed of the monitor, the analyzer, the planner, the executor, and the knowledge base elements.

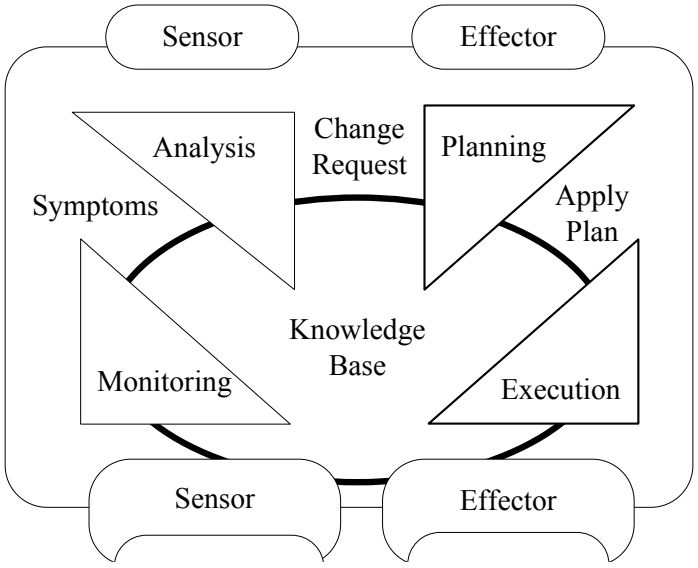


Figure 5.1: MAPE-K loop

We employ the MAPE-K loop to manage the dynamic repository in SDDS to adapt to changing environments (cf. Figure 5.2).

5.4 Autonomic Quality Selection (AQS)

As demonstrated in Section 4.1.1, the service dynamic manager is responsible for handling the dynamic attributes of services in order to best identify service properties. A set of dynamic attribute values are measured in the dynamic manger to evaluate whether the desired values are achieved. This component checks the quality

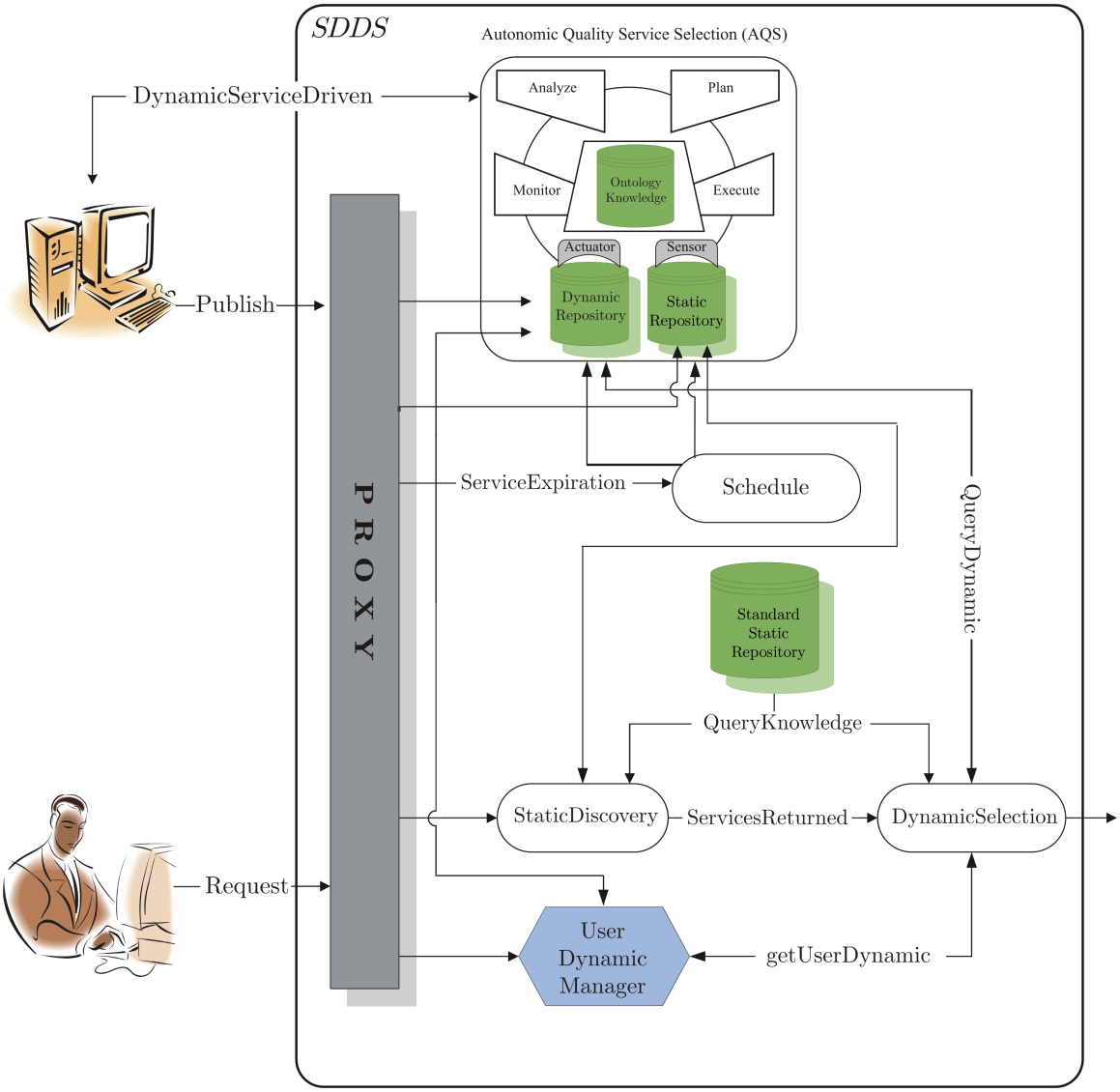


Figure 5.2: SDDS with adaptive quality of service

of the services in the standard static repository and stores the services in the dynamic repository. This is being processed by a monitor-analyze-plan-execute (MAPE-K) autonomic loop (i.e., IBM's autonomic manager) [HM08]. In our research, we employed MAPE-K to automate quality checking through an intelligent attribute recognition, measurement, and evaluation process. The measured output values of dynamic attributes impact the dynamic manager, causing the desired effect (i.e., switch between services). Figure 5.2 shows how autonomic quality service selection (AQS) monitors the standard repository for changes through sensors, comparing to the current state

of the static repository. When new services are published in the standard repository, the entries are analyzed—a process known as attribute recognition—using information such as user context, service context, and domain knowledge (i.e., knowledge base). Here, we exploit context information at query time for the attribute recognition process. Our dynamic manager employs the MAPE-K loop to adapt to the current context during the service discovery process in order to improve quality. To evaluate the service quality attributes, the attributes are observed in accordance with the context. In other words, the quality attributes are crucial properties when evaluating the self-adaptability of a service discovery system. There have been several surveys on self-adaptation properties and their relationships to the quality of software systems. Our system is a simple example of this concept as it applies to web service discovery. For further study, Salehi and Tahvildari studied a hierarchical view of self-* properties and their relation to the quality of software systems [ST09a].

The planning phase, which is policy-driven, identifies a strategy to measure the relevant attributes in order to evaluate the quality of the service. Finally, executors evaluate the service according to the selected strategy by running through the recently created plan. In the end, effectors update the dynamic repository with results of evaluation. Furthermore, the dynamic repository is updated periodically to ensure that the currently tracked attributes fulfill the requirements of the current situation.

5.4.1 SDDS Knowledge base

The knowledge source that drives this entire loop can be broken down into the following components:

- Domain Information Base (DIB): Contains measurable dynamic information related to the semantic domain. It maintains a default weight vector, defined by domain experts, for each service type.

- Policy Information Base (PIB): Maintains guaranteed quality requirements in a machine-readable format referred to as a service-level agreement (SLA) [LNZ04] [BLM08]. This information is specified by the policy as an additional attribute to be used in planning the evaluation strategy for a given service. SLAs govern the quality of the services that consumers expect from the service provider. We formulate a set of service discovery requirements as a collection of service level objectives (SLOs). For instance, one of the SLAs that contains the one

SLO here is: *availability of the web services in finance/stock domain should not be less than* %90. For simplicity, we specify a set of QoS attributes and their target values as well as their measurement mechanism in SDDS's SLAs. This information can be updated and reused to improve the quality of service discovery. Notably, no penalties for not meeting the quality objectives of service discovery are included, however the services that couldn't meet the objectives could not register, or they have to be removed from dynamic repository.

Some of this information is not available in advance and should be captured, decided and operated upon at the time of service discovery. In contrast, most existing service selection mechanisms are based on predefined information and are hard-coded.

5.4.2 SDDS QoS Computation

To establish valid and reliable web service discovery, QoS is one of the promising features that gives consumers confidence when using the service. Availability, response time, and accessibility are the ubiquitous set of QoS attributes that can be measured in our framework. As we discussed in Section 1.6, these attributes belong to the *performance* and *dependability* category of dynamic attributes. To illustrate the importance of the above-mentioned attributes we introduce an example from telemedicine [LNZ04].

Consider a patient who needs to have frequent contact with her doctor. Using autonomic quality selection, the patient's vital signs such as blood pressure and temperature can be monitored, analyzed, and in case of anomalies, the system can automatically make a decision to contact the doctor, exchange data with the doctor, and enable some form of virtual person-to-person interaction. Telemedicine requires effective and reliable communication due to the potential for tragic effects on a patient's health should communication fail. Availability and accessibility are two major QoS concerns, due to the patient's need for immediate contact with the doctor. Cost is determined by how much the service provider charges consumers. Security is another important attribute, because a patient's personal information must be protected and only disclosed to the intended healthcare provider.

We represent our measurement mechanism for a couple of these attributes in the general domain based on these studies [ZBD+03] [LNZ04] [TRPA06]. These measurement techniques are restored from PIB and can be upgraded or updated. Simply, the computation can be performed in a context-aware manner, where the availability can

be measured differently in different contexts.

Some of these QoS computations are outlined below.

- Accessibility is the capability of serving a request. It can be measured with the following formula:

$$\mathrm{A_{ccessibility}} = \left(1 - \left(\frac{S_{(downtime)}}{S_{(uptime)}}\right)\right)$$

- Availability is the probability that the service s is accessible over time interval t. The value of t depends on the domain. In more frequently accessed applications such as stock values, a small value of t provides a more accurate result. A service can be available but not accessible at query time due to a high volume of requests. The availability ($\mathrm{A_s}$) is computed as follows:

$$\mathrm{A_{vailability}} = \frac{A_{ccessibility}}{t}$$

- Response time is the time that elapses from the moment that a web service receives a SOAP request (T1) until it sends the corresponding SOAP response (T2). It is measured as:

$$\mathrm{RT} = \mathrm{T1} - \mathrm{T2}$$

- Execution Cost is the value of a service; that is, the amount of money that the service consumer has to pay to the service provider to get a commodity. This attribute is either advertised by the service provider or inquired upon by the consumer. Let s be a web service, then $\mathrm{C_s}$ is the cost of using s.

In our research, we monitored some of these attributes. Notably, AQS is extensible: the new QoS parameters and their computational rules can be added without altering the underlying computation of SDDS.

5.5 Experiments

We implemented the first research prototype of our adaptive, context-sensitive service discovery system as a front-end to a collection of web services accessible through seekda.[1] We gathered services from seekda and published them in the SDDS standard

[1] http://webservices.seekda.com

static repository. The services are chosen from at least two different domains and are mostly WSDL-based. Here, the categories represent the general functionalities of web services. For each service we keep the following function attributes: name, category, WSDL, and description. For experimental purposes, in the seekda collection, we identified 350 sample services from two categories: weather information, and finance in particular stock. To make the retrieval challenging, we look for related services or simply services with similar functionality. Figure 1 shows the categories and number of services registered in the standard repository.

To analyze the impact of dynamic attributes on the quality of web service discovery, we compare the number of services returned for several queries between the seekda and SDDS systems.

We quantified the following scenario:

1. Preliminary experiments show a significant reduction in the number of returned results against the static service repository, while ensuring that all the returned data from the context-aware search is relevant (i.e., same category).

   Query 1: Consumer wants a set of services among many functionally identical ones. SDDS achieves better results by returning a smaller set of web services selected through a set of quality attributes.

   Query 2: Consumer's preferences are either specified explicitly or captured from their context history. To facilitate the interaction between SDDS and consumers, an interface for formulating and submitting queries is provided. First, based on the consumer's query, a set of information appears in the interface. Our objective is to create a simple way for consumer to interact with SDDS.

2. We are aware that services from different domains may have different critical properties that may change under different situations. The system's recognition of such properties through analyzing domain and both user and service context is referred to as adaptability in our system. To evaluate the effectiveness of our adaptive approach, we conducted experiments with and without the AQS component.

   Query 3: The critical quality attributes for a given query are automatically chosen by SDDS from the domain information repository. This information is basically static, specified in advance, and thus is neither flexible nor scalable. For a given query from different consumers, the results from SDDS always return

a smaller set of results compared to Seekda. However, for the same query with AQS, the results are different for each category and are based on context as well.

In general, based on our quantitative evaluation, a smaller set of results is always generated compared to the standard repository. Chapters 6 and 7 provide quantitative discussion and evaluation.

5.6 Summary

This chapter discussed our approach to providing high-quality results in web service discovery. We proposed that dynamic attributes are the most significant contributors to reducing the set of results returned while maintaining quality. We introduced dynamic attributes as part of the context that affects web service discovery quality. To adapt to the current conditions of both services and consumers, we employed an autonomic manager. The autonomic manager can monitor dynamic attributes and user context to obtain quality selection criteria. This manager resembles IBM's MAPE-K loop in that it can perform attribute recognition, measurement, and evaluation using both context and domain information. Then the chosen criteria are used to constrain the discovery of relevant services.

Performance and *dependability* are two categories of dynamic attributes that can be handled by an autonomic manager. For instance, availability and accessibility belong to the dependability category, and can be measured through different mechanisms depending on the domain. The monitoring mechanism is stored in DIB and PIB. Importantly, these databases can be updated without altering the underlying system.

The effectiveness of our method was demonstrated with a prototype implementation of our model. We experienced that increasing the adaptability of web service discovery by including context information provides a significant reduction in results returned compared to traditional static web service discovery methods.

Further research is discussed in Chapters 6 and 7 to establish the matching algorithms that can handle multiple attributes and work under different matching criteria, as well as to find a good ranking mechanism for the discovered web services.

Chapter 6

DYNAMIS: Effective Context-Aware Web Service Selection Algorithm

6.1 Introduction

With the increasing proliferation of web services, selecting the best one for a consumer's needs can be an overwhelming task. Service consumers naturally expect automated and context-aware service discovery and selection techniques. Current mechanisms based on Information Retrieval (IR) techniques, particularly string similarity, are of limited use because the service descriptions on which the similarity is calculated are often short, ambiguous, and syntactically incorrect.

A promising alternative is to evaluate the suitability of each service using *quality of service* (QoS) attributes. This is challenging in practice, because QoS attributes can be difficult to measure, potentially fluctuate, are context-sensitive, and depend on environmental factors such as network availability.

We use the services of a telescope in an observatory to illustrate our context-sensitive service discovery techniques. In this domain, geographical position, time of day, and weather conditions affect the quality of service selection; these elements contribute to the context in which a picture of the celestial sphere is taken. For instance, one of the attributes specific to this domain is the visibility of the sky—a weather property; another is the relative position of a specific star with respect to the telescope and the darkness of the sky—properties of both geographical position and time. Contrast, the range of brightness in an image, is another important QoS attribute for telescopes—the significance of the contrast attribute varies from user

to user. In addition, users may request a picture with a specific composition—the number of objects captured in a photograph, and whether the image is to be in color or black and white.

On certain days, particularly those when the telescope is offline due to maintenance, it may be impossible to determine a value for some attributes. In other words, there will be days where photographs are unavailable or the quality is too low to recognize celestial properties.

This telescope example illustrates why smart evaluation based on QoS attributes is useful and necessary to improve service selection. One approach is to ignore the fact that attribute values fluctuate and merely assume that all functionally-equivalent services are equally valuable. The user then has to manually sift through the available services, even extreme examples where one service outperforms or matches another on every measurement. This frustrates users and lowers the adoption rate of the service discovery technique.

The best techniques used in practice come in two flavors—those that do not consider QoS attributes [CFV07, CZC08, LCS97, VSS+05], and those that take a snapshot of the values into account [CYZ06, dAdSBdS+06, TD09]. In the former case, the real potential of QoS attributes is squandered. In the latter case, services are misrepresented by temporary fluctuations of QoS values or harshly penalized because the mechanism for obtaining those values fails. Both cases will inevitably arise in practice.

In this chapter we investigate two approaches to deliver a well-chosen subset of services, each with its own advantages: context-sensitive ranking by *aggregation* [FLN01]; and context-agnostic selection by *skyline* [TZT08, BKS01]. Both approaches require reliable, non-null values for each QoS attribute and, therefore, break down in the presence of dynamic attributes.

To satisfy the consumer's need for timely, well-chosen web services, new ideas are needed. This research contributes to the area of web service discovery; we present a novel technique for generating reliable values for QoS attributes that can withstand temporary fluctuations (regardless of their magnitude) and null values. Our approach extends the aggregation and skyline techniques to deal with dynamic attributes effectively. In this chapter, we present the design, implementation, application, and evaluation of our smart QoS-based service selection techniques.[1]

Section 6.2 provides background information on skyline and aggregation. Section 6.3 details the problem of web service selection using dynamic attributes. Sec-

[1]This work has been submitted for publication [PDTM12]

tion 6.4 discusses related work on web service discovery and in particular, database research on aggregation query and skyline techniques. Section 6.5 presents our context-aware web service selection algorithms based on dynamic attributes. Finally, Section 6.6 presents conclusions.

Table 6.1: An example set of telescope web services with multiple dynamic QoS attributes, as measured on three consecutive days.

		Days			
Telescope	Context/QoS Attributes	1	2	3	4
T1	functional	Y	Y	Y	Y
	availability(%)	20	40	50	60
	user preference/contrast	8	9	8	5
	time/dayLight	8	9		5
	user preference/composition	8			5
	latency(ms)	6	2		3
	weather/visibility(%)	70	90	90	80
	geographical position/viewing angle	8	8	4	5
T2	functional	Y	Y	Y	Y
	availability	85	93	65	89
	user preference/contrast	8	9	4	5
	time/dayLight	8	9	5	5
	composition	8	9	9	5
	latency	9	8		9
	weather/visibility	90	80	80	90
	geographical position/viewing angle	8	6		8
T3	functional	Y	Y	Y	Y
	availability	30	60	30	60
	user preference/contrast	8	9	4	1
	time/dayLight	8	9		1
	composition	8			3
	latency	5	7	4	1
	weather/visibility	70		70	50
	geographical position/viewing angle	8	9	9	9

6.2 Background

There are two primary approaches to service selection: *aggregation* [IBS08], aiming to identify the services most appropriate to a given context and *Skyline* [TZT08], aiming to identify, in a context-agnostic manner, the best services. Both encounter practical difficulties in the presence of dynamic attributes. The remainder of this section describes both approaches and the root causes of those difficulties.

6.2.1 Aggregation

An *aggregation query* (a.k.a. *top-k* or *ranked query*) ranks each tuple of a database relation by combining the attributes with a single, aggregate scoring function and then reports the highest scored tuples (i.e., either the k highest ranked tuples or the ones whose score exceeds a predetermined threshold) [IBS08,FLN01]. Consider web service s, with numeric attributes $\langle a_1, \ldots, a_i, \ldots, a_m \rangle$, the rank is computed as follows. First, the user context q is modeled by weights for each QoS attribute, $\langle q_1, \ldots, q_i, \ldots, q_m \rangle$, where the magnitude of each q_i reflects the importance of the i^{th} attribute within the context. Then the context-aware score for a service is determined as $\sum_{i=1}^{m} a_i * q_i$. Then the *rank* of service s is the number of other services with a score higher than that of s. This offers the advantage of user customization since context is built into the ranking mechanism.

6.2.2 Skyline

A disadvantage of the aggregation query approach is that the reduction of m attributes to a single score often reduces the accuracy of the retrieval process in multi-criteria decision-making [SSS+09].

The *skyline* of a relation is context-agnostic—in contrast to the top-k—which proffers the advantage of circumventing the non-trivial task of modeling user context numerically. Conceptually, skyline is the subset of services which are top-ranked for *a selected* user context [TZT08, BKS01].

We denote the numeric attributes of service s_i as $(s_i^1, \ldots, s_i^k, \ldots, s_i^m)$. Service s_i is said to *dominate* another service s_j if it has an equal or better value for every attribute a, and a strictly better value on at least one of those attributes; that is, if $(\forall k, s_i^{a^k} \geq s_j^{a^k}) \wedge (\exists k, s_i^{a^k} > s_j^{a^k})$. The *skyline*, then, is the set of those services that are not dominated by any other service. Unlike aggregation, this results in a

set of *incomparable* services, rather than a total ordering and, unlike aggregation, the cardinality of the resulting set is entirely data-dependent as a result of being context-agnostic.

6.3 Practical Challenges of Dynamic Service Discovery

Some practical challenges arise when implementing dynamic service discovery because the attributes are, well, dynamic. Ignoring these challenges comes at a definite cost, where a service best suited to a user context can be penalized because the evaluation mechanism is insufficient. Using our telescope example we examine how these challenges arise and specifically why they constrain the applicability of the aggregation and skyline techniques.

6.3.1 Expanding the Telescope Example

Assume that there are hundreds of telescope web services available around the world. To a user interested in photographing a particular star, only a subset of these services are of particular use. Some telescopes are located in the wrong hemisphere. With others, the user may not have a service-level agreement (SLA) that permits her to use that telescope. By considering all such attributes, ideally one can reduce the set of telescopes to a small subset which are *functionally equivalent*. This is referred to as functional selection—capturing a set of predefined values. Consider Table 6.1 depicting attributes and their values captured on four different days for four different telescope services. In this table, the functional attributes are assumed to be satisfied indicated by a Y for yes on all three days. After functional selection, the service subset needs to be ranked and possibly further pruned with respect to their dynamic, quantitative QoS attributes.

The values of all these attributes fluctuate over time, perhaps even wildly, which complicates the matter of selecting the web service most likely to deliver the highest quality photographs. A knowledge base can maintain an expert-derived threshold value for each attribute, in each domain—as shown in Table 6.2—to ensure an adequate quality of service. But this approach still suffers from practical problems.

Because the values are captured or possibly computed at runtime, they may not always be available, perhaps they did not arrive on time, or were lost due to network

problems. Consider what happens when the observatory weather station cannot be reached, the service provider does not provide a value for contrast, or a power outage takes out the sensor that measures these values. In these scenarios the telescope should not necessarily be discounted from consideration, but neither the skyline nor the aggregation function is well-defined over these *null values*.

Null values are not the only concern. Consider visibility: even on a superb day there can be patchy cloud cover. A particular measurement of visibility, one that captures the precise value at an exact moment in time, might signify that the visibility is poor, when in fact there is only a momentary lapse before the cloud passes over. Another issue is latency: perhaps a particular telescope is generally slow to respond, but right after the service has been restarted there are few connected users and a measurement at this time would overestimate the quality of the service.

6.3.2 Manifestation of the Challenges

Both aggregation and skyline have practical issues when applied to dynamic web service discovery.

Null values are perhaps the most debilitating because they introduce increased incomparability. The summation on which aggregation depends is ill-defined in the presence of null values. The biggest drawback of skyline is that it often is insufficiently selective, especially as the dimensionality increases; introducing more incomparability via null values only serves to exacerbate this problem. A conceivable approach is to replace null values with zeros or estimates, but zeroing values can be unnecessarily harsh, especially when the null arises from network issues with an external data source.

Temporary fluctuations and unreliable measurements can significantly affect the skyline. A service that is consistently part of the skyline because of a typically maximal value on one attribute could be dropped at runtime.

The context should affect the relevancy of dynamic attributes. For instance, availability is usually extremely important for quality-driven service selection. However, the importance of this attribute can be tempered if the user decides to wait for a higher quality service that is temporarily unavailable rather than settle for a lower-quality service that is immediately available.

Ideally, one could apply both well-established selection approaches—aggregation and skyline—to dynamic web service selection; however, the challenges detailed above

obfuscate QoS comparisons between services. The definition of skyline (or, more precisely, of dominance) requires a well-defined, ordinal relationship (i.e., the comparison operators $<$ and $>$ must be well-defined for the attribute values of any two services). Null values invalidate this, because they are incomparable to non-null values. Snapshot values distort this, because they cause the ordinal relationship among services to fluctuate wildly. Thus, we need to restore the relationship somehow when null and fluctuating values arise.

For aggregation, the problem is even trickier, because the scoring function relies on attributes that are real-valued, not just ordinal. Thus, it is important that, for example, the difference between 3 and 1 is larger than the difference between 3 and 2, since this will affect the aggregate score.

Naturally, any technique that supports the case of aggregation can be used to re-establish the concept of dominance, because the ordinal operators $<$ and $>$ can be applied to any real-valued attribute. The goal is then to derive a reasonable value for null attributes to assess the magnitude by which the values, if known, would differ from the other known values. Using this approach, we can continue to apply the skyline and aggregation techniques.

There is another problem with dynamic web service selection as it relates to skyline and aggregation. As a first step, a set of web services must be narrowed to those that are functionally equivalent and functionally appropriate. This step must be done first, otherwise there is no guarantee that the top-k services match the functional requirements. The consequence of this first selection is that neither an index nor a pre-sort-based algorithm can effectively resolve the aggregation or skyline component of the query because the data structures are designed to query the entire dataset efficiently, not the particular, ad-hoc subset of it. This characteristic establishes a scenario wherein the results of the functional selection are fed to the aggregation (equivalent to skyline) selection.

As such, an important two-part research problem naturally arises: (1) how can one establish a reasonable replacement for null and rapidly fluctuating values in order to apply aggregation and skyline to a set of services; and, (2) given that such replacements can be produced, which algorithms will best support functional selection with aggregation and skyline?

6.4 Related Work

In this section, we review related work spanning several disciplines. In particular, we describe research to support web service discovery in Section 6.4.1, relevant literature on aggregation querying in Section 6.4.2, and skyline research, paying particular attention to research that assumes a streaming setting in Section 6.4.2.

6.4.1 Web Service Discovery

The ranking of services by QoS properties has been approached in two ways: (1) with regards to only static attributes [KD09, TD09]; (2) by taking a snapshot of the current values of the attributes [SSS+09, EHMR10]. The former squanders the potential to improve the user experience. In our telescope example in Table 6.1, capturing a measure of the *availability* of telescope service T3 on Day 1 and using that information to evaluate the service on Day 4, when the value has changed substantially, is misrepresenting the reality. Nonetheless, the work done here is still important, because it introduces the idea of using aggregation querying in the context of web service discovery.

Regarding the second option, capturing instantaneous measurements of QoS properties ignores the practical issues that arise naturally when using dynamic attributes. For example, on Day 3, the latency measurement for T2 is null, but one should be lenient because historically this attribute performed well.

Skoutas et al. [SSS+09] introduced the concept of dominance and skyline into the field of web service discovery. They examine every subset of the QoS attributes and count the number of subsets in which a given service is dominated. The services can then be ranked in ascending order of score. In this way, they combine the notions of skyline and aggregation without having to model user context. They also present several algorithms known as *TKDD, TKDg,* and *TKM* to determine the top-k web services based on the dominant scores. They improved performance by having boundaries for each object consisting of the lowest and highest values of the dominating scores. Therefore, instances (i.e., attributes of a service) are maintained in three lists: *maximum*, *minimum*, and *current* values of instances for a service. This approach helps with static service discovery where values of attributes are available, but cannot support missing values.

Hadad et al. [EHMR10] proposed an algorithm—TQoS-driven selection—that is embedded in the transactional service selection for web service composition. The

algorithm is basically designed for local QoS optimization—that is, the selection of the qualifying services for each activity that fulfills the transactional requirements. The algorithm's input is a workflow that includes a set of activities to fulfill a request. The output is a set of the services for each activity in the workflow with similar transactional properties but dissimilar nonfunctional properties. The TQoS selection uses user preferences as a weight assigned to each QoS criterion. The web service is then scored through a conventional scoring function to rank services and find the one with the maximum score. If there are multiple services with the same score, one of them is chosen randomly. Similar techniques are applied to select the maximum score path for web service composition.

Possibility theory is another approach that has been used to tackle the issues of skyline for uncertain QoS attributes in web service discovery [BBH12]. This work is based on a possibility distribution that contains all possible values of QoS attributes for each service. Then two skyline algorithm extensions—*pos-dominates* and *nec-dominates*—compute whether service s_i dominates service s_j in accordance with the threshold values computed from the possibility distribution. As far as we could tell, this work has not been applied to dynamic web service discovery.

Dustdar et al. [LRMD08,MRLD10] proposed the QoS-aware VRESCO runtime environment for dynamic binding, invocation and mediation. They describe functional attributes in their service meta-data model. These QoS attributes can be either specified manually using a management service, or measured automatically, and integrated into VRESCO. For each service, a schedule can be defined that specifies when the monitor should trigger the measurement and publish their values. The average QoS values can be aggregated based on the information stored in the QoS events. Unfortunately, this approach cannot support fluctuating or unavailable values. The challenge for this algorithm lies in capturing and computing dynamic attributes at query time.

SDDS (Static Discovery and Dynamic Selection) by Pahlevan and Müller is an initial framework that takes static and dynamic attributes into account during web service discovery [PCM10,PM09]. We examined context-sensitive QoS attributes [PM10], and their use in ranking services using skyline and aggregation algorithms [PCTM11]. Table 6.1 shows an example with SDDS-determined attributes enumerated in Table 6.2, identifying seven example QoS attributes.

The values for these attributes are computed in a variety of ways; for example, visibility is obtained from weather services such as *AccuWeather*.[2] Other attributes

[2] `http://www.accuweather.com/`

Table 6.2: Example: SDDS threshold knowledge base

	Parameters	Threshold
Photography of star by telescope domain	availability	80
	contrast	8
	dayLight	8
	composition	6
	latency	1
	visibility	90
	viewing angle	9

that refer to context, such as environmental conditions or user preferences, can be obtained via sensors or directly from the user.

6.4.2 Database Research on Aggregation and Skyline

Aggregation Querying

There is a significant body of work covering how to determine, given a user context, the top-k services [IBS08]. Here we summarize the two most relevant approaches. One is to use a traditional scan-based algorithm to process all candidate tuples in the database, calculate their scores, and return the k tuples with the highest grades. This involves evaluating every service for every query.

The *Threshold Algorithm (TA)* introduced by Fagin et al. [FLN01] improves on the scan-based alternative by detecting opportunities to terminate earlier (i.e., before scanning every tuple). They exploit the fact that each attribute's data source can often be provided in sorted order and thus one can determine when no further tuples can score higher than the ones already seen. The idea was refined further by Nepal and Ramakrishna [NR99] and Guntzer et al. [GBK00] to handle data that is homogeneous with inherently fuzzy attributes. Other extensions to TA have been proposed to decrease access costs and deal with data access in situations where certain sources cannot provide sorted access [BGM02].

As discussed above, a bigger hurdle for threshold algorithms in dynamic web service selection settings is the ill-defined behaviour for when null values occur. One approach to alleviating the null value problem is the *probabilistic top-k query* algorithm, introduced by Cheng et al. [CKP03], in which each attribute is modeled as a range of possible values from a probability density function (PDF). The motivating example for this research is the modeling of moving objects, where at any point in

time, the location is within a certain range d of its last reported location value. If the location changes, then the sensor reports its new location value to the database and changes d accordingly. The PDF can be an equal chance of locating the object anywhere in the range of d, and can be estimated using time-series analysis [Cha89].

In many applications, it is only necessary to know whether the probability exceeds a given threshold. Accordingly, a *probabilistic threshold query (PTQ)* is a variant of probabilistic queries where only answers with probability values over a certain threshold are returned [CXP+04]. These methods are more expensive to evaluate than their traditional counterparts (TA) due to the calculated probabilities that accompany the answers. Methods to improve the efficiency and reduce the cost of such queries have been explored. For example, to speed up the otherwise inefficient use of an interval index to answer the PTQ, it is possible to improve the interval indexing (i.e. R-tree) by expanding the probabilistic query to its internal node, thus avoiding having to compute the probability values of those intervals which cannot satisfy the query [CXP+04].

These approaches cannot be applied in the dynamic web service selection setting because they depend on the construction of an extensive index structure. As mentioned before, the set of web services will already have been subjected to an earlier, functional selection, which renders indices useless.

Skyline

Borzonyi et al. [BKS01] introduced Skyline with the canonical example of a hotel reservation system, where the skyline are all hotels not worse than any other hotel in terms of both the distance to the beach and the price per night. The skyline technique has one key drawback. First, the cardinality of the skyline operator cannot be controlled—it may include every tuple in the worst case and, more often than not, performs close to the worst case if attributes are uncorrelated. This is especially likely in domains with multiple QoS attributes, because the skyline cardinality is subject to the *curse of dimensionality*—many dimensions which need concurrent optimization [HDWX08]. Additionally, the traditional skyline operator is designed for static dimensions, such as distance and price [HLOT06], whereas dynamic service discovery is dealing with attributes that fluctuate over time.

To take advantage of the promise proffered by the dominance concept in multi-criteria service discovery, refinements are needed to align with dynamic service se-

lection. Xia et al. [TZT08] presented the concept of ϵ-skyline to control the size of the skyline by ranking the skyline result, and finally reflecting the user preferences at different dimensions. They assign weights to dimensions to rank the skyline results and involve user preferences to rank and control the size of the skyline. However, one of our objectives in studying skyline in service discovery is to offer a context-agnostic option to overcome the problem of modeling user context (e.g., user preferences).

Another technique to address the curse of dimensionality is *k-dominance*, which relaxes the idea of dominance to k-dominance. Huang et al. [HDWX08] proposed algorithms, such as δ-skyline employing the concept of k-dominance to rank results without requiring user context. The definition itself suggests its efficacy in controlling the cardinality of the skyline—that is precisely what the user-defined δ is supposed to indicate.

Papadias et al. introduced another extension of the skyline query, the *top-k dominating query* [PTFS05]. It combines the advantages of top-k and skyline queries to control the size of the output independent of the dimension scales, and without having any ranking functions. Yiu et al. [LN09] conducted a study with algorithms on top-k dominating queries and proposed to sum the number of points it dominates from all subspaces.

There are approaches that extend the traditional skyline technique to support dynamic service selection. *Continuous skyline query processing* is able to handle dynamic datasets that arise in domains such as location-based services. Instead of snapshot queries, continuous queries require continuous evaluation as the query results vary with the datasets. In terms of location-based services, the movement of a service object changes the skyline according to the service's current location. In real-time applications, these situations become more complex due to the movement of all the query points. For web service discovery, a hybrid algorithm that combines the salient advantages of top-k and *continuous skyline query processing* have a good chance to be effective when the portfolios of the dynamic datasets need to be refined without acquiring user context.

In dynamic service discovery, the services and also the dynamic attributes are constantly added, removed, or changed. The dynamic repository in our SDDS approach is updated periodically and again at query time, verifying whether the registered services are still qualified or not by checking their dynamic attributes. Thus, we regularly need to compute a skyline for the data points that are valid at query time. Huang et al. [HLOT06] proposed the first work on continuous skyline queries in the mobile

environment. Here, continuous query processing is conducted for users by updating the skyline instead of computing a new query from scratch. The possible change from one time to another is predicted and processed accordingly. This method uses a data structure that exploits the spatial-temporal coherence of the problem. Morse et al. present the continuous time interval skyline operator, which continuously computes the current skyline over a data stream [MPG07]. They discuss an algorithm called *LookOut* for evaluating such queries. Each data point is associated with an interval of time for which it is valid. The interval consists of the arrival time and an expiration time for the point. If the service is dominated, no changes are made to the skyline. If any of the services in the skyline are dominated by a new service, the new service is added to the skyline. These approaches are an ideal solution to improve the efficiency of service discovery since they avoid redundant updates; they have yet to be applied in this area of research. Thus, this research encourages the use of similar approaches for service discovery.

6.5 Dynamis Service Selection

Our proposed approach to handling dynamic attributes is complementary to existing web service selection techniques. This is an important point that warrants further elaboration, because it shows how our research can be seamlessly integrated into existing systems. Here we examine the compatibility of our techniques with existing literature and then with existing user behavior.

First, we assume that the system is compatible with our SDDS framework because the thresholds can be assumed to be set by an expert administrator rather than by a user. Although the static nature of such thresholds presents a missed opportunity for efficiency, the system proposed here can handle that scenario well. Second, any static selection mechanism for evaluating which services are functionally equivalent is, in fact, a prerequisite step of our problem. Third, existing methods for dynamic attributes that are based on snapshot measurements are also a special case of our problem.

In this section, we refer the reader to, as an example, the *finviz* trading site.[3] This is a popular stock picking website that allows users to set thresholds for various parameters and uses them to recommend stocks. This illustrates the readiness with

[3] `http://finviz.com/`

Algorithm 1 : Dynamis Aggregation

Inputs *A set s of functionally equivalent web services;*
    A set a of dynamic attributes for each service;
    A vector $\vec{\tau}$ of thresholds for each attribute;
    A vector of weights $\vec{w}$ for each attribute;
    A time t where the evaluation takes place;
    A time window ν over which attributes are considered;
    A scalar k to indicate the number of services to return.
Output *The top-k services that best fit the request*
Procedure DynamisAggregation$(s, a, \vec{\tau}, \vec{w}, t, \nu, k)$
    for each $s_j \in s$ **do**
        $Score_{s_j} \leftarrow 0$
        for each $a_i \in a$ **do**
            $h \leftarrow \text{Histogram}(a_i, t - \nu, t)$
            $Area \leftarrow ART(h, \tau_i)$
            $Score_{s_j} \leftarrow Score_{s_j} + w_i \times Area$
        end for
    end for
    $Dynamis \leftarrow \text{Sort}(Score_s, >)$
    return $Dynamis[1:k]$ ▷ return the top-k scores
end Procedure

which users can adopt a system that requires them to set thresholds (that can be more or less exact) for a web service selection engine. We then propose a solution that addresses these issues in the form of the *Dynamis* algorithm.

6.5.1 Dynamis: Histogramming, ART, Top-*k*, and Skyline

The above issues can be addressed using our proposed technique: Dynamis.[4] Rather than taking an instantaneous measurement of each attribute, we collect the ν most recent measurements in a histogram. In this case, null values only become a concern when all of the last ν measurements are null, at which point it can safely be assumed that there is something significantly wrong with the property.

Neither aggregation nor skyline are defined for histograms (i.e., collections of measurements). A natural approach is to define a function on a histogram to convert it into a sensible value. We define the *area-right-of-threshold* (ART) function, for precisely that. Specifically,

$$ART_s(a_i, \tau_i)$$

[4]Dynamis is an ancient greek word meaning "power" or "force."

is the number of measurements for service s for attribute a_i that are greater than threshold τ_i. The intuition for the *area-right-of-threshold* comes from the idea of integrating in service selection the most important part of the histogram.

Now we formalize both the aggregation and the skyline problems on histograms.

The top-k histogram aggregation problem is to retrieve from the set of services $\mathbb{S}$, with attributes

$$\mathbb{A} = \langle a_1, \ldots, a_n \rangle$$

given a weight vector

$$\vec{w} = \langle w_1, \ldots, w_n \rangle$$

and a threshold vector

$$\vec{\tau} = \langle \tau_1, \ldots, \tau_n \rangle$$

the top-k ranked services with respect to aggregation function

$$\sum_{i=1}^{n} w_i \cdot ART_s(a_i, \tau_i)$$

The aggregation function is similar with the traditional definition of top-k aggregation, except that we have replaced each a_i with $ART_s(a_i, \tau_i)$, the value we compute from each histogram (cf. Algorithm 1).

Similarly, we define the skyline of histograms by replacing each a_i with $ART_s(a_i, \tau_i)$. Formally, a service $s_i \in \mathbb{S}$ *dominates* another service $s_j \in \mathbb{S}$ if

$$ART_{s_i}(a_i, \tau_i) \geq ART_{s_j}(a_i, \tau_i)$$

For each $i \in [1, n]$, and there exists an $i \in [1, n]$ for which the above inequality becomes strong ($>$). The skyline of web services are those not dominated by any other functionally equivalent web service (cf. Algorithm 2).

6.5.2 Algorithmic Considerations

In this section, we review case studies to establish references points using the most efficient existing approaches—top-k aggregation and skyline—for evaluating Dynamis.

Algorithm 2 : Dynamis Skyline

Inputs *A set s of functionally equivalent web services;*
    A set a of dynamic attributes for each service;
    A vector $\vec{\tau}$ of thresholds for each attribute;
    A time t where the evaluation takes place;
    A time window ν over which attributes are considered;
    A scalar k to indicate the number of services to return.
Output *The top-k services that best fit the request*
Procedure DynamisSkyline$(s, a, \vec{\tau}, t, \nu, k)$
    $best \leftarrow \emptyset$
    for each $a_i \in a$ **do**
        for each $s_j \in s$ **do**
            $h \leftarrow \text{Histogram}(a_i, t - \nu, t)$
            $Area \leftarrow ART(h, \tau_i)$
        end for
        $Scores \leftarrow \text{Sort}(Area_s, >)$
        for each $s_j \in s$ **do**
            $Rank_{(s_j,a_i)} \leftarrow$ Position of s_j in $Scores$
        end for
    end for
    for each $s_j \in s$ **do**
        $Scores_{s_j} \leftarrow \sum_i Rank_{(s_j,a_i)}$
    end for
    $Dynamis \leftarrow \text{Sort}(Score_s, <)$
    return $Dynamis[1:k]$ ▷ return the top-k scores
end Procedure

Top-k Aggregation

The most renowned—and efficient—method for handling top-k aggregation queries, especially when attributes arrive from external data sources, is the Threshold Algorithm by Fagin et al. [FLN01]. However, its efficiency evades the histogram setting because it relies on sorted access to each individual attribute. In the case of context-sensitive thresholds, the $ART()$ values change depending on the choices of thresholds, thus altering the sorted order. Consider the following example from our telescope domain (cf. Table 6.1).

Focusing on the *latency* attribute, note that if the user provides a threshold $\tau_\ell = 4$, then the ranking by $ART()$ of the services is (T2, T3, T1) with values $(3, 3, 1)$. If the user instead provides a threshold $\tau_\ell = 8$, then the ranking changes to (T2, T3, T1) with values $(3, 0, 0)$. This demonstrates that to use an algorithm dependent on sorted

access to individual attributes, the $ART()$ and histogram approach may require re-sorting for every query, which is impractical.

Moreover, functional equivalence selection modifies the set of potentially available services. This renders pre-processing or indexing ineffective, and we look at operators that can operate on streams. Sequential scan is a sensitive choice in this context: we maintain a buffer of the k best services seen thus far, and update it. Thus, the new results from functional selection are processed.

Streaming Skyline

Originally, the skyline algorithm was intended for static query points over a static dataset, where there is no temporal data associated with these elements. To handle dynamic datasets, continuous skyline processing is incorporated into service discovery selection to update the skyline at the query time, though this is impractical when all the query points are changing. Another challenge arises because the $ART()$ values change depending on the choice of thresholds in our histogram setting. However, using the stream skyline algorithm, the change in $ART()$ values does not depend on the choice of thresholds. For instance, in our telescope example (cf. Table 6.1), the default threshold value for availability is defined as 80 in this domain. Based on this threshold, our first skyline result is $(\mathrm{T2}, \mathrm{T3}, \mathrm{T1})$. This result is independent from the day threshold. However, our histogram approaches require the day threshold with a value $(1, 2, 3, 4)$ in order to compute the skyline. The $ART()$ will be different depending on what day we choose to take it from. In this case, the skyline result, if the threshold value is taken from day 3 is $(\mathrm{T2}, \mathrm{T3}, \mathrm{T1})$.

6.6 Summary

A large number of web services perform the same function but in different ways or with different qualities, that is, they have different non-functional attributes such as quality of service properties. This complicates the web service search and discovery process and gives rise to the need for an effective, automated, quality-driven service retrieval and dynamic selection algorithm that considers only a qualified subset of the services. With our approach, we strive to serve consumers better and achieve effective service selection from the vast number of services available by narrowing the search to include only qualified services—the top-k services.

Today, service selection by QoS attributes only considers static attributes or with a snapshot of current values, resulting in low-quality, low-accuracy results. To address this challenge, we focused on capturing snapshot measurements of QoS attributes at the time of the query, which necessitates considering dynamic attributes, effectively selecting services, confronting the challenges posed by fluctuating and missing attributes, and the task of monitoring and storing these values. After considering these aspects, we were able to derive a more effective algorithm, Dynamis, without relying on stale dynamic attributes. Our histogram-based approach addresses these challenges that hamper other approaches.

Chapter 7

Evaluation and Results

7.1 Evaluation

In this area of research, evaluation is based on information retrieval systems [TAH06] [LKYL07] [HZ07] [ST09b] [WS03] [SW05]. The test collection includes a set of service offers $\mathbb{S}$, a number of service requests $\mathbb{Q}$, and *Relevance Judgments (RJ)*. The relevance judgments are identified by how, and to which degree, the offer is relevant to a request by human experts. The evaluation is the comparison of the RJ with a system under evaluation; simply it is comparing the computed rank with the one induced by the relevant judgement. Therefore a reliable RJ has an important role for a precise evaluation. As these evaluation techniques become ubiquitous, the accuracy and meaningfulness of the result is subjective [Sar08]. Due to conflicts in human judgment, research has been conducted to determine how to obtain reliable relevance judgments [KKR09, KKR08]. Here, we recapitulate some of the ubiquitous IR measurements, then we explain the extended version of these metrics that have been adopted in the service discovery domain, and finally describe our evaluation methodologies towards a trustable evaluation measurement.

The existing evaluation approaches are based on *binary relevance*: typically *precision* and *recall*. Binary relevance is limited to identifying whether the result is relevant or not. Recall is the proportion of relevant items that are in the set of returned results, and precision is the proportion of items in the returned results that are relevant [KKR08]. Let r be the set of the relevant items, and q be the set of

returned results. Recall and precision are computed as follows:

$$\text{Recall} = \frac{(q \cap r)}{r}$$
$$\text{Precision} = \frac{(q \cap r)}{q}$$

The average of recall/precision is ubiquitous when comparing the retrieval performance of systems. These coarse grained evaluation methods have been improved by various researchers. Tsetsos et al. [TAH06] rectify these with a fuzzy linguistic variable where matchmaking is the degree of match, and it is specified by a set of fuzzy terms such as *irrelevant, slightly relevant, somewhat relevant, relevant*, and *very relevant*. Jervlin et al. [JK02] proposed a *cumulative gain* (CG), at rank i, CG_i, to generalize these binary metrics to support the graded relevance. It measures the gain g the user receives by scanning the top i items of ranked list j. Gain g is defined depending on the application. Specifically, CG_i is defined as:

$$CG_i = \sum_{j=1}^{i} g(r_j)$$

As we show in the following section, we calculate the gain in our setting based on real quality values rather than on RJ as in typical IR.

7.2 Evaluation Setup

To evaluate our algorithm we need a high quality web service application. Gathering weather data from a collection of weather stations matching the location of real-world telescopes has several issues. First, telescopes tend to be located at a few high-altitude sites with exceptional visibility—this makes for a small number of weather stations; second there are large gaps where the automated weather observation station (AWOS) does not record the weather conditions because they are unserviceable. This results in a dataset that is too small to be used for validating our algorithms effectively.

A better strategy is to use data for stock tracked by the S&P 500 index. In particular, we used stock tracking data for the period from January 1 2009 to November 22 2011—almost two years' worth. Stock can be considered as "services to your money." This dataset is larger in size, and has no gaps in reporting. The data collected includes the low, high, and closing price for the stock, as well as transaction volumes

for each trading day; we use these as proxies for dynamic non-functional telescope attributes . We also tracked the Dow Jones Industrial Average (DJI) over the same period, which we use as an additional attribute when evaluating stock quality.[1]

Our evaluation targets the use-case of a consumer searching for a web service that provides functionality needed in some application being developed. The individual services that could be chosen are in fact proxied by the various stocks that we tracked. This allows us to compare some 500 services (stocks) by evaluating several attributes varying with high frequency, such as price (P) and trading volume (V), and to compare service selection strategies (stock picking)[2] [PDTM12].

After gathering data, we compute the following additional attributes to be used during evaluation: change in price (ΔP), change in volume (ΔV), and Relative Strength vs. Dow-Jones index (RSD); d and $d-1$ are a specific trading date and its predecessor, while s represents a specific stock in our dataset. Formally, we define

$$\Delta \mathrm{P_s} = \frac{\mathrm{P}_d - \mathrm{P}_{d-1}}{\mathrm{P}_{d-1}}$$

$$\Delta \mathrm{V_s} = \frac{\mathrm{V}_d - \mathrm{V}_{d-1}}{\mathrm{V}_{d-1}}$$

$$\mathrm{RSD_s} = \Delta \mathrm{P}_s - \Delta \mathrm{P_{DJI}}$$

We now demonstrate that our Dynamis approach increases the effectiveness of selection by providing finer filtering and ranking of the services in the registries.

The evaluation task is to rank a list of web services according to their relevance for a given consumer request—identify stocks that could generate profits. Using the data collected, we evaluate the quality of our picks by looking at subsequent profits or losses of the stocks selected, thus avoiding human judgment conflicts and their consequences in the evaluation result (cf. Section 7.1) [KKR09].

Most of the analysis is illustrated using one stock from the technology sector: INTC (Intel Corp.); but we also look at AAPL (Apple Inc.) and GOOG (Google Inc.) when we need to compare performance between stocks. For clarity, the data

[1]The full dataset is available at `https://github.com/Atousa/DYNAMIS`

[2]Note that the analysis should not be considered investment advice.

for a given stock is shown consistently in the same color throughout this section. Comprehensive results for all the stocks tracked are available at github footnote link in the footnote.

Figures 7.1 and 7.2 show significant variability in the trading volume data from one day to the next; reported here as the percentage change with respect to the previous trading day's transaction volume (ΔV). However, as shown in Figures 7.3 and 7.4, by computing the histogram of the data over a period of time some of the information contained therein becomes more readily visible. For example, we can see that the ΔV fluctuation is usually within $\pm 50\%$ of the previous day's volume. We rely on the fact that histograms expose high-level information contained in a dense data set, particularly the various quantiles.

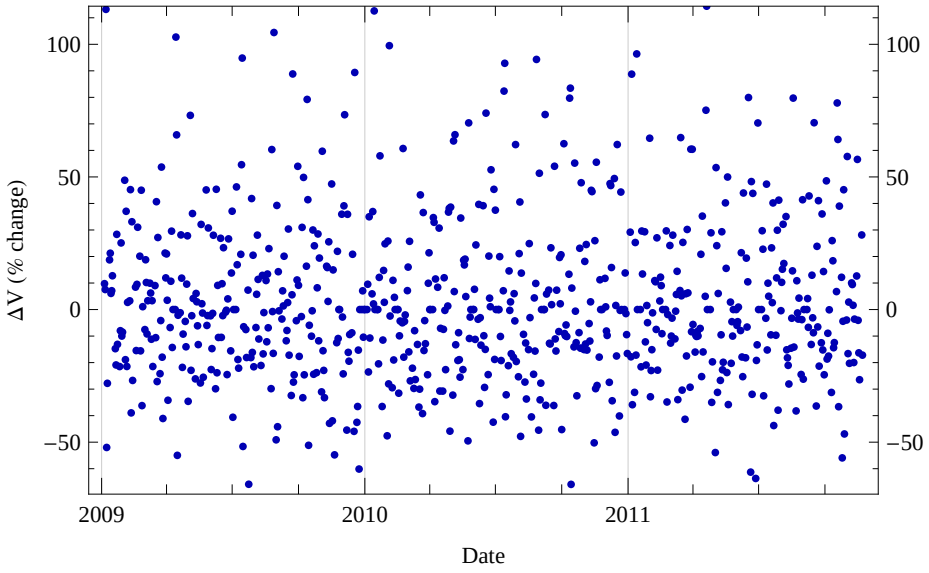


Figure 7.1: INTC transaction volume change with respect to previous trading day.

The area-right-of-threshold (ART) function counts the data in the histogram whose value is greater than the specified threshold—in Figures 7.3 and 7.4 to the right of the dashed line. That is, the ART function counts the number of points in a given quantile.

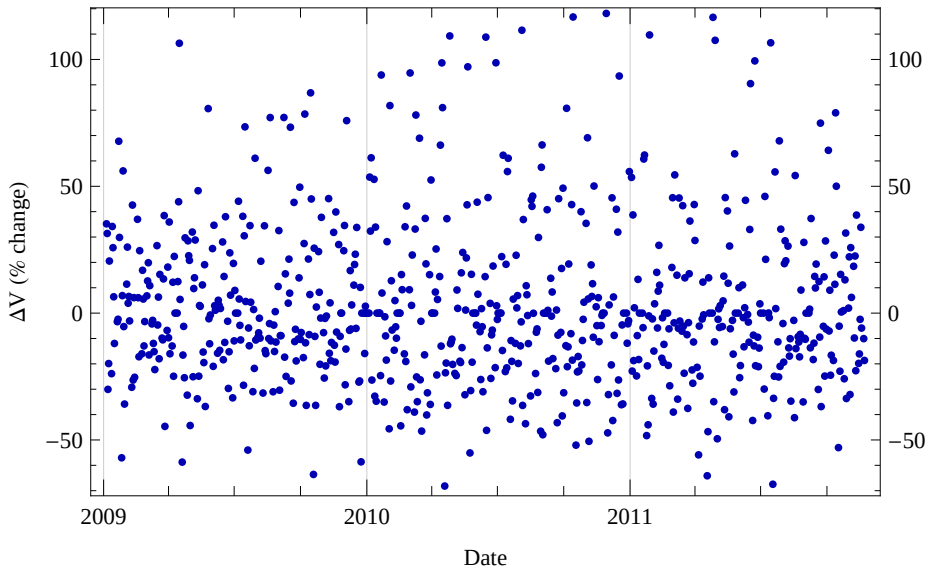


Figure 7.2: GOOG transaction volume change with respect to previous trading day.

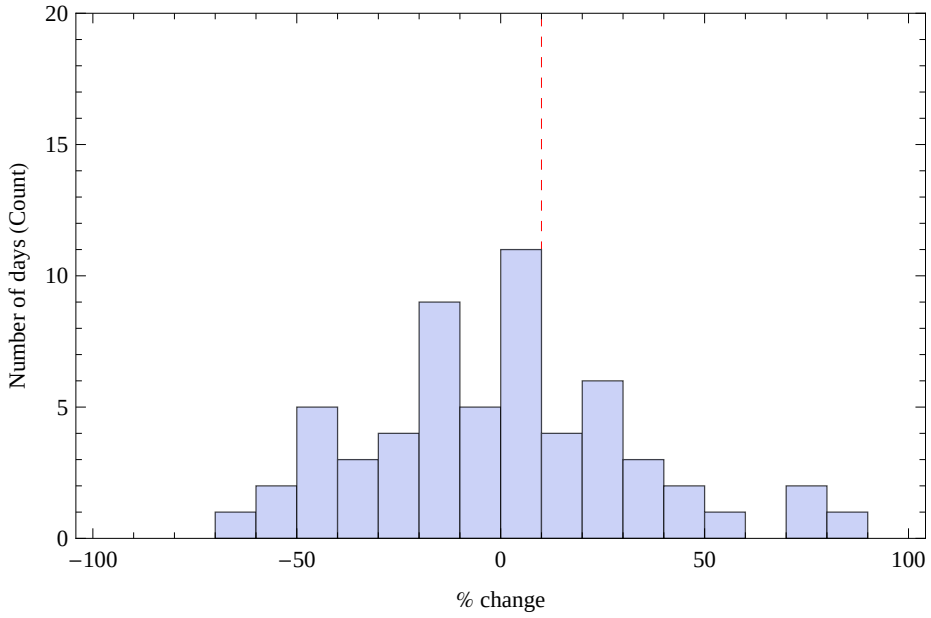


Figure 7.3: INTC histogram of trading volume daily change, starting from Oct 13 2009 over 60 trading days. The dashed vertical line is the 10% threshold. $\mathrm{ART}(\Delta \mathrm{V60}, 10\%) = 19$.

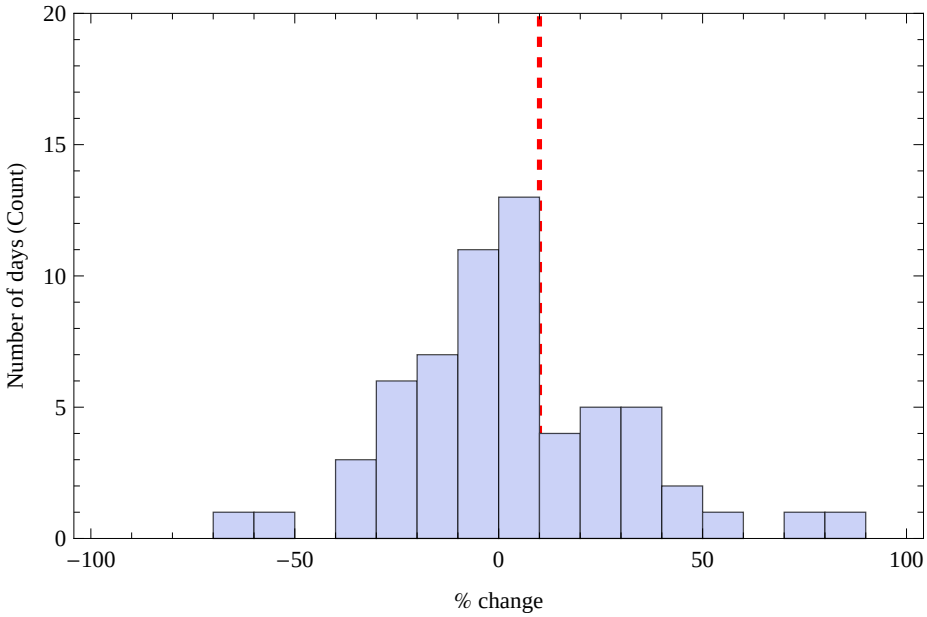


Figure 7.4: GOOG histogram of trading volume daily change, starting from Oct 13 2009 over 60 trading days. The dashed vertical line is the 10% threshold. $\mathrm{ART}(\Delta\mathrm{V60}, 10\%) = 19$.

In Figures 7.5 and 7.6, we show the evolution of $\mathrm{ART}(\Delta\mathrm{V60}, 10\%)$ over time (i.e., for every day we histogram ΔV over the following 60 trading days), then count the number of elements to the right of the 10% threshold. Note that ART() is an integer-valued function: on a given date the daily change is either greater than the threshold or not; respectively contributing the value 1 or 0 towards the value of ART().

Scoring and Ranking

Now that we have collected data and computed various metrics, we aggregate attributes to determine the desirability of a specific service.

The *scoring function* computes a score for each stock: for each day in the snapshot aggregation approach, and for a range of days in the Dynamis aggregation. The scoring function employs a weight vector (user preferences) in the snapshot aggregation approach, and a weight vector as well as a threshold vector in the Dynamis aggregation approach.

Let $\vec{w} = (w_1, \ldots, w_n)$ be the weight vector, and $\vec{\tau} = (\tau_1, \ldots, \tau_n)$ be the threshold vector.

Let $s = \langle a_1, \ldots, a_n \rangle$ be some service. For snapshot aggregation, the scoring func-

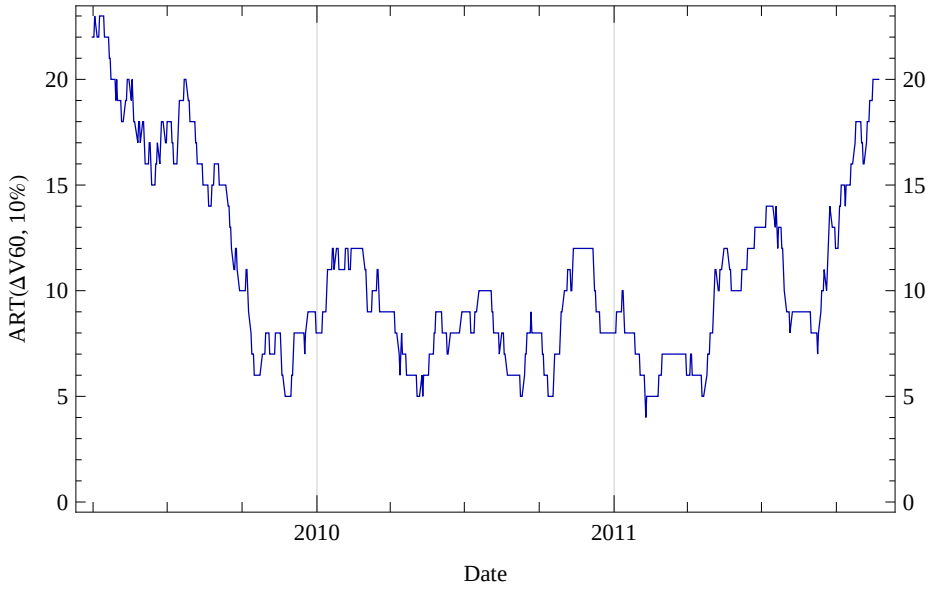


Figure 7.5: Evolution of the ART(ΔV60, 10%) integer-valued function over time for INTC.

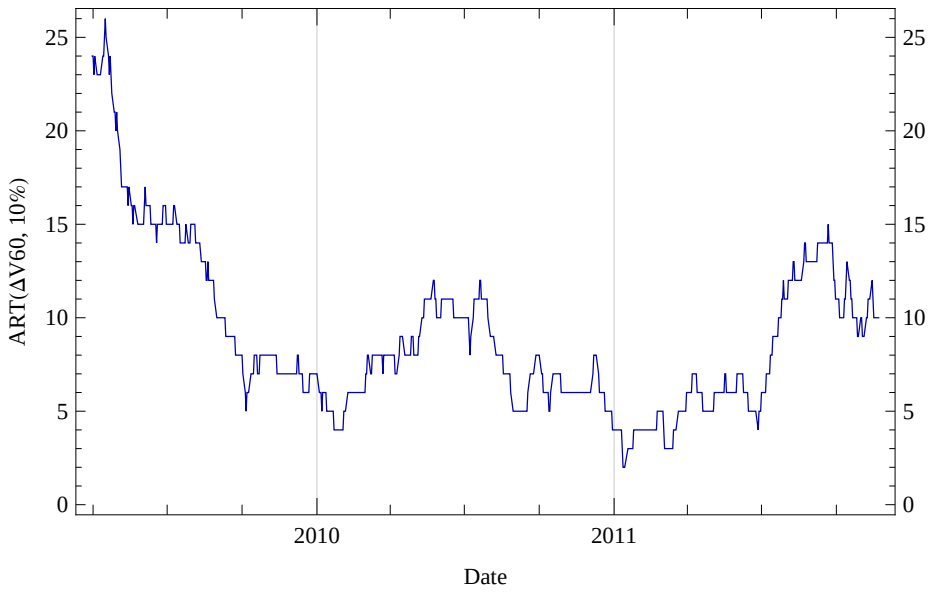


Figure 7.6: Evolution of the ART(ΔV60, 10%) integer-valued function over time for GOOG.

tion ($SAgg$) is:

$$SAgg(s) = \sum_{i=1}^{n} w_i \cdot a_i.$$

For Dynamis aggregation, the scoring function ($DAgg$) is:

$$DAgg(s) = \sum_{i=1}^{n} w_i \cdot ART(a_i, \tau_i).$$

In both the snapshot and Dynamis aggregation, we choose a linear combination of the attributes available to score the services. For our stock setting, we picked

$$\vec{w} = (2.0, 0.1, 3.0).$$

These three weights correspond to ΔP, ΔV, and RSD. The values reflect our opinion that the relative performance of a stock against a benchmark is slightly more important than the daily percentage increase, whereas the volume percentage change is not as important. Nonetheless, we obtain similar results by setting these weights to other reasonable values.

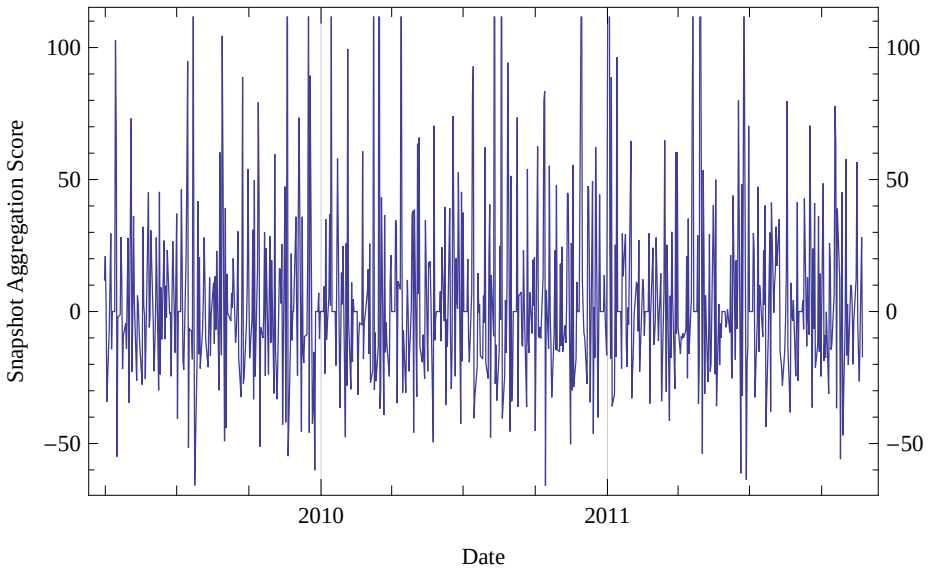


Figure 7.7: INTC score computed daily using snapshotting with scoring function: $SAgg(\text{INTC}) = 3 * \text{RSD} + 2 * \Delta\text{P} + 0.1 * \Delta\text{V}$.

The score computed with ($SAgg$) for INTC as it varies over time is depicted in Figures 7.7 and 7.8. For the Dynamis aggregation approach the resulting scores vary more slowly over time (cf. Figures 7.9 and 7.10).

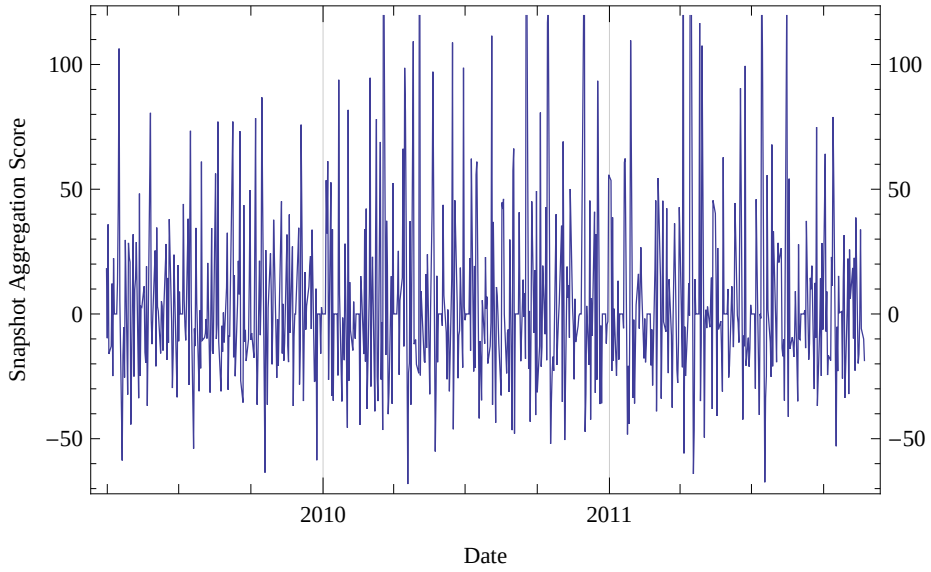


Figure 7.8: GOOG score computed daily using snapshotting with scoring function: $SAgg(\text{GOOG}) = 3 * \text{RSD} + 2 * \Delta\text{P} + 0.1 * \Delta\text{V}$.

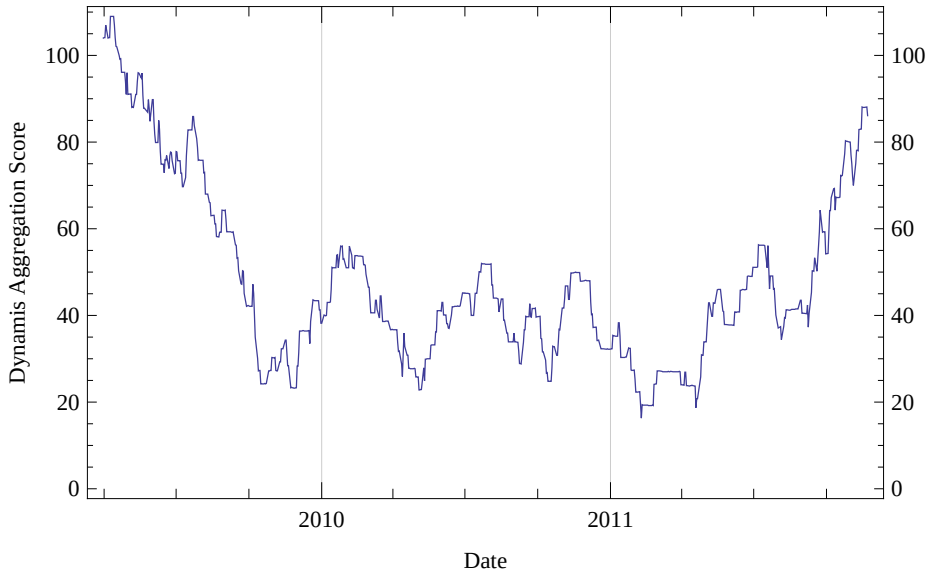


Figure 7.9: INTC score computed daily using Dynamis with the following scoring function: $DAgg(\text{INTC}) = 3 * \text{ART}(\text{RSD60}, 1\%) + 2 * \text{ART}(\Delta\text{P60}, 2\%) + 0.1 * \text{ART}(\Delta\text{V60}, 10\%)$.

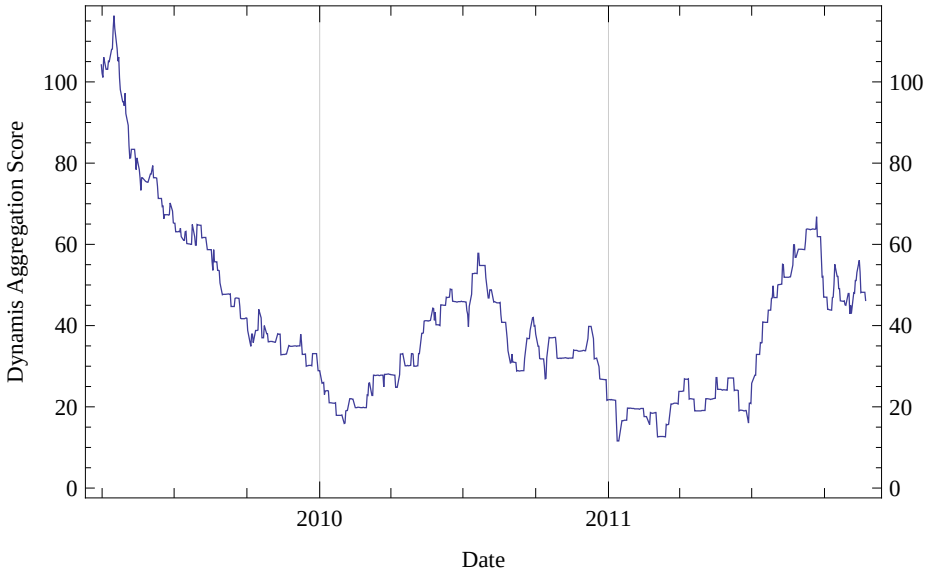


Figure 7.10: GOOG score computed daily using Dynamis with the following scoring function: $DAgg(\text{GOOG}) = 3 * \text{ART}(\text{RSD60}, 1\%) + 2 * \text{ART}(\Delta\text{P60}, 2\%) + 0.1 * \text{ART}(\Delta\text{V60}, 10\%)$.

Table 7.1: Top-10 stocks on October 3, 2011, ranked with Dynamis aggregation. These correspond to the highest-scored tickers in Figure 7.11.

Rank	Ticker	Company
1	CF	CF Industries Holdings, Inc.
2	RRC	Range Resources Corp.
3	RHT	Red Hat, Inc.
4	TLAB	Tellabs, Inc.
5	NVDA	NVIDIA Corporation
6	GT	The Goodyear Tire & Rubber Company
7	JWN	Nordstrom, Inc.
8	SNDK	SanDisk Corporation
9	CTXS	Citrix Systems, Inc.
10	PFG	Principal Financial Group Inc

In our dataset, we have scored every stock, on every day. So far, we have looked at the score of a single stock as it changes over time. We can examine the scores for all the stocks on a specific day, sorting from best to worst. The position of a

stock in this sorted list is its *rank* for that day; therefore, a high score results in a low rank—rank 1 is the best stock pick. On a specific date, say October 3, 2011, the curve depicted in Figure 7.11 represents the sorted, and Table 7.1 lists the top-10 ranked stock tickers for that day. These present the ranking of tickers on a specific date, but the evolution of a given ticker's rank over time can also be graphed.

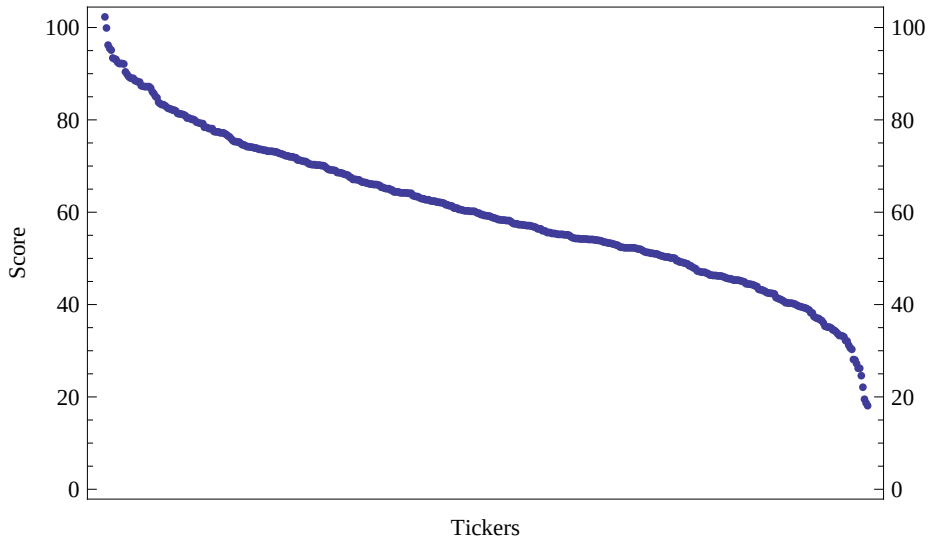


Figure 7.11: Sorted scores computed with Dynamis aggregation on October 3, 2011.

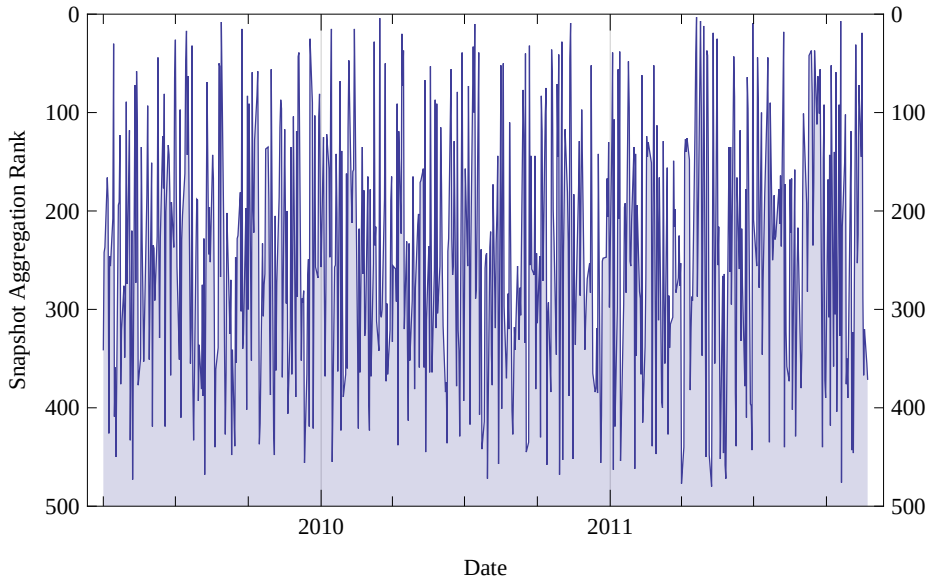


Figure 7.12: INTC rank as its score (snapshot aggregation) changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means better stock pick.

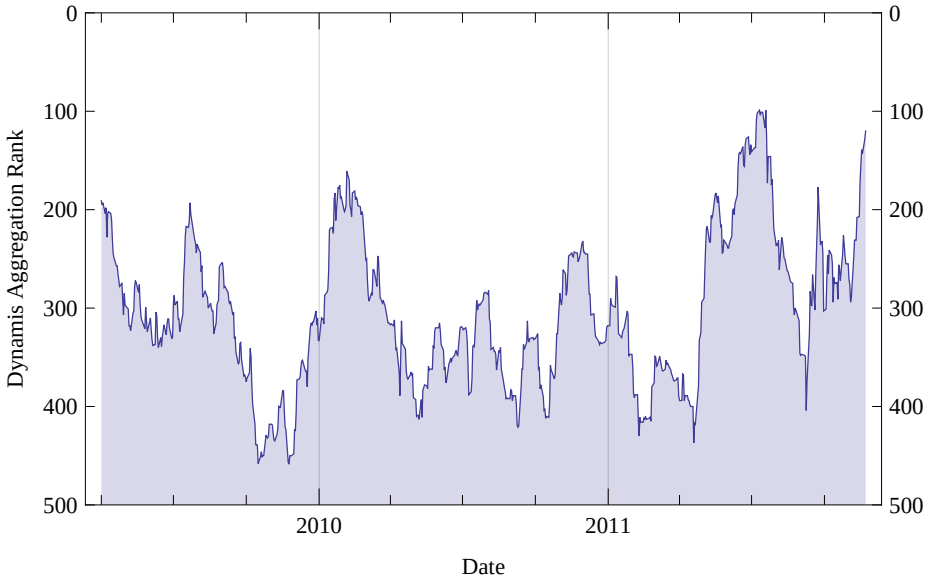


Figure 7.13: INTC rank as its score (Dynamis aggregation) changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

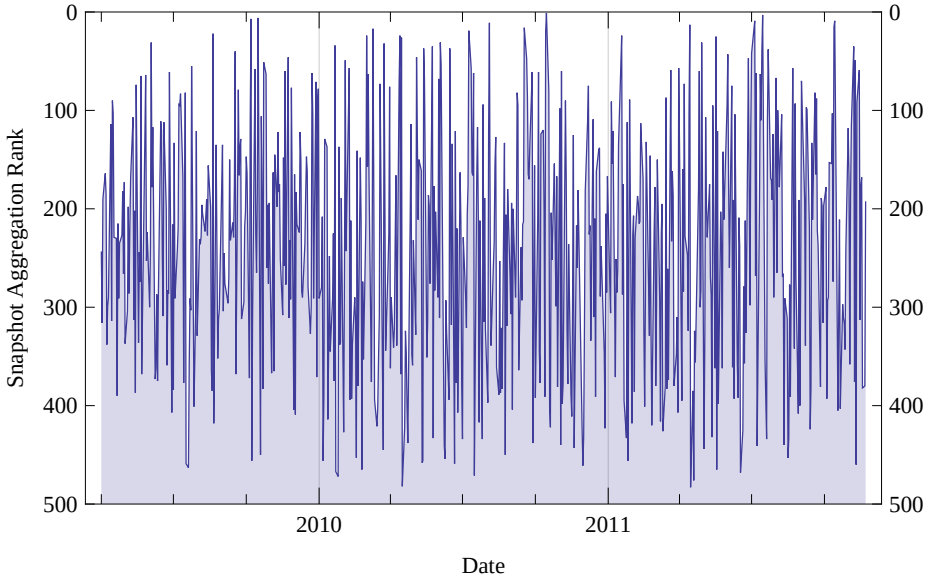


Figure 7.14: GOOG rank as its score (snapshot aggregation) changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means better stock pick.

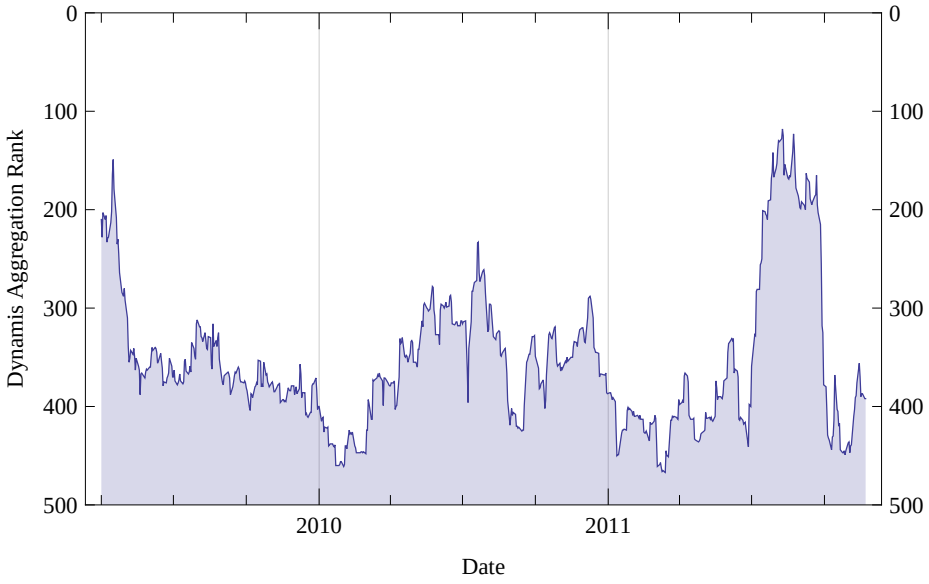


Figure 7.15: GOOG rank as its score (Dynamis aggregation) changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

Figures 7.1, and 7.5–7.10 are time-based representations of the data. Figure 7.11 is, instead, a look across all our tickers on a fixed date. The following figures are again plotted against time. In the case of INTC and GOOG, the results based on the snapshot and Dynamis aggregation techniques are shown in Figures 7.12, 7.13, 7.14, and 7.15, respectively. Note that for clarity, when graphing ranks we shade the area under the curve; this allows to visually distinguish ranks from other quantities graphed.

Skyline

Since the result of the skyline can be too big and furthermore not ranked, we employ the notion of k-dominance introduced by Huang et al. [HDWX08]. Thus, we compute a score for each service as follows. Suppose $\langle a_1, \ldots, a_n \rangle$ are the attributes of the service s. We now consider the sorted lists of values for each of these attributes considering the results of the skyline. Let $s = \langle a_1, \ldots, a_i, \ldots, a_n \rangle$ be a service in the result where $i \in [1, n]$. For each value a_i we check to see what its rank is in the sorted list. By r_i we denote the rank of a_i in the list. Let $r_s = \langle r_1, \ldots, r_n \rangle$ be the vector of the ranks

for s. Then, the scoring function simply sums these ranks.

$$SSkyline(s) = \sum_{i=1}^{n} r_i.$$

Finally, we retrieve the top-k services from the skyline according to this scoring function.

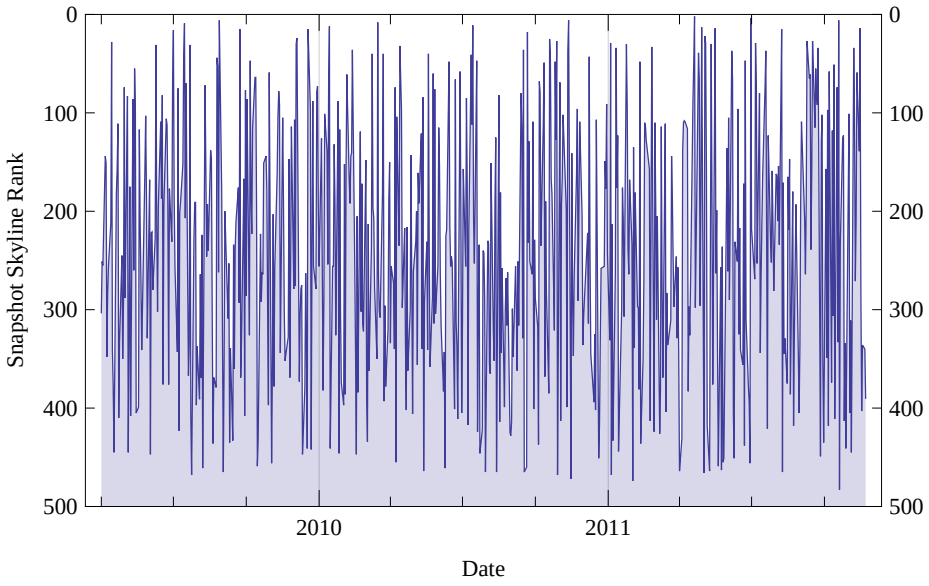


Figure 7.16: INTC snapshot skyline rank as its score changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

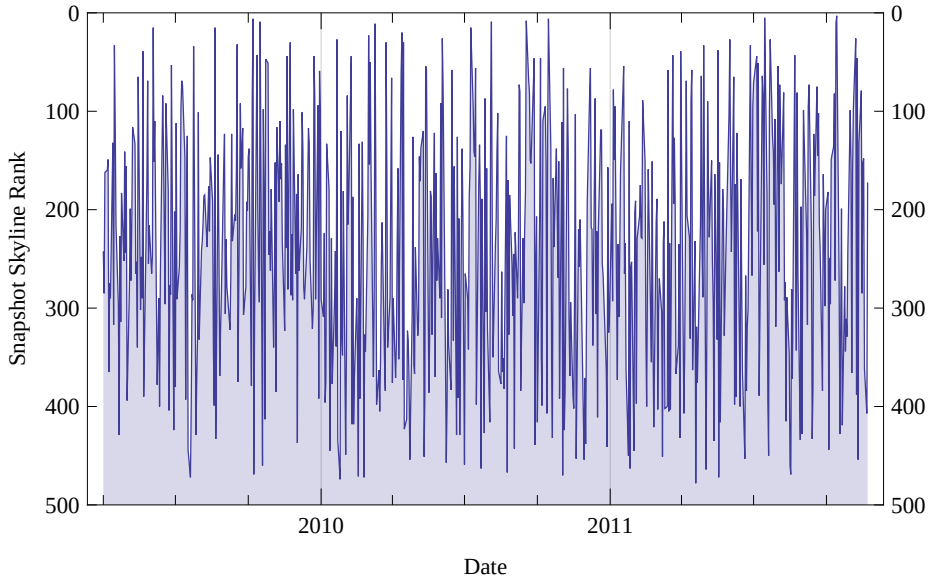


Figure 7.17: GOOG snapshot skyline rank as its score changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

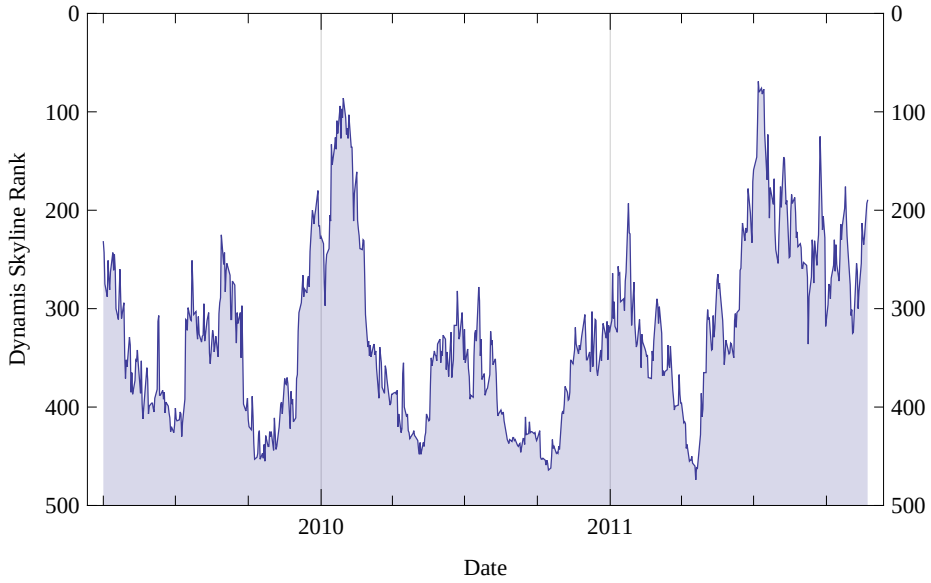


Figure 7.18: INTC Dynamis skyline rank as its score changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

The daily evolution of the snapshot skyline rank for INTC and GOOG are depicted in Figures 7.16 and 7.17, while that of the Dynamis skyline rank is presented in Figures 7.18 and 7.19.

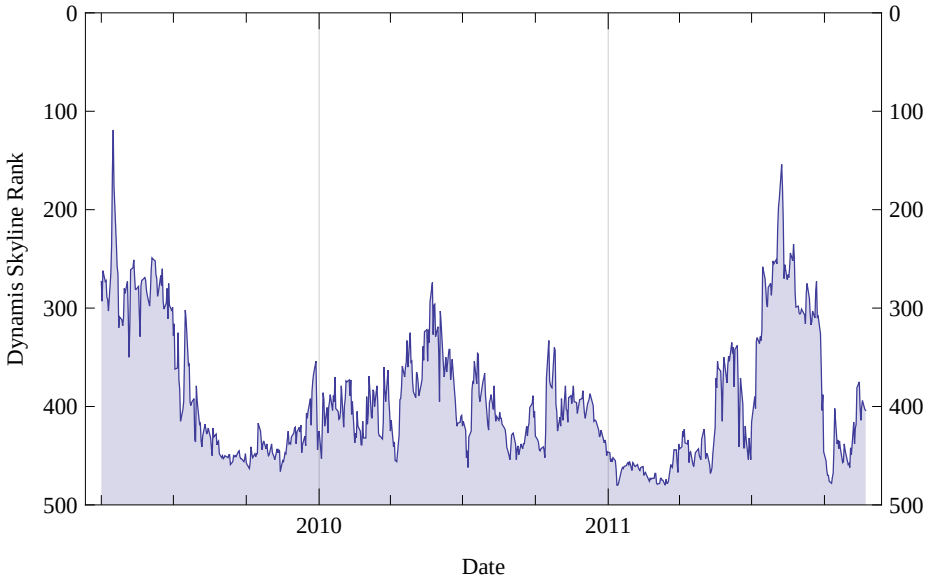


Figure 7.19: GOOG Dynamis skyline rank as its score changes over time in a pool of 500 tickers. Lower-numbered rank (visually higher in the graph) means higher score.

7.2.1 Results

In order to compare and evaluate the rankings computed, we looked at three tickers from the technology sector: AAPL, INTC and GOOG. Figure 7.20 shows the relative price change of these stocks over the entire time period, normalized to start from the same relative value and overlaid on the same figure. This helps visualize the performance of the three stocks relative to each other.

Figure 7.21 shows the computed ranks using snapshot aggregation for our three stocks. The data is noisy enough that we chose not to connect the points, otherwise we would have been faced with a solid block of color. Much like every graph seen so far, snapshotting gives us noisy data: the rank for a given stock will vary significantly from one day to the next.

This may or may not be a disadvantage; there may be a fixed cost to switching services (e.g., a transaction fee), which could overwhelm the benefits of frequent change. But there may be an advantage to changing the selected service frequently, since switching providers load balances the services nicely. Therefore, the high frequency of change in ranks in the snapshotting technique cannot, in and of itself, be counted against the technique. However, in Figure 7.22, by computing the histogram of the data over a period of time, it becomes more readily visible.

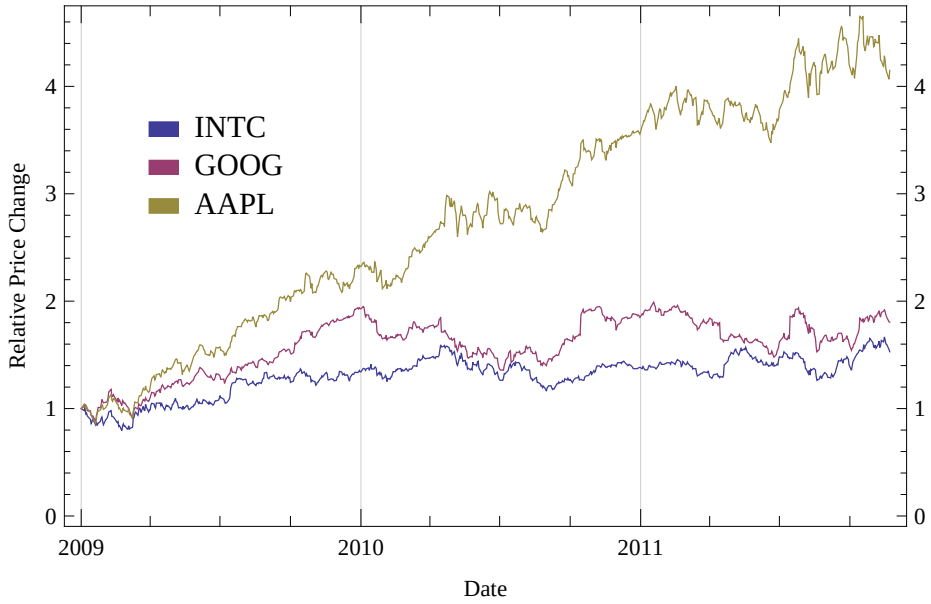


Figure 7.20: Relative price change of 3 stock tickers over the time period considered, normalized to start from the same relative value.

The snapshot skyline ranks are not shown, but they looked visually much like Figure 7.21 in that they are scattered with no obvious trend. Figure 7.23 instead shows the ranks computed using the Dynamis skyline techniques.

On a specific day d we select the top-k stocks from using the four methods we described. Specifically, we obtain the following top-k lists day's rankings:

- L-SAgg using *SAgg*
- L-SSkyline using *SSkyline*
- L-DAgg using *DAgg*
- L-DSkyline using *DSkyline*

The top-k ranks are different between the algorithms.

$$\begin{cases} L-SAgg = (s_1, s_2, \cdots, s_k) \\ L-SSkyline = (s'_1, s'_2, \cdots, s'_k) \\ L-DAgg = (s''_1, s''_2, \cdots, s''_k) \\ L-DSkyline = (s'''_1, s'''_2, \cdots, s'''_k) \end{cases}$$

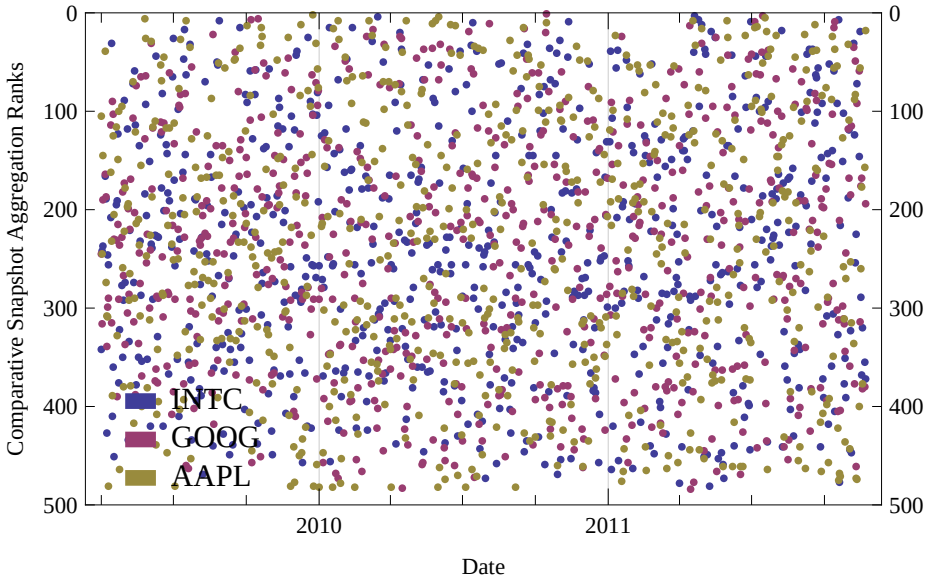


Figure 7.21: Snapshot aggregation rank of our 3 stocks over the time period considered.

In order to evaluate the success of these four methods, we now evaluate the cumulative gain obtained from purchasing the top-k ranked stocks.

Using the historical data gathered for stock prices, we are able to follow the evolution of a hypothetical investment on day d for each of our top-k picks; the period over which we follow that investment we call *window* w. We can compute the gain for stock s, purchased on day d over the investment period w as:

$$\text{Gain}_d^s = \frac{\text{P}_{d+w}^s - \text{P}_d^s}{\text{P}_d^s}$$

We divide the gain by the initial price in order for our computations to remain independent of the magnitude of the price. This quantity is known as the arithmetic return or yield.

We assume a sale on the last day of the window, when the gains or losses are realized. At that time, we add the gains for each one of the stocks selected as part of the top-k pick. In order to keep our results independent of the number of stocks selected, we divide the total gain at the end of the investment period by k. The cumulative gain (CG) is therefore:

$$CG_d = \frac{1}{k} \sum_{s=s_1 \ldots s_k} \text{Gain}_d^s$$

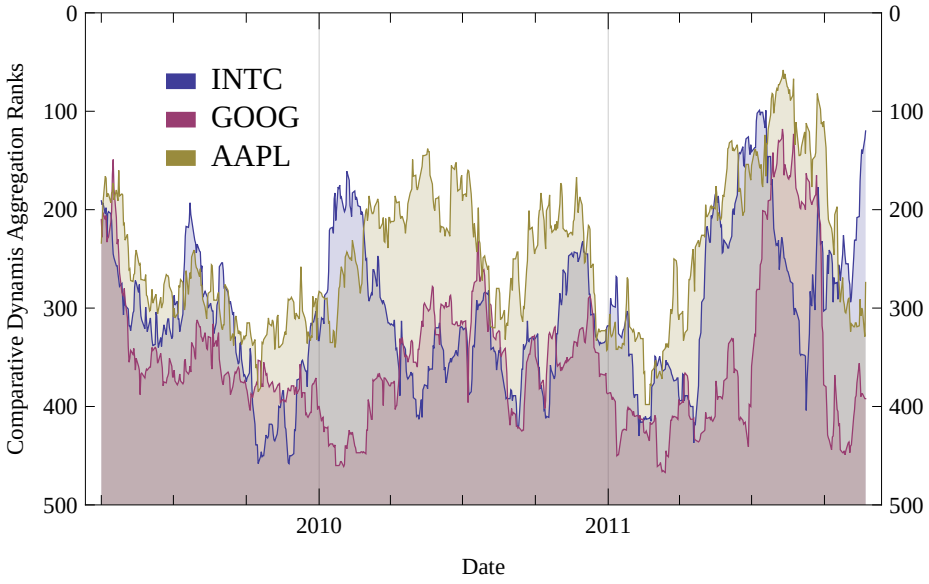


Figure 7.22: Dynamis aggregation rank of our 3 stocks over the time period considered.

This quantity is technically known as the arithmetic average rate of return. Thus, a result greater than 0 represents a net gain and a result less than 0 represents a net loss over the period. To illustrate this, assume we pick the top-10 stocks ($k = 10$) and track them over the course of one month, i.e., 20 trading days ($w = 20$). This is illustrated in Figure 7.24, using the highest ranked stocks selected by the snapshot aggregation technique. Figure 7.25 shows these same gains depicted in a histogram. The mean gain is a modest 1.7%.

Since each day is an independent investment, we can show the evolution of the portfolio over time by compounding the cumulative gains. It is critical to realize that the CG_d only affects that fraction of the overall investment that was invested on that day, that is $1/w$. Let us explain this fact with an example: we shall invest a fixed amount of money x in each of the top-k stocks selected by one of our algorithms, and keep the funds invested for w trading days. At the end of the period, each stock has a rate of return Gain_d^s, and the portfolio of k stocks has a cumulative gain $CG_d = 1/k \sum_s \text{Gain}_d^s$. That is, we invested $x \times k$ on day d and recovered $x \times k \times CG_d = x \times \sum_s \text{Gain}_d^s$ on day $d + w$.

Now assume that we repeat this process every day, over a long period of time. Since we are investing a fixed amount per day, the total amount of money that needs to be available at the begining of the process is $\text{Funds} = x \times k \times w$. The cumulative gain CG_d on any given day only affects $1/w$ of total sum invested. A little algebra

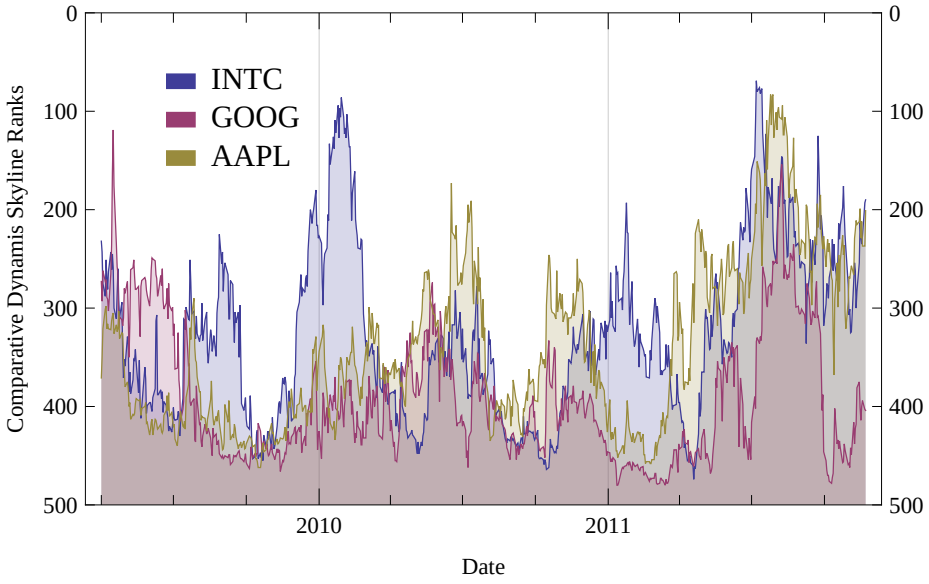


Figure 7.23: Dynamis skyline rank of our 3 stocks over the time period considered.

will show that:

$$\frac{1}{w} \times \text{Funds} \times CG_d = \text{Funds} \times \frac{1}{k} \sum_s \frac{\text{Price}^s_{d+w} - \text{Price}^s_d}{w \times \text{Price}^s_d}$$

That is, the price difference is scaled by $1/w$. To compute the evolution of the overall gain (OG) over time, we compound the cumulative gains as follows:

$$OG = \prod_d \left(1 + \frac{CG_d}{w}\right)$$

The gain Gain^s_d and cumulative gain CG_d are both zero-based quantities, compounding these would converge to zero very quickly. Adding 1 to the quantity CG_d/w allows us to multiply them meaningfully. This is the same as saying that a 20% yield is equivalent to a factor of 1.2.

Figure 7.26, shows our four algorithms side by side. We observe that Dynamis aggregation performs best. Snapshot aggregation and snapshot skyline performed almost identically, they are indistinguishable in Figure 7.26; but they are in fact not identical.

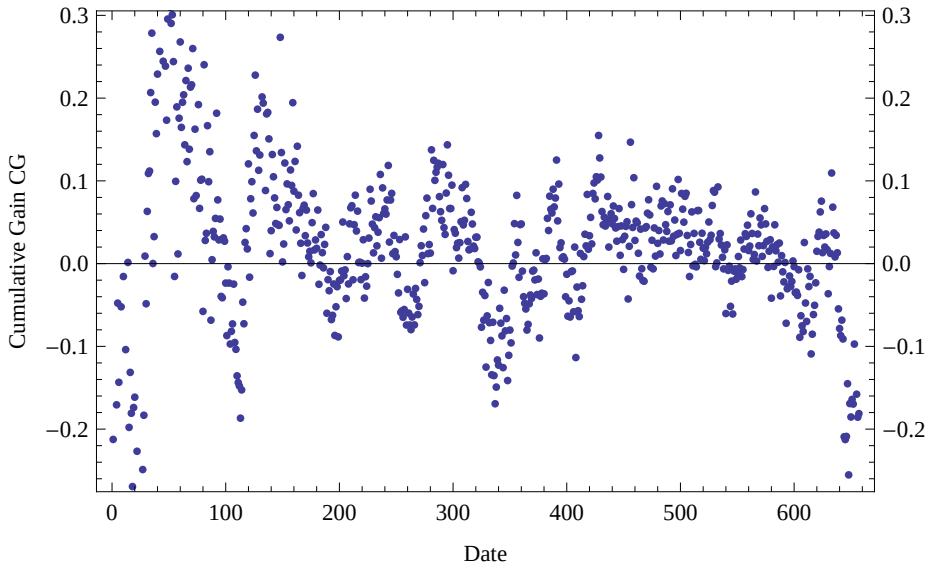


Figure 7.24: Gain realized by picking the top-10 tickers chosen from the snapshot aggregation algorithm and selling after 20 days.

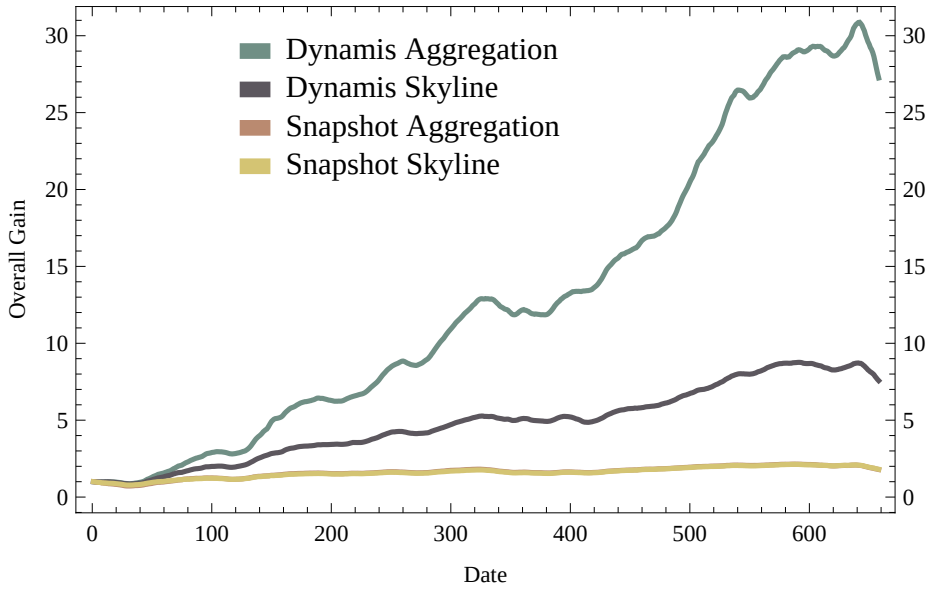


Figure 7.26: Overall gain (OG) for our four algorithms with k=10 and w=20.

Finally, we compute the geometric average rate of return, or average yield, as:

$$Y = \sqrt[n]{\prod_{d=1}^{n}\left(1+\frac{CG_d}{w}\right)} - 1 = \sqrt[n]{OG} - 1$$

Comparing the average yield Y at the end of the evaluation period in Figure 7.27,

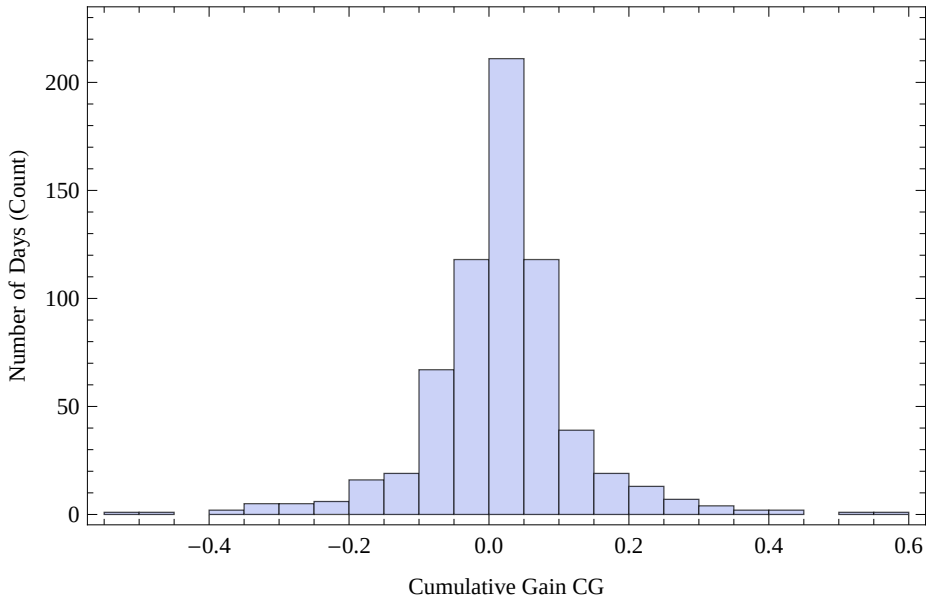


Figure 7.25: Histogram of the gains realized by picking the top-10 tickers chosen from the snapshot aggregation algorithm and selling after 20 days.

we see that Dynamis performs better than snapshotting, with Dynamis aggregation being the clear winner.

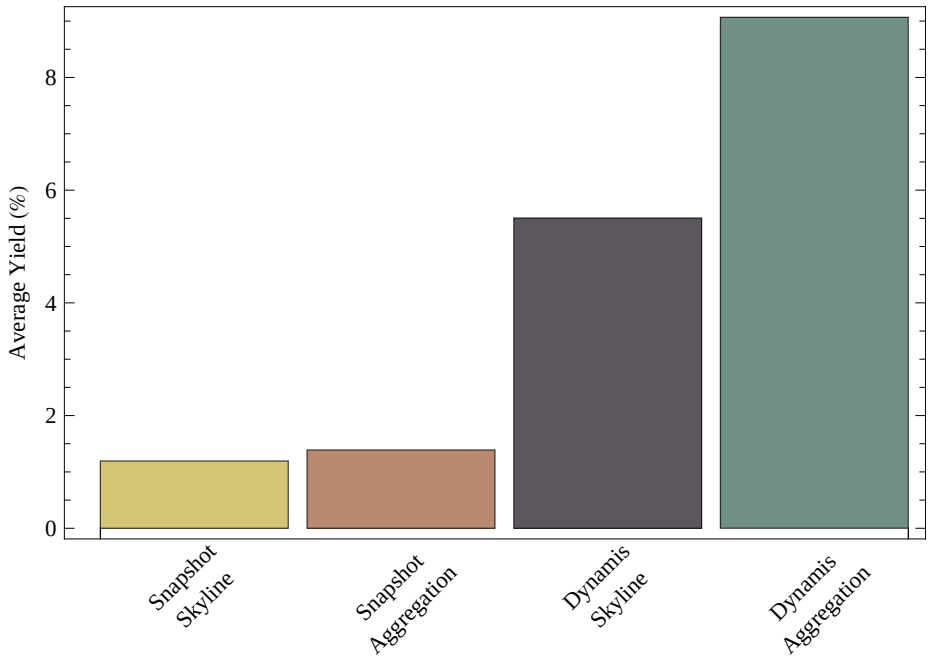


Figure 7.27: Average yield at the end of the evaluation period for our four algorithms with k=10 and w=20.

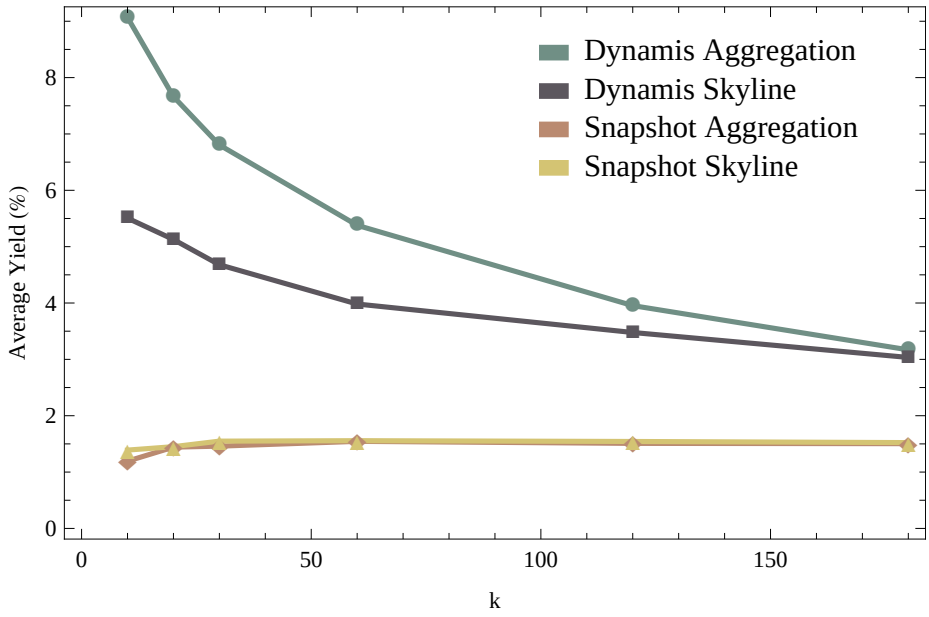


Figure 7.28: Average yield (Y) for a 20 trading days window ($w = 20$) as we pick an increasing number of top-k picks along the X axis.

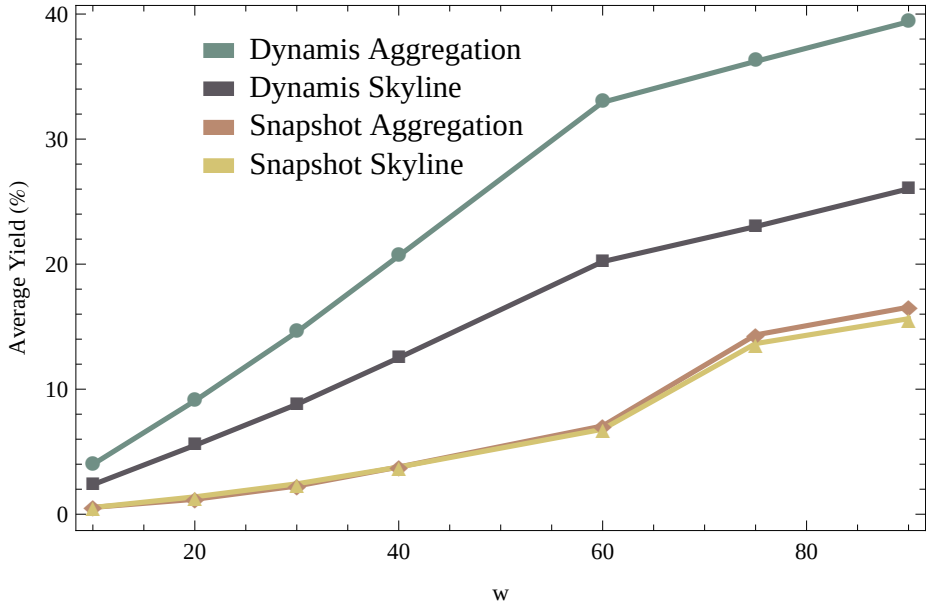


Figure 7.29: Average yield (Y) for the top-10 picks as we pick an increasingly large window w along the X axis.

7.2.2 Generalization of Results

The previous section established that Dynamis performs better than snapshotting, and Dynamis aggregation performs better than Dynamis skyline. We now show that these results holds for other values of k and w. Figure 7.28 looks at the average yield Y for a 20-trading day window ($w = 20$) as we pick an increasing number of top-k picks along the X axis. The actual gain does not matter, the intent is to show that the relative performance of the algorithms remains the same regardless of the value of k. While the skyline and aggregation implementation do converge, Dynamis remains superior compared to the snapshotting technique. As k increases, we select more stocks and performance trends towards the S&P 500 market index.

Similarly, for a fixed value of $k = 10$ the relative performance of the algorithms remains the same as we vary w; this is shown in Figure 7.29. While the yields may look surprisingly high, one must consider the fact that the overall market did well in the period under consideration: both the Dow Jones and the S&P 500 indices trended upwards. As a result, picking the top-10 performing stocks everyday and holding these for a period of several weeks is expected to provide good returns. Consider however that the amounts invested are substantial as w increases, and the risk is high. Thus we have shown that the relative performance of our algorithms remains the same as k and w vary, with Dynamis outperforming snapshotting in all cases investigated.

7.3 Discussion

The use of stock market data was chosen as a proxy for dynamic non-functional attributes, primarily due to the paucity of actual data on telescopes; it represents a large data source of high-frequency information. More importantly, the stock market allows us to evaluate the performance of our algorithms by comparing the performance of the stocks picked with respect to the market as a whole. This is something that cannot be done with weather data, because we have no way to evaluate the quality of a picture that was not taken by a telescope. The ability to compare our algorithms with an unbiased metric over which we have no control allows us to perform a quantitative performance analysis that is not possible in a subjective context.

7.4 Summary

In Chapter 6, we discussed the Dynamis algorithms to address the existing challenges in web service discovery such as avoiding stale dynamic attributes. We accentuated the need for reliable non-null values for each QoS attribute in order to improve the QoE. Dynamis enables such improvement for selecting web services using well-established techniques, such as skyline and aggregation. We evaluated our algorithms using stock market services, and performed a qualitative performance analysis with respect to the market as a whole. This method of analysis is not possible in a subjective context where the evaluation mechanisms are employed in IR systems. Our evaluation shows significant improvement in service selection, outperforming existing techniques in all cases investigated.

Chapter 8

Security: Web Service Assurance

8.1 Overview

The goal of software assurance is to provide to software purchasers and users the justifiable confidence that the software will consistently exhibit some desired functional and non-functional properties, including *security* QoS attributes (cf. Section 1.6). The expanding portfolio of web services available over the global network has the welcome effect of enabling on-the-fly service selection, but at the cost of increased security and reliability risks with respect to ordinary components—off-the-shelf or developed in-house. For this reason, a number of assurance tools have been proposed, aimed at monitoring the delivery of web services and remotely diagnosing and resolving web service issues. Generally speaking, assurance tools consolidate and correlate the measurements of some aspects of the QoE perceived by a web service user. Such measurements are then stored in a database for further processing, including data analysis and visualization. Today, most commercial web service engines, a.k.a. *containers*, such as WSO2[1], IBM webSphere[2], OpenESB[3], Apache AXIS2[4], and Glassfish Metro[5] support web service assurance in some way.

Security assurance, that is, assurance aimed at monitoring security attributes, is not as well established as the performance, dependability and transaction support subjects. Well-known web security standards, such as the WS-* stack [GHH+02], implement secure transaction, conversation, and access control systems. However, these

[1] `http://wso2.com`
[2] `http://www.ibm.com/software/websphere`
[3] `http://java.net/projects/openesb`
[4] `http://axis.apache.org/axis2`
[5] `http://metro.java.net`

standards do not include any provisions for web service assurance; rather, they are aimed at satisfying pre-defined security requirements that service designers identify at development time. Some well-known packages for service security, such as Apache Rampart[6] and Java WSIT[7], provide basic facilities for monitoring web service security mechanisms; however, few if any assurance techniques based on such facilities have been proposed in the literature.

This chapter explores the idea that machine-readable certificates of security and privacy properties of web services can provide a way to bound, transfer, and alleviate security risks at run time. In 2008, the Software Engineering Institute (SEI) published a requirements document on the service certification process for the U.S. Army, Aos Chief Information Office/G-6 (CIO/G-6) organization to address security and provisioning concerns the Army foresees in its development of Service-Oriented Architecture (SOA) environments [GHH+11]. This document presents a methodology for certifying web services to assure their compliance with stated non-functional requirements. This work represents a first step in defining methods and techniques to certify that the service satisfies specific security requirements, producing security assertions and documents that could be accepted as a proof of service assurance. It is important to note that certification of web service security attributes involves both the container and the code implementing the service itself. Even if a web service container is certified to support, say, the confidentiality of data flowing through a service interface, no inference can be done on the internal implementation of the service, where information leaks could happen.

Damiani et al. introduced container- and service-level certification of security attributes of services [DM09]. Their work aims to support the definition and runtime handling of certificates stating that a set of security attributes are satisfied by web services deployed in a given container. However, handling context changes—involving the container, the underlying hardware, or the service itself—remains an open issue.

8.1.1 Research Objectives and Contributions

Deploying software applications over the Internet as web services and dynamically discovering and composing them for a specific purpose demands a service quality

[6] `http://axis.apache.org/axis2/java/rampart/`
[7] `http://wsit.java.net`

assessment mechanism. Run-time assessment is increasingly important due to the proliferation of software-as-a-service (SaaS), the adoption of service-oriented architectures (SOA), and the increased use of wireless and mobile technologies. These trends point to large-scale heterogeneous infrastructures hosting applications that are dynamically built from loosely-coupled and orthogonal services. In such scenarios, assessing security attributes such as integrity and confidentiality is of critical importance for evaluating services made available online. These security attributes should be used as constraints over service selection and as part of collaboration contracts, particularly service-level agreements (SLA). Unfortunately, the effort of modelling, measuring, monitoring, and assessing these properties is often perceived as superfluous due to the contextual nature of their interpretation. Services returned from discovery mechanisms that ignore these aspects are often unreliable. That is a major problem that devalues the search results.

While there are standards and protocols available, a capability gap remains in ensuring that a particular service satisfies, at execution time, specific criteria. When selecting services available on the web, consumers mostly rely on security information derived from a pre-deployed service evaluation—information that is potentially stale. SLAs could cover a wider range of issues, including security and reliability [RK09]. The creation and management of SLAs is a substantial effort, since it is a contract legally binding the provider to deliver a technical performance. Current techniques for SLA development introduce significant overhead in terms of negotiations [CMJ+08].

We develop an architecture that enables service consumers to specify the desired set of security attributes services must satisfy, thereby providing a mechanism to recertify them at run-time when the context changes [O'S06] [RK09]. One goal of this research is to define a machine-readable representation of security and reliability to implement assurance level agreements (ALA), which could be part of existing SLAs, thus preserving backward compatibility with existing standards. We explore intelligent and efficient service selection techniques based on ALAs. Finally, we develop efficient analysis algorithms enabling run-time enforcement of ALA-related policies. One expected outcome of this research is to further integrate SLAs into the service life-cycle and provide the basis for some preliminary standardization work.

In this chapter, we: (1) Investigate models to represent security and reliability requirements, enlarge the scope of SLAs to new categories of security attributes named ALAs, and provide efficient tools for security attribute-driven service selection. Having a unified vocabulary for discussing and negotiating security attributes is essential

for the benefit of both the service provider and the service consumer [O'S06] [RK09]. (2) Specify high-level system objectives and detailed security attributes of services and service compositions. These objectives would guide the discovery process based on specified objectives first, properties second, in order to increase the relevancy of the results. (3) Investigate a complete framework to manage the creation, editing, validation, publication, and discovery of ALA documents along with their service descriptions, thus facilitating automatic negotiation of security attributes during the discovery process. (4) Study a selection mechanism where the security objectives of the contracted services are updated based on dynamically changing context.

This chapter is organized as follows: Section 8.2 provides background information on existing standards and security mechanisms. Section 8.3 presents related work. Section 8.4 is an introduction on how to express security requirements. Section 8.5 discusses our approach to security certification of services and presents different types of certificates. Section 8.6 describes the concept of adaptable security certification using a practical example and introduce a conceptual architecture for adaptable assurance-based service discovery.

8.2 Background

Web services, whether stand-alone or composed, must provide a solid mechanism to guarantee the security of their basic components and interactions in heterogeneous dynamic environments. Preventing the misuse and malicious behaviour of web services that are deployed in SOA and communication through ad hoc connections has always been critical. Threat modelling is a conventional methodology to specify, determine, and document the possible *threats*, *vulnerabilities*, and *attacks* against a software component's security as well as any *countermeasure* that could be employed [BMPS09]. Herein, threat denotes any event with the potential to impact organizational operations through unauthorized access [Kis11]. Vulnerability is defined as any flaw in a software system that can be triggered by an adversary. A software vulnerability could include flaws introduced at design time, inappropriate coding, configuration errors, and access control misconfiguration [Com10]. Attack refers to any intrusion in a software system that exploits any software vulnerability. Finally, countermeasure is an organization's actions to alleviate the risk of attack. Threat modelling has not yet been applied to web services due to the lack of a standardized mechanism. W3C proposed the development of a threat model as part of the web service

framework [ABFG04].

We briefly introduce some of the relevant methods and standards that have been formulated and can be employed to assure web service security.

8.2.1 Towards Threat Modelling for Web Services

Threat modelling was initially used by Microsoft [MMW05] to mitigate the security risks during design and operation. Eventually, it became an integral part of Microsoft's Trustworthy Computing Security Development Lifecycle (SDL) [LH05]. With respect to web services, threat modelling could be applied to the entire web service stack such as the SOAP processor, XML parser, and web application. In this vein, threat and vulnerability in web applications and their infrastructure could be identified and categorized (e.g., tampering or input validation). Then at design time the aforementioned categories could be used to check which and where security functions need to be implemented (e.g., session management, at application). Another example of the vulnerability category and its corresponding security function is *authentication* and *encryption*, respectively [MMD+03]. In fact, not all threats can be identified during the design phase, and threat modelling continues during the remainder of the application life cycle. The output of the process is a catalog of the threats, vulnerabilities, and attacks on web applications and their architectures that are effective during both the design and operation phases [BMPS09].

More precisely, the catalogues are built through the following major steps: identification of the valuable assets that can be accessible through web application (e.g., resources) and the business assets that could be compromised as a result of an attack (e.g., company reputation), definition of security objectives to determine the level of the security attributes required for the identified assets, studying the web application and its architecture, creating a security profile to represent the approaches chosen for each security function to prevent a given vulnerability category, and finally creating the threat catalogue. To discover possible threats that are implemented and used in the security functions of the application, a team including application architects, developers, testers, and security experts are recommended by Microsoft. The ultimate output of the process is a threat and attack profile that represents the likely path of attacks and their category. For clarity, below we introduce Microsoft STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of service, Elevation of privilege) categories that explain common threats [Mic05].

- Spoofing: Illegal access to a user's identification
- Tampering: Unauthorized alteration of information
- Repudiation: Disproved actions of users
- Information disclosure: Divulgence of private data
- Denial of service: Malicious action of making a service unavailable
- Elevation of privilege: Misused privileged user identity to have extended access to applications

In order to find and represent other threats, *attack trees* [Sch99] and *attack patterns* [CAP12] are designed. Attack trees embody possible paths followed by an attack as a tree, where the root node is the objective of attackers and the children and the leaf nodes are the refinements of the node. To complete an attack, the tree might contain thousands of paths. Attack trees could be very complicated and require substantial security expertise and effort. Attack patterns derive from the concept of design patterns [Wol94] and are based on the analysis, utilization, and communication of discovered attacks on software. Attack patterns contain the information about attacks along with their schema and classification taxonomy. Common attack pattern enumeration and classification (CAPEC) [CAP12] describes the most common attack patterns.

8.2.2 Web Service Security Framework

There are several requirements for web services and SOA security, of which integrity, confidentiality, and availability are the most ubiquitous. Integrity guarantees an unchanged message transmission across heterogeneous intermediary services. Confidentiality requires that a message's contents can not be exposed to unauthorized users. Finally, availability ensures that legitimate users receive the right services [BMPS09]. In addition, authentication and authorization means are provided in web service discovery to ensure that all the parties sending/requiring services send/get the authentic and correct information. These above-mentioned security aspects need to be considered at both the network- and service-level to guarantee SOA security. Service-level security could involve the protection of a web service's resources against unauthorized access, the avoidance of *denial of service attacks* where a malicious party could make

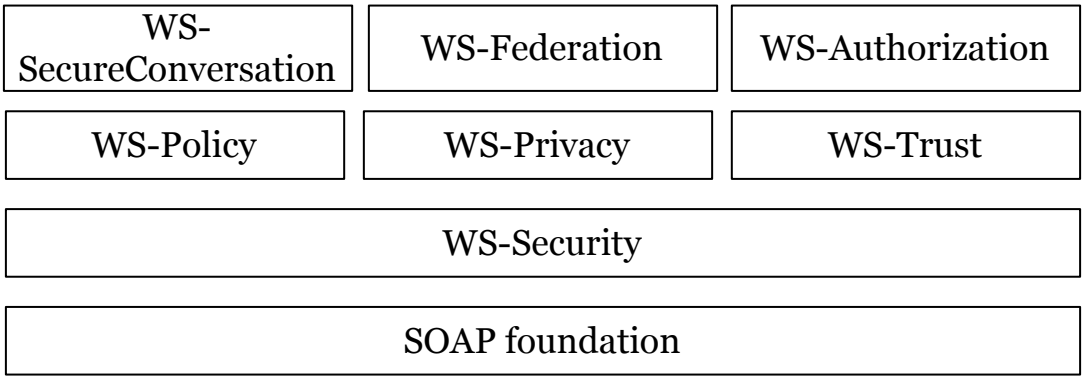


Figure 8.1: Web service security framework

the web service unavailable, and ensuring the integrity and confidentiality of discovered web services for protection of service consumers. Similarly, the data protection in SOA is not limited to simple client-server interaction, since the business process could be executed through multiple web services. Considering these facts, several web service security standards have been proposed, primarily by IBM and Microsoft [IC02]. We represent the web service security framework (cf. Figure 8.1), and briefly survey the primary standards used at both the communication and application layer.

- *Secure Socket Layer (SSL) and Transport Layer Security (TLS)*: provide the basic level of security for point-to-point communication in web applications [FKK11] [DR08]. SSL/TLS is a protocol between the network and application layer protocol, TCP and HTTP. It provides facilities for confidentiality (e.g., symmetric cryptography), integrity (e.g., Message Authentication Code), and authenticity (e.g., public key encryption) with optional session caching to optimize the computational load. There are several inadequacies for applying this protocol to web services. Web service communication is end-to-end; hence several intermediaries might process the message during transmission to the final recipient. The intermediary services and applications can maliciously modify the data while allowing the data to appear to be transmitted. In addition, there is no control over partial encryption of the message.

- *XML Security*: securing the selected language for exchanging data in web services. XML Encryption [IDS02] and XML Signature [ERS+11] are standards that provide normative indication to represent and convey the digitally encrypted web resources along with the signatures in an XML format. These

standards support confidentiality at an application layer, when the SSL/TLS is transport layer only. Here, the encrypted data is distinguished from the information that includes the encryption key and algorithm. The integrity of the encrypted data could be ensured with a signature that is represented in standardized XML format. This allows the required parts of the document to be signed, as opposed to the conventional signature method (e.g., PKCS 7[8]).

- *Security Assertion Markup Language (SAML)*: exchanging security information as an *assertion* between trusting parties [RHPM06]. The assertion is composed of security attributes, authentication, authorization decision (e.g., the action that the entity is entitled to execute on a resource), and authentication context (e.g., detailed information about the strength of authentication). The security of the system is based on the validity of the assertion, independent from the security of the issuer (i.e., trusting issuer party). SAML supports end-to-end transactions where there are several intermediary web services involved. Several weaknesses have been identified that make this framework imperfect as a secure technique. For instance, an issuer could control neither lifespan nor the parties which share the delivered assertion. Subsequently, the attacker might access the encrypted contents of assertion, or a valid party could use the generated assertion maliciously in another transaction.

- *SOAP Message Security*: Reinforcing end-to-end protection in web service transactions to support confidentiality and integrity. Several standards are utilized to achieve the SOAP message security in SOA. Implemented examples of these standards are WS-Security (WSS) [NKMHB06], WS-SecureConversation [NGG+07], and WS-Reliability [DKP+09]. WSS allows the encryption of different parts of a single SOAP message in a multi-hop path. It is designed for referencing and including multiple formats of security-related information, referred to as security tokens (e.g., SMAL assertion 8.2.2, Kerberos tickets,[9] X.509 certificates[10]), signatures, and encryption technologies. To authenticate the security token that is claimed by issuer, third parties have to sign the token (e.g. XML Signature 8.2.2).

[8] `http://tools.ietf.org/html/rfc2315`

[9] `http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf`

[10] `http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf`

Consumers and web services need to exchange multiple SOAP messages to complete a relevant transaction. To preserve a secure session for such communication, WS-SecureConversation has been proposed. A security context token is created among the communicating parties, holding a secret key that could be referenced from the same message for the lifetime of a communication. Thereby, the security and efficiency of exchanging multiple SOAP messages is ensured.

WS-Reliability prevents loss, duplication, and reordering of messages. It accommodates a level of reliability towards secure SOAP communications. It is a protocol between exactly two parties, at two levels. It is between applications as well as a contract with the underlying message processor. The contract assures the quality of the delivered message through four main operations (submit, deliver, respond, notify).

- *Web Service Policy*: "Defining goal, course or method of actions to guide and determine present and future decisions" [WSS+01]. In the context of web services, a policy determines a variety of conditions under which a service can be used. The Web Service Policy Framework (WS-Policy) standard allows the policies regarding different aspects of a web service to be expressed in a single model which encompasses message security, access control to resources, and Quality of Service (QoS) [BMPS09]. Herein, the behaviour (requirement) of the policy is specified as a policy assertion. WS-PolicyAssertion [WS-03] and WS-PolicyAttachment [WS-07] are used in the web service policy framework to specify generic domain-specific policies that are perceivable by any client. For instance, in a domain, the integrity could be recognized as a policy assertion, and also attached to an entity of WSDL (e.g., portType) or a repository (e.g., UDDI). Through such policy attachment, the policy is exposed to web service consumers while they are searching for suitable services.

  To protect web service resources and digital information from unauthorized consumers, an access control policy must be specified, deployed, and maintained [BMPS09]. EXtensible Access Control Mark-up Language (XACML) [Ris10], and eXtensible Right Markup Language (XrML) [And02] prevent the malicious use of resources and restrict the dissemination of information to the intended recipients. In a web service environment, an XACML assertion [And06] expresses the policy requirements and behaviours that the service providers are able to provide to other parties. Then the web service provider and consumer XACML

assertion compatibility is computed in order to discover the right services. In XACML the policy is made from a set of distributed rules listed from different organizations. As such, multiple rules and policies might be applicable to a given request. XACML can prioritize the order of evaluation of policies and rules with decision algorithms. On the other hand, XrML is an XML language used to describe access privileges (e.g., condition and fee) that are granted from the provider to a principal. The principal is obliged to exercise her/his rights under the issuer's terms and conditions.

- *Secure Publishing Methods*: avoiding web service information tampering during the search and discovery process through any intermediary. We explain several techniques to sustain the integrity of information over communication, as well as the application in the remaining sections. Another issue that should be taken into account is the integrity of a repository of registered web service information, in a third-party architecture. In such a repository, the web service information (e.g., the structures and standards) are published, advertised, and discovered through a discovery agent to the consumer. This information might easily be penetrated by an untrusted intermediary. A naive solution might be verification of the third parties, but that is not a feasible approach for a large web-based system. Besides, using a typical XML digital signature is considered inappropriate. When consumers query the part of the repository that is not known in advance, even if the service provider has signed some elements, the consumer cannot detect whether a third party has altered the response.

  To address the integrity of data without requiring a trusted discovery party, Merkel's hash tree techniques [Mer89] are employed. These techniques have previously been applied in different contexts, in particular, UDDI integrity and authenticity issues [BCF04]. This method aims to forward the query result along with the provider's signature of the document on which the query is performed. Having this information, the client could re-compute the signature and compare the computed signature to the provided signature, to verify authenticity. More precisely, a Merkel hash tree yields a single digital signature (Merkel signature) for the whole XML document through the concatenation of the Merkel hash values of each individual element of the document. Therefore, the integrity of the whole document, as well as any portion of it, is traceable. For instance, the service provider can sign each element of the UDDI (e.g. business Entity,

tModel) included in UDDI specification (*dsig* : *Signature*) [CHvRR04]. An enhanced version of UDDI has been implemented, in order to convey the Merkel signature with the find services for a given query [BCF04].

- *Service Level Agreement (SLA)*: SLAs have been applied to various topics and generally include commitments to deliver different types of compensation in response to poor performance of the service. Through the contractual obligation of the party (e.g., service provider), outsourcing becomes less risky. To avoid any possible opportunities for cheating from either party, the SLA should be written clearly and precisely so that the intention of the agreement is retained and not misinterpreted. There are two main properties that clients should take into account. First, "what the service offers"; that is, the behaviour of the service in a manner that is described. Then, "when the service arrives", which refers to performance and latency. These mentioned service properties are referred to as "reliability" and "timeliness" conditions [SRE10], and are known as Quality of Service (QoS). Here, one of the concerns is that the service provider cannot control the timeliness due to lack of capacity of some resources and unpredictable request rate at run-time. Therefore, the service provider is required to specify a limit on the service request rate (throughput), as part of the SLA. This enables the service provider to recover timeliness and reliability against the resource contention that may be caused by a malicious client attempt. Additionally, early termination and payment are other aspects that increase the financial risk of outsourcing, and are being included in SLAs.

  For the purpose of automated SLA evaluation in a heterogenous environment, several studies have been done [JF05]. More specifically, monitorable SLA is a key requirement. Pertinent evidence must be presented in order to avoid any sort of claim from either party. In the three parties paradigm in SOA, the SLA should be monitorable to both service requester and provider. This is referred to as "mutually monitorable" [SRE10]. Also it is monitorable by a third party trusted by both requester and provider, named arbitratable [SRE10]. In addition, the SLA by itself couldn't eliminate all risks and must be used with a management mechanism to determine the client expectation, while the service provider makes provision for such expectations, and becomes protected against any unfair criticism [SMJ00].

8.3 Related Work

Previously, researchers have approached the problem of service verification and certification by applying two families of techniques to prove that a service possesses a given property: *test-based techniques*, which require the application of suitable tests to the service, and *formal-based techniques*, where an abstract model of the service (e.g., a set of logic formulas, or a formal computational model such as a finite state automaton) is generated and used for validation [DAE09].

Test-based certification of software components is a time-honored software engineering problem; for instance, assurance levels from 1 to 4 of the well-known Common Criteria standard [Her02] are themselves test-based [DAE09]. Collecting evidence supporting security attributes differs greatly from standard software testing procedures, making existing security certification schemes not applicable in the service ecosystem. The loosely coupled nature of web services (as compared to traditional component-based software systems) poses strict limits on the way testers can interact with the services making the usual stub-based testing techniques hardly applicable to web service security testing. A major problem is that web service source code is usually inaccessible, and remote execution may involve a cost. In addition, existing certification techniques have been provided for static and monolithic software [Her02, US 85], where certificates are usually human-readable statements signed by a trusted certification authority, and do not suit the service requirements in terms of run-time evaluation of certificates during the discovery process.

There are two major approaches to testing a web service [CDP06, Blo02]: 1) consider web services as independent software components to which traditional interface testing methods can be adapted, by considering the released interfaces and protocol bindings; 2) consider web services as composite elements, where integration testing methodologies should be adapted. Web services can be tested from the perspective of the different stakeholders (i.e., developer, provider, integrator, certifier, and user) and there is the need to specify who can perform a test, which types of tests are needed, and what are the challenges in carrying out the tests and documenting their results. Others focused on the automatic generation of test cases starting from WSDL released by the service provider or general service specification [HM07, JDS09, Mao09, NS08, BDTC05, DY06]. For example, Noikajana and Suwannasart present a new methodology in which the web service specification is used to generate test cases based on a decision table [NS08]. Moreover, an approach

to testing web services using fault-coverage to check conformance of the web services to their WSDL specification has been proposed by Wen-Li Dong et al. with the goal of automating testing [DY06].

Model-based certification of web services has also been used for service representation and certification [DESS09]. Certificates based on formal proofs deal with verifying the properties of the models of web services [GAHL00]. A milestone is the definition of Web Services Business Process Execution Language (WSBPEL) [AAA+07], an industry standard for specifying workflows. Other industry driven modeling approaches [SGS04,KKKR05] are based on the Unified Modeling Language (UML) [RJB04]. A UML extension also exists and enables, with some limitations, the modeling of security attributes to be provided by the system [JÖ2, LBD+02].

The problem of service verification has been extensively studied and is the topic of several research projects, highlighting its intrinsic importance in the context of the future Internet. An interesting approach is the one taken by the AVISPA (Automated Validation of Internet Security Protocols and Applications) research project [AVI06] which defines a High Level Protocol Specification Language (HLPSL) for modeling communication and security protocols [CCC+04]. AVANTSSAR (Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures) introduces a platform supporting the validation of trust and security aspects of service-oriented architectures and automated techniques to reason about services composition security [AVA]. One of the proposed tools, the SAT-based Model Checker [AC04, Com05], was recently used to identify a security flaw in the SAML-based Single-Sign-On protocol for Google Applications [ACC+08]. The SPaCIoS (Secure Provision and Consumption in the Internet of Services) project concentrates on the problem of providing security in a complex service ecosystem [SPA12]. It aims to provide a solution and new generation analyzers for automated security validation of services not only at production time, but also at deployment and consumption times. SPaCIoS tries to combine technologies for penetration testing, security testing, model checking, and automatic learning. Moreover, project ANIKETOS (Ensuring Trustworthiness and Security in Service Composition) provides service developers and providers with a secure service development framework [ANI12]. Such a framework includes methods, tools, and security services that support the design-time creation and runtime composition of secure services in environments where both services and threats are evolving. Finally, the project ASSERT4SOA (Advanced Security Service cERTificate for SOA) aims to define and develop a certification infrastructure dealing with both test-based and model-based

certification to provide a certificate-aware SOA [ASO10]. Gürgens and Rudolph proposed another interesting approach to find security flaws in a number of key exchange, authentication, and non-repudiation protocols [GOR02, GR04, GRS+07]. Their approach is supported by the Simple Homomorphism Verification Tool [fSITS07].

Another important area of research analyzes the computation of test results at service invocation time and the definition of enhanced UDDI supporting QoS. Tsai et al. propose an enhanced UDDI server specification to manage check-in and check-out testing for services [TPC+03]. In particular, the check-in test is performed when the service is first registered in the system, while the check-out test is done when the service receives a request from a user. Ran presented a new service discovery model taking into account both functional and non-functional requirements and proposes a QoS certifier responsible for checking claims made by the service provider [Ran03]. Serhani et al. illustrate an architecture based on a QoS broker for efficient web services selection [SDHS05]. After verification of the service by the broker, the client can select services on the basis of its QoS requirements. In this chapter, we focused on security certification and on the definition of a discovery process that considers the certified properties and the way in which these properties are proven to hold.

8.4 Expressing Service Security Requirements

Any *assurance process* consists of a set of activities aimed at increasing the users' confidence that a given service will satisfy their functional and non-functional requirements. Functional requirements are already considered in the context of dynamic web service search and discovery, while performance assurance can be addressed via suitable Service Level Agreements (SLAs). Other non-functional properties, such as security and reliability, are usually not addressed. Therefore, there is a need for a solution that 1) permits to specify security requirements that a given service must satisfy [HKK06, KR04], and 2) uses these security requirements in the context of web service search and discovery.

There are two key aspects to the notion of *service security requirements*: (1) security requirements describe a set of security attributes whose semantics are shared between the service supplier and the service user; and (2) security requirements specify the process and techniques (evidence) used to verify that these security attributes hold for this service [AAD+12] [AAD+11b].

The first building block to support an assurance process for service security is

the definition of the set of security attributes that are of interest. The literature defines several classes of security attributes including *authenticity*, *integrity*, and *confidentiality* [KR04]. In a Service-Oriented Architecture (SOA), security attributes may concern message-level security (i.e., security of data in transit) and service-level security evaluating the service implementation (i.e., security of data at rest). In other words, we distinguish between security attributes that can be proven to hold for a given service at container-level and those that need to be certified on the real service implementation. The process of evaluating a service at container-level considers the compliance of the container with web service security standards (e.g., WS-Security [NKMHB06] and WS-SecureConversation [NGG+07]) and the correct enforcement of security policies (e.g., WS-Policy [VOH+07]) defined by the service provider, to prove that the security attributes are supported by the service deployed in the container. In contrast, service-level security considers security attributes that need be proven by analyzing and evaluating the real service implementation. Service-level properties complement the container-level ones by providing an overview of the service that goes beyond the service interface, and considers its implementation and the developed operations.

Here we distinguish between *abstract properties*, as for instance confidentiality and integrity, and *concrete properties* that enhance abstract ones by adding *class attributes* [AAD11a]. Indeed, no security property is entirely specified without an adversarial model, that is, a description of what an attacker can do to compromise the property. Therefore our class attributes represent (1) the threats against the property to hold (e.g., eavesdropping or replay attack), and (2) the security functions that ensure that the property holds (e.g., the access control system) [AAD11a].

Security attributes are organized in an abstraction hierarchy (cf. Section 8.5.2). Each node in the hierarchy is a security property of the form $p{=}(\hat{p},A)$, where $\hat{p}$ is an abstract property and A is the set of class attributes in the form $a{=}v$, with a the attribute name and v its value. The first-level nodes of the hierarchy are represented by all security attributes $p{=}(\hat{p},-)$, namely abstract security attributes with no class attributes specified. An ordering relation $\preceq$ between properties p_i and p_j ($p_i \preceq p_j$) indicates that p_j is a specification of p_i, if and only if $p_i.\hat{p}{=}p_j.\hat{p}$ and for each class attribute $a \in A$, $p_j.a$ dominates $p_i.a$.

Upon the specification of security attributes, an important aspect of the assurance process is the definition of requirements over the mechanisms used to provide evidence for those properties. We can verify the service support for a given property using two

different approaches: *a test-based approach* providing evidence that a test carried out on the software has given a certain result, and *a formal approach* providing evidence based on an abstract model of the service. We focus on test-based evidence: each security property can be associated with one or more classes of tests, which in turn contain a set of test types (e.g., equivalence partitioning, fuzzy/mutation) used to generate the test cases providing the evidence that the property is supported by the service. As for security attributes, test-based evidence (including the results of test execution) is stored in a certificate awarded to a service and can be compared with users' requirements, to search and discover services on the basis of their non-functional characteristics.

Section 8.5 presents our approach to service security certification using a test-based mechanism for certifying that a service possesses a concrete security property that can be integrated into the lightweight web service discovery technique presented in Section 8.6.

8.5 Security Certification of Services

Our approach relies on a model-based testing certification solution for services that produces security assurance metadata (i.e., *test-based evidence*) supporting a given security property for the service. The test-based solution used in this paper requires the generation and execution of suitable test cases on the service, starting from a formal model of the service itself. The test-based mechanisms used to certify a service and the related results compose the certificate evidence.

8.5.1 Model-based Testing

Model-based testing targets the representation and certification of complex web services. Services can be modeled as finite state automata and transition systems for the automatic generation of test cases, and the evaluation of correctness of tested services [FTdV06,KKK+06]. Keum et al. [FTdV06], authors proposed symbolic transition system (STS) [FTW04] as a suitable solution for the certification of complex web services that involve communications, allowing the specification of typed variables, guards (i.e., constraints on state transitions), and actions (i.e., function calls).

Anisetti et al. [AAD11a] proposed an extended version of STS suitable for generating tests based on container and service behavior. Specifically, web service description

language (WSDL) specification is used for the creation of STS models. The definition of service interface, describing operations, and input and output data models, can be exploited to build a specific model describing three states for each described operation:

- *initial state*, no inputs have been received;
- *intermediate state*, the inputs have been received but the outputs have not been produced yet;
- *final state*, the outputs have been generated and returned to the counterpart.

The modeling can also be extended to the web services conversation language (WSCL) specification, where WS-Conversation policies and constraints are defined and applied. In particular, WSCL regulates the communications between users and services, which could be modeled in STS to take into consideration data exchanges [AAD11a]. Also, the WSCL specification can be extended to integrate implementation details describing internal functions of the services.

Finally, the model can handle container certifications; STS service models can be extended to include interactions between the customer and the service, which are outside the service implementation but part of the security specification flow (e.g., key exchange for data integrity).

8.5.2 Security Assurance Certificates

When test-based certification is used, test cases and related evidence can be attached to web service interfaces in order to evaluate the assurance level. According to Damiani et al., three different certificate types can be identified, each one characterized by specific metadata and a certification process [DESS09].

Self-Certificate (SC): SCs are characterized by a four-way interaction between service provider (SP) and consumer (C). First of all, C sends a request to SP specifying the list of security attributes to be certified on the service. SP is already mapped to each service consistent test suites, including functional and QoS-based test cases; those tests are then used to build the reply, sending to C the test cases related to the specified set of properties. If the reply is satisfactory, C can directly execute the test cases on the service and analyze the results. It is important to note that this mechanism does not require a trust relationship

between C and SP; however, it reduces the number of actors involved in the certification process and, therefore, the certification time.

Lightweight Certificate (LC): LCs introduce a new actor in the certification process, namely a third-party certifier ($3PC$). In this case, all tests are executed by the $3PC$. A consumer C can send a request to $3PC$ specifying the requested security attributes, and a list of candidate providers. Then $3PC$ can contact each specific provider, supply the test cases, and apply them to services. $3PC$ can also play the role of certificate repository, storing available certificates for future usage.

Collaborative Certificate (CCert): CCerts are generated and stored independently from C requests. An extension of the UDDI protocol could support CCert, as well as the storing and managing of test suites. Test suites are signed by SP during the service registration phase; then $3PC$ can access them periodically to verify the test results, to generate missing certifications, and to invoke one or more tests as needed to reconfirm or strengthen service quality.

In the remainder of this chapter, we further develop the CCert scenario to enable classifying services based on their level of assurance. We adopt this well-established approach [DESS09] to provide users with a subset of certified services. Then, we show how the CCert scenario can be implemented, using our approach so that real, practical implementations can properly meet user needs.

8.6 Selection of Services

We now focus on the practical challenges that arise when implementing a web service discovery (WSD) system that considers certificates for security assurance. When security attributes of individual web services are certified, changes in the context may require re-certifying them at run-time in a context-agnostic manner. In this section, we illustrate the challenges arising in our context with a simple example and propose a first conceptual architecture for adaptable assurance-based service discovery.

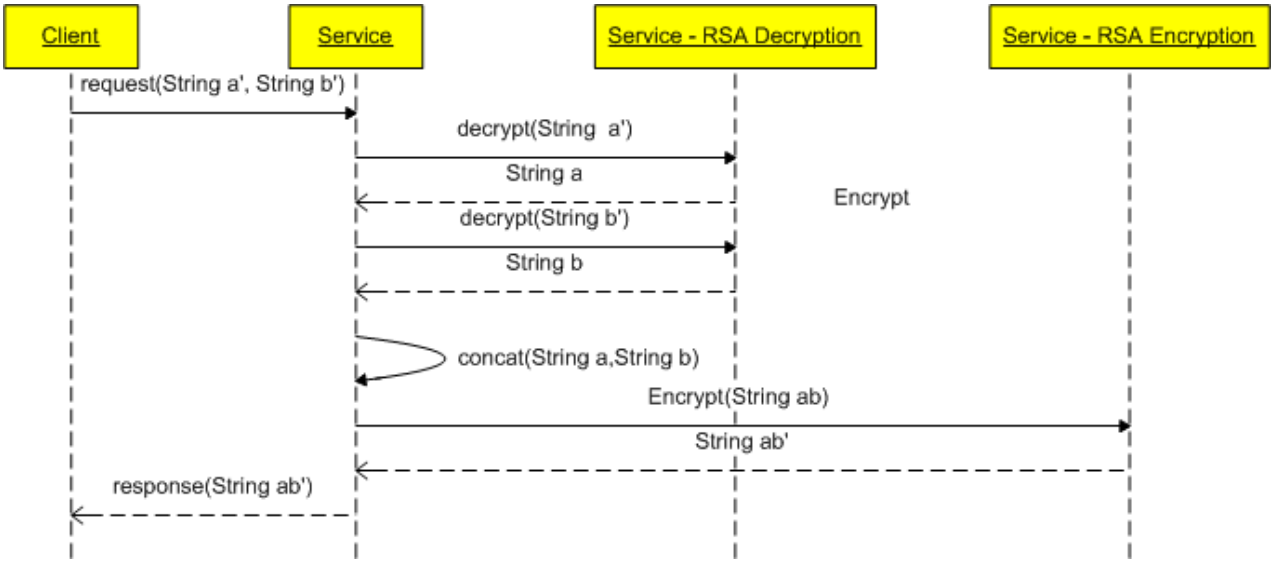


Figure 8.2: String concatenation service sequence diagram.

8.6.1 An Example of Service Discovery and Dynamic Certification

Let us assume that a web farm is supplying a secure storage service based on an ad hoc security framework. The service is tested in a given context (i.e., on a physical machine), and CCerts are issued (i.e., static certification), containing evidence that all the tests have passed successfully (cf. Section 8.5). For the sake of simplicity, we consider a simple service implementing the concatenation of two strings, whose sequence diagram is presented in Figure 8.2.

The service first receives as input the two RSA encrypted strings (*String a'* and *String b'*), then decrypts the strings (*String a* and *String b*) concatenating them (*String ab*), and finally encrypts the concatenated response using the RSA algorithm (*String ab'*). Such encryption is performed by the service itself and not by the container. The example can be easily reformulated in the case where the container performs SOAP encryption according to WS-Security. In this scenario, it is the container that, using an ad hoc component (e.g., Apache Rampart), is responsible for encrypting and decrypting the input strings and their concatenated version. In the following we consider the first scenario in which encryption and decryption are managed by the service implementation.

Test cases in the service certificate assert the confidentiality of the response data by providing the encrypted response string, the key, and the input parameters used to

compute the response. Suppose that the test suite includes some critical input parameters that may generate a service response too big for the local RSA implementation to encrypt. In fact, in the case of very long strings, their concatenation could exceed the number of characters accepted by the RSA implementation; furthermore, different systems could manage very long strings differently, having unpredictable results at client-side. In particular, those tests are to guarantee that no data leakage happens during the execution on the server, and that the information is secure at the application level during the entire exchange of messages between the clients and the service. Each time a user searches for a secure storage server certified for the "Confidentiality" property having the class attribute *MsgEncryption* set to "RSA", the repository will return the certificate along with the property and the evidence of the tests applied to the service interface (i.e., triples <*EncrypTedtext, key, PlainText*>).

The above discussion assumes static certification. However, there are cases in which changes in the service context and environment invalidate the certificates awarded to the service. As an example, consider a service provider rationalizing its software infrastructure, moving its storage service to the cloud. The service is instanced in a dedicated virtual machine and, from that moment on, the certification is no longer valid. In fact, moving the architecture to a virtualized system has added a new virtualization layer to it, and the encryption/decryption takes place on the same physical server hosting other containers and services. In this situation when a user asks for a string concatenation service with the confidentiality property on parameters, the original certificate can no longer be considered since the context has changed. The provider can dynamically re-create the certificate re-running the test suite enclosed in the certificate evidence, and obtain a certificate for the new context. Alternatively, re-certification can be carried out by the service user, executing the test suite in the old certificate in the new context. In principle, server-side re-certification may be preferable for two main reasons: 1) the re-certification process could be too heavy for common client infrastructures; 2) online re-certification can represent a security threat for the service.

The same idea can be applied in cases where services are dynamically moved from a server to another, or aggregated on the same virtual machine to optimize resources. Notably, not only changes in the hosting infrastructure require service re-certification. In fact, users can change their infrastructure, thus introducing the need for a new selection process or at least service re-certification. For instance, doing online banking in a different workspace such as mobile, owned, or shared computers, may change the

required authentication method and encryption algorithm.

In summary, to improve service certification and selection at run time, it is important to consider the context and environment in which the services are deployed and to handle context-aware re-certification for WSD. This idea is complementary to the approach by Anisetti et al. which mainly deals with static certification [AAD11a].

8.6.2 Adaptable WSD Security Certification

The above example is a good starting point towards dynamic and adaptable WSD based on security certification. Adaptable service discovery aims to recognize which security attributes need to be certified to support the dynamic selection of services in a context-agnostic manner. Moreover, it provides a mechanism that enables service consumers to define their preferences in terms of certified properties, evidence, and tests, and automatically matches them against the certificates awarded to a service at discovery time.

The service security attributes that are used for an adaptable WSD security certification can be further broken down into the following groups.

- *Abstract Security Property (ASP)*: An abstract security property represents a generic security requirement for the service such as confidentiality, integrity, and authentication. It can also be referred as a concrete security property with no class attributes.

- *Concrete Security Property (CSP)*: An ASP enhanced with class attributes. Given two instances of CSP, p_i and p_j, based on the same abstract property $\hat{p}$, p_j is a specialization of p_i, if a certificate proving p_j always proves p_i. For instance, given the abstract property *integrity*, and two concrete properties p_1=($integrity$,{$algorithm = RSA$, $|key| = 1024bits$}) and p_2=($integrity$,{$algorithm = RSA$, $|key| = 2048bits$}), a certificate proving p_2 always proves p_1. The relation between p_1 and p_2 is called intra-property relation, because it involves properties with the same ASP and only considers the class attributes.

- *Semantic Security Property (SSP)*: An SSP is a concrete security property. The only difference between CSPs and SSPs is that the latter refers to order and equivalence relations, called inter-property relation, involving different abstract properties. Inter-property relations are defined based on expert knowledge. As an example, given the two properties p_1=($authenticity$,{}) and

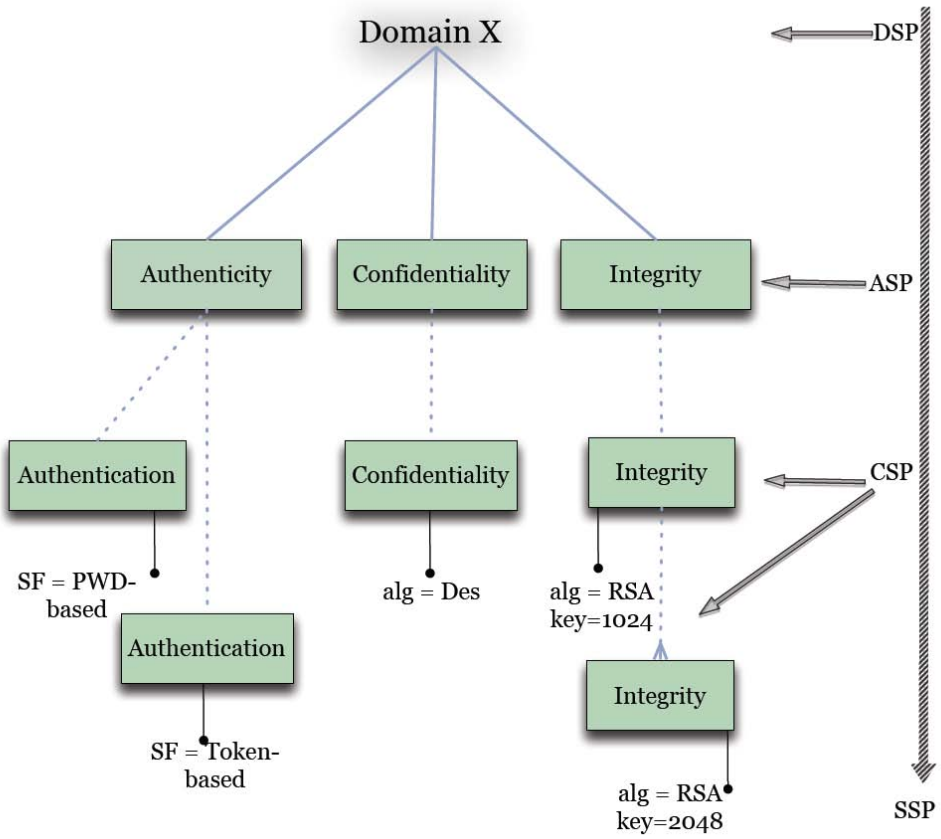


Figure 8.3: An example of a hierarchy of security attributes for Domain X

p_2=(*non-repudiation*,{}), p_2 implies p_1 meaning that each certificate for property p_2 also applies to property p_1. ASP, CSP, and SSP form a hierarchy of security attributes.

- *Domain Security Property (DSP)*: A domain-aware security property specification. Since a hierarchy characterization of a security property could be different in different domains (both intra- and inter-property relations), a DSP enables an accurate evaluation of the security attributes relevant for a given domain. This alleviates the problem of having a complex hierarchy of security attributes that are suitable for all domains. Fragments of the hierarchies for *Domain X and Y* are depicted in Figures 8.3 and 8.4, respectively.

After defining different categories of security attributes, we need to integrate the certification process into the WSD process. To realize this, we first introduce the con-

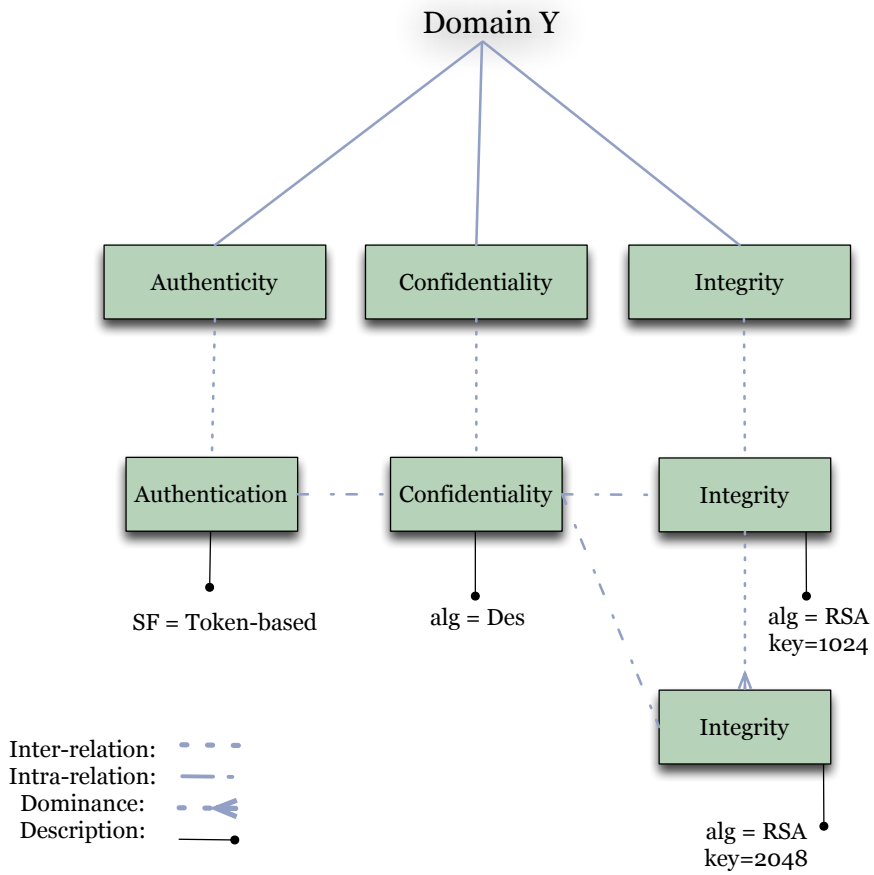


Figure 8.4: An example of a hierarchy of security attributes for Domain Y

cept of Assurance-Level Agreement (ALA), as the part of the Service Level Agreement (SLA) that exposes the security attributes supported by a given service in the form of machine-readable certificates. The user has then to define its requirements in a *policy*, that is, a machine-readable format of its security requirements. Finally, WSD matches them against certificates to provide a list of compatible services. Figure 8.5 depicts the evaluation of ALA in WSD. Each certificate is an XML-based document that stores and manages security attributes and test evidence. Certificates can be attached to web service interfaces in order to provide the assurance information (i.e., artifacts and evidence) to be matched during the discovery process.

The format of the service certificate can be specified using the XML-based schema shown in Figure 8.6. The service certificate includes the following main sections.

- *TestProperty*: Information about the certified security attributes. Each property includes the *PropertyName* and a set of *ClassAttribute* fields.

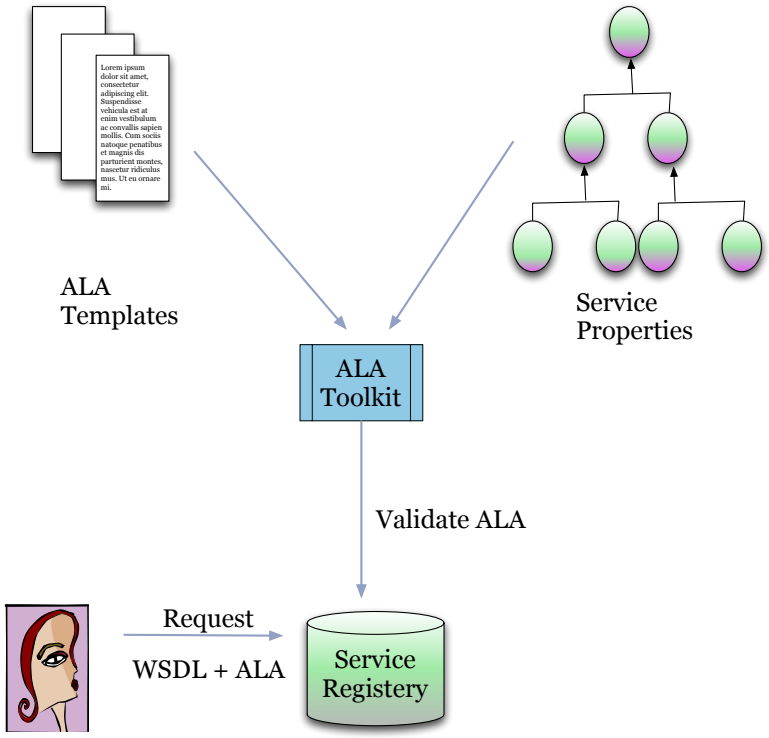
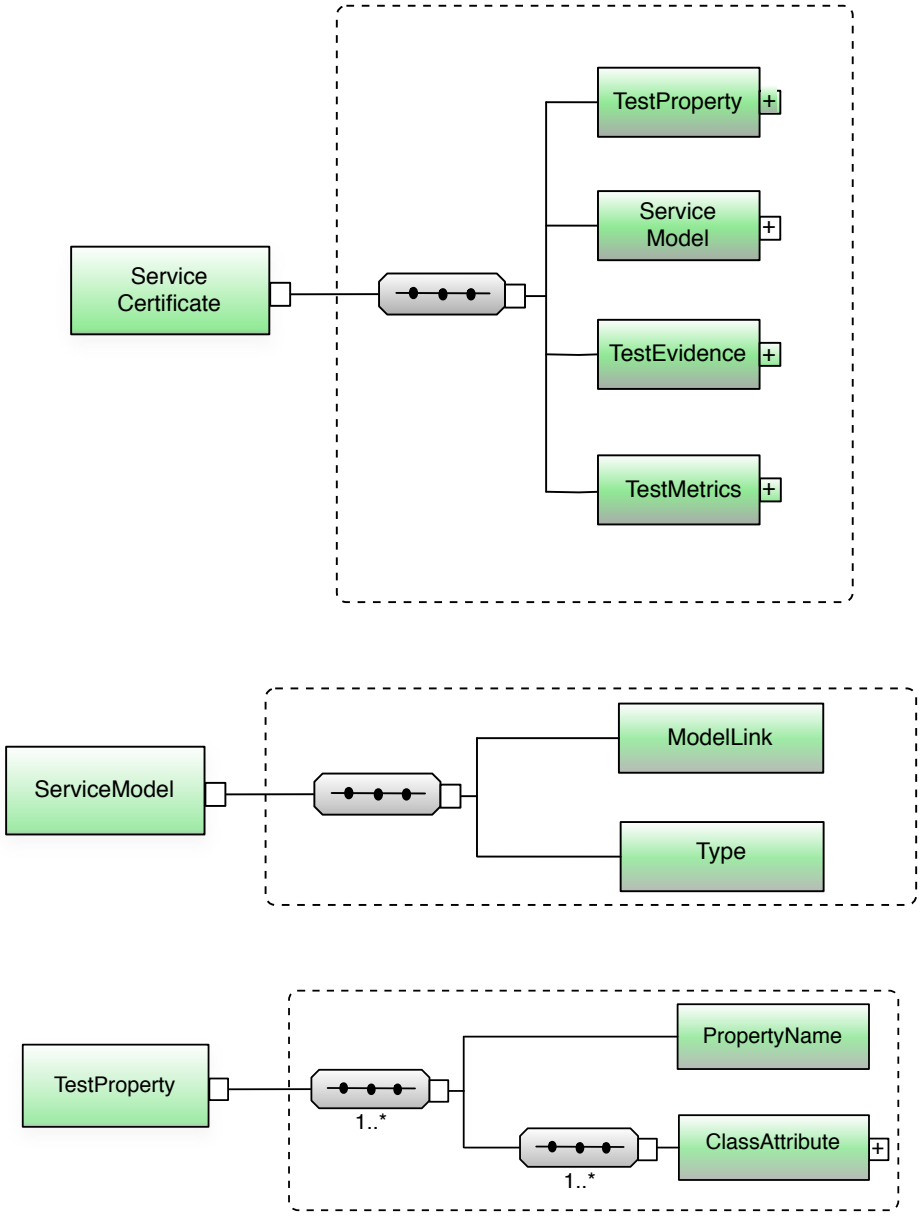


Figure 8.5: ALA in WSD

- *ServiceModel*: A reference (i.e., URI), named (*ModelLink*), to the location where the model of the service is stored, and the type of service model (i.e., WSDL-based, WSCL-based, implementation-based) is declared in the *Type* element.

- *TestEvidence*: All artifacts related to the test cases executed on the service for its certification. It includes test class (*TestClass*), type (*TestType*), attributes (*TestAttribute*), specifications (*TestSpecification*), and the result of test case execution (*TestResult*). *TestSpecification* is specification about the real test case that is declared through test id, description, and a link to the test model that is used to generate the test case. *TestResult* is a set of pairs holding a reference to the test case and a pass-fail result.

- *TestMetrics*: A set of metrics representing the quality of the test cases executed on the service. These measurements are used in WSD matching to compare the services by representing the result of different tests on various services.


Service Certificate
TestProperty
Service Model
TestEvidence
TestMetrics
ServiceModel
ModelLink
Type
TestProperty
1..*
PropertyName
1..*
ClassAttribute

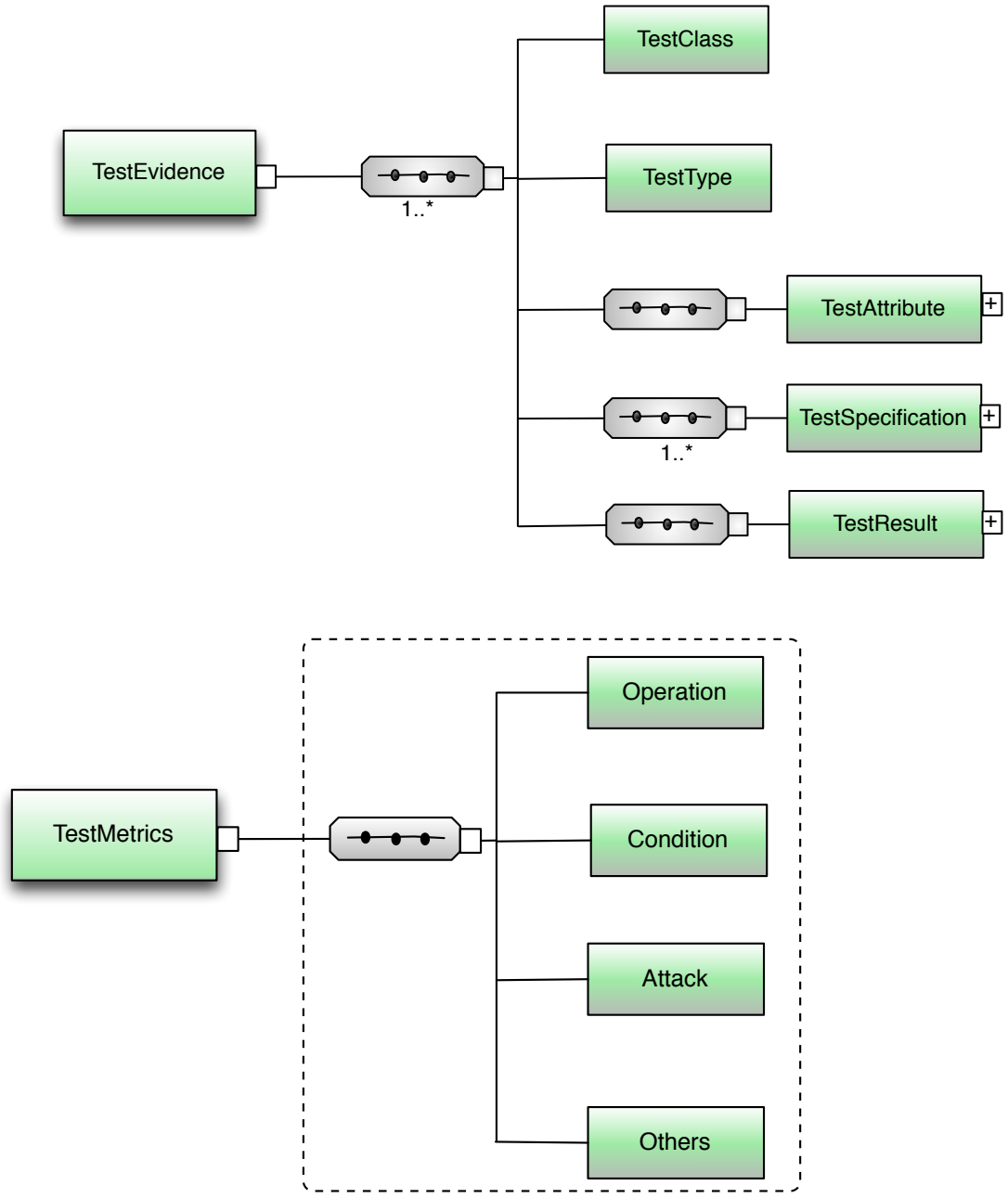


Figure 8.6: Certificate schema

The above-mentioned information is used by WSD to find the certified web services that comply with the user's request.

8.6.3 A Conceptual Architecture for an Adaptable Assurance-based WSD

Our approach to dynamic certification of service security attributes consists of two phases: *Static Certification (SC)* and *Dynamic Certification (DC)*.

Whereas the SC process involves certification of services at development time, independently from the context in which the services are deployed, the DC process supports run-time certification of services. Based on the generated service security certificates (*ServiceCertificate*) depicted in Figure 8.6(a), WSD provides an assurance-based selection process. Upon receiving a consumers' requests with security preferences, it matches them against the *ServiceCertificate*. According to the matching results, WSD returns a set of services compatible with the users' preferences.

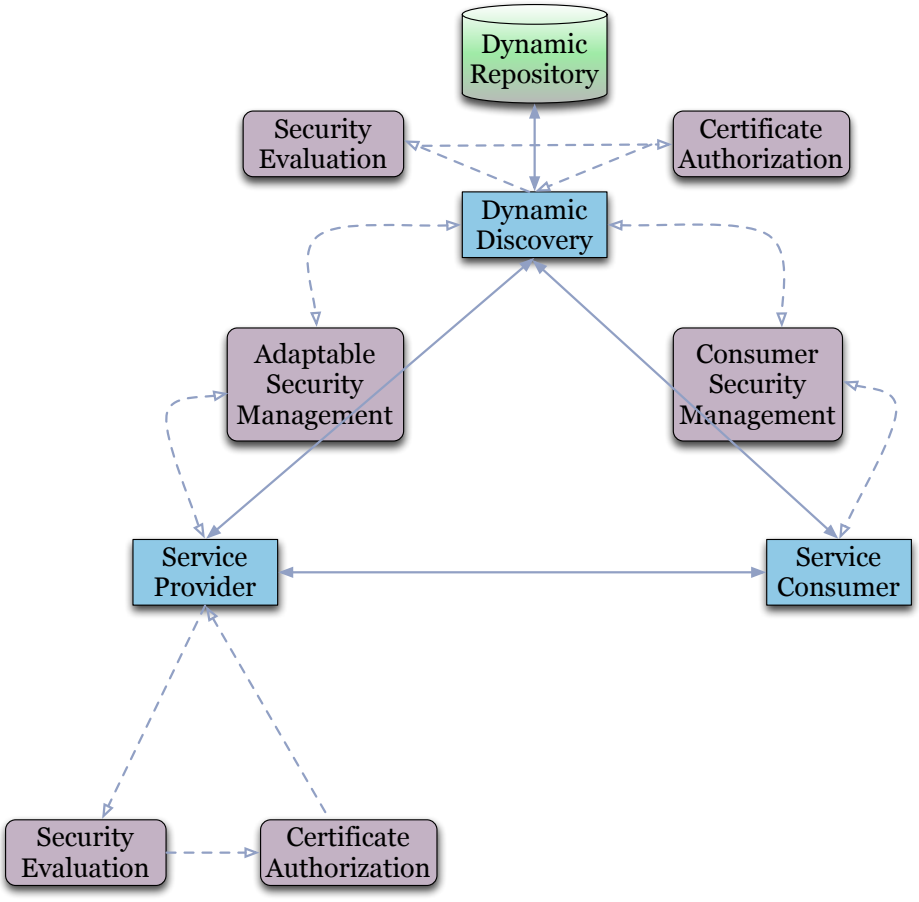


Figure 8.7: Overall architecture of adaptable assurance-based WSD

Figure 8.7 depicts the overall architecture of the adaptable assurance-based service selection, which is composed of the following components and processes.

Service Consumer: The party requesting access to, or integrating a remote service, according to users' preferences.

Service Provider: The party providing remote web services that are accessed by service consumers.

Dynamic Repository: The component storing the web services together with the security certificates awarded to them. Here, the certified services are registered and published, and periodically re-certified to assess their security attributes.

Consumer Security Management: The component dealing with the consumer's security requirements and preferences. It allows service consumers to define their preferences in terms of certified properties, evidence, and tests.

Adaptable Security Management: The component enabling automated run-time service certification, beyond the security implementation and pre-deployment certification of the web service. It monitors the properties that could hold at run time and identifies new and old security attributes that need (re-)certification. The module evaluates the propriety of the security attributes claimed by the service.

The run-time web service assurance process is executed in three distinct phases as follows.

Security Evaluation: An accredited process that executes test cases for service evaluation. It generates new test cases, if needed, according to the security requirements given by the "Adaptable Security Management" component and the service security specification. If the required test evidence is not available in the service certificate, a new set of test cases is generated and executed on the service (run-time evaluation). As a result, new evidence is generated and used in the assurance process.

Certificate Authorization: Services are certified using the evidence provided by the Security Evaluation phase. It generates an evidence-based certificate guaranteeing that a set of test cases is executed on the service, or on an entire business process in a service container.

Dynamic Discovery: The conformance of the selected services with the consumer's security preferences is evaluated by means of a matching process. The latter

measures the degree of compliance between users' preferences and service certificates.

In summary, run-time certificate matching allows a service consumer to ascertain that the assurance level provided by the service certificate complies with its own preferences. This solution increases consumer confidence because their assurance requirements have been met at service execution time and in the context of certification.

8.7 Summary

The rapid worldwide deployment of web services on enterprise IT infrastructure is the enabler of a new generation of applications. Web service security standards are now firmly in place, but the provision of a security architecture that can ensure that key security attributes actually hold at service execution time is still an open question.

Current solutions to this problem are based on compliance to standards in a static execution context and do not address security requirements of service-oriented applications when execution contexts can change and evolve rapidly.

We posit that our context-aware certificates will enable administrators to specify their requirements in terms of security properties to be re-checked when the execution context changes. In this chapter, we discussed the notion of context-aware certificates and gave a preliminary description of a dynamic, context-aware web service discovery architecture aiming to fulfill the security requirements of enterprise applications.

Chapter 9

Conclusions

9.1 Summary

The methods commonly used for service discovery assume that the world is static, and therefore do not support dynamic attributes. To enable high-quality discovery, it is important to consider quality attributes. We need to find and select a service by considering its domain, operating environment, and the situation and preferences of the user, in order to provide relevant services that will fulfill a consumer's request (i.e., QoE). We refer to the above-mentioned attributes as dynamic service attributes. We studied these dynamic attributes to improve the accuracy and integrity of service discovery. By using specific examples, such as telescope and stock market services, we analyzed the crucial role of dynamic attributes for identifying quality services.

We looked at five main groups of dynamic attributes: performance, dependability, application-specific, transaction support, and security. To evaluate these attributes during service discovery, we proposed a framework that can select services in a specific domain and in different contexts. Our model, static discovery dynamic selection (SDDS), is an effective technique that measures the relevance of services based on their dynamic attributes. Our architecture includes two phases. The SD phase enables service discovery based on static attributes. Then the DS phase evaluates qualified services based on dynamic attributes. To enable automatic web service selection we employed knowledge-based databases in which the priorities and preferences are stored and updated in each domain (i.e., PIB, DIB). This information is being used during service selection to evaluate the quality of services.

We showed that these attributes play a crucial role in finding relevant services

by allowing the utilization of context to adapt service discovery to dynamic changes. To monitor the dynamic attributes, a monitor-analyze-plan-execute (MAPE-K) loop automates service quality checking. For instance, *availability*, which belongs to the *dependability* category of dynamic attributes, is evaluated differently in various contexts (i.e., context-aware). The evaluation mechanism is restored from PIB and DIB. Notably, these mechanisms can be added and altered without modifying the underlying system.

Our approach aims to ensure that consumer requests for web services are served successfully and effectively. Thus, one must narrow the web service search to only the most qualified, highest-ranked services. However, today the ranking of services is done only with regard to static attributes or by taking snapshots of current values, resulting in low-quality search results. This is quite challenging in practice because dynamic attributes can be difficult to measure; they constantly fluctuate, can be context-sensitive, and may depend on environmental factors such as network availability. In this research, we propose using histograms and an area-to-right-of-threshold function to effectively handle the fluctuation or absence of attributes' values. This permits utilizing well-established techniques for selecting web services, such as skyline and top-k. We discuss and evaluate our proposed algorithm, Dynamis, which efficiently produces high-quality dynamic web service discovery results. We evaluated our algorithm using stock market data as a proxy to web services, which allows us to perform a qualitative performance analysis of our algorithms by comparing the performance of the stocks picked with respect to the market as a whole.

Finally, web service technology provides basic infrastructure for deploying collaborative business processes. Web service security standards and protocols aim to provide secure communication and conversation between service providers and consumers. Still, for a client calling a web service it is difficult to ascertain that a particular service instance satisfies—at execution time—specific security attributes. In this dissertation, we examined how service consumers can specify the set of security attributes that a service should satisfy. We study a selection mechanism where the security objectives of the services are updated based on dynamically changing context. Also, we illustrated a mechanism to re-check security attributes when the execution context changes.

9.2 Contributions

Without the ability to distinguish available services on dynamic quality metrics, all services with the same functional attributes are considered to be equal. The contributions of our research are as follows:

- A taxonomy that describes the ubiquitous and spontaneous characteristics of web services. For the purposes of this glossary, a mapping of web service discovery requirements to service attributes is required.

- A dynamic web service discovery framework that can act on the aforementioned service attribute taxonomy during the process of publishing, searching, and locating web services. We demonstrate that ignoring any of the characteristics of a service could lead to inaccurate service discovery.

- A strategy for integrating into existing service discovery methods with our dynamic mechanism to achieve high quality service selection.

- A mechanism to verify the *consistency* and *correctness* of the proposed framework through modelling the behaviour of the system by automata. The objective of this mechanism is to evaluate whether the desired requirements can be satisfied and maintained in the proposed architecture. The behavioural properties are motivated by the fact that in a large scale service discovery system, synchronization and interaction complexity are critical issues when employing existing discovery mechanisms.

- A context-aware model that enables us to reason about the current context including service, user, and environment in order to make a *smart discovery*. The provision of context information is one of the more challenging research areas in service discovery due to the following factors: the recognition of the relevant context for a particular domain, consumer's priorities and tastes, and the dynamic nature of the environment.

- A self-adaptive web service discovery system to improve consumers' experiences in a context-agnostic manner. The self-adaptive mechanism, and in particular feedback control, is a plausible mechanism to enable dynamic service discovery. In the context of web service discovery, the primary adoption concerns are *why, what, when*, and *where* but the quality attributes need to be included in service search and find.

- A high-level security objectives of system which guide the discovery process, in order to increase the relevancy of the results. We investigated a complete framework to manage the creation, editing, validation, publication, and discovery of services considering the security attributes during the discovery process.

9.3 Future Work

Several directions we could take this work, in order to improve the quality of returned results are: a) work on a comprehensive measurement technique to rank the discovered services (i.e., factors that affect the result of service selection need to be ranked using a more robust mechanism); b) design an algorithm to support all types of progressive processing, such as user preferences and arbitrary dimensionality; c) extend SDDS with infrastructure that supports a P2P environment which will enable large-scale, decentralized data, in which each dynamic repository acts as a peer on the network.

Appendix A

Observatory web service description

A.1 WSDL example—observatory portType

In WSDL, an operation defines the method of web service. It includes method name, input, and output parameters. The ObsevatoryCheck portType defines several operations, such as checkTelescope, checkCelestialPhotography, and PhotoOrder.

```
<!--observatory port definition-->
<portType name="ObsevatoryCheckPortType">
   <operation name="checkTelescope">
     <input message="tc:checkTelescopeRequest"/>
     <output message="tc:checkTelescopeResponse"/>
   </operation>
   <operation name="checkCelestialPhotography">
     <input message="cp:checkCelestialPhotographyRequest"/>
     <output message="cp:checkCelestialPhotographyResponse"/>
   </operation>
   <operation name="PhotoOrder">
     <input message="po:PhotoOrderRequest"/>
     <output message="po:PhotoOrderResponse"/>
   </operation>
</portType>
```

A.2 WSDL example—observatory messages and parts

The abstract form of the message is modeled as an XML document in WSDL. The input and output messages in observatory WSDL are:

```
<!-- message definition-->
 <message name="checkTelescopeRequest">
   <part name="name" type="xsd:string">
</message>
<message name="checkTelescopeResponse">
   <part name="result" type="xsd:string">
</message>
<message name="checkCelestialPhotographyRequest">
   <part name="photoRq" type="xsd:string">
</message>
<message name="checkCelestialPhotographyResponse">
   <part name="photoRs" type="xsd:string">
</message>
<message name="PhotoOrderRequest">
   <part name="photoOrder" element="po:OrderRequest">
</message>
<message name="PhotoOrderResponse">
   <part name="result" element="po:OrderResponse">
</message>
```

A message is a collection of parts. Each part is specified with a name and a type. Parts can be as a SOAP header or mapped to an HTTP header. Here is what PhotoOrderRequest looks like in wire.

```
<!--message definition-->
<!--PhotoOrderRequest is an element -->
<?xml version="1.0" encoding="UTF-8"?>
<soap:Envelope
         xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
         xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance">
   <soapenv:Body>
```

```
    <ns1:PhotoOrderRequest>
    soapenv:encodingStyel: http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:ns1="http:www.ManuaKea.com/services/PhotoOrder">
           <photoOrder href="\#id0"/>
    </ns1:PhotoOrderRequest>
  </soapenv:Body>
</soap:Envelope>
```

A.3 WSDL example—observatory types

Some of the type elements in PhotoOrder require further explanation. Consider the type element in po:OrderResponse:

```
<!--type definition-->
<types>
  <xsd:schema
    <targetNameSpace="http:www.ManuaKea.com/ns/photoOrder">
    <xsd:import namespace="http:www.ManuaKea.com/ns/po"
           schemaLocation="http:www.ManuaKea.com/ns/schema/po.xsd"/>
  </xsd:schema>
</types>
```

The schema is defined in an import statement.

```
 <!-- Type Definitions -->
<types>
  <xsd:schema
    targetNamespace="http:www.ManuaKea.com/ns/po"
      <xsd:complexType name="photoOrder">
          <xsd:sequence>
             <xsd:element name= ref="po:OrderRequest"/>
    <xsd:element name="clestialName" type="xsd:string"/>
    <xsd:element name="composition" type="xsd:integer"/>
    <xsd:element name="size" type="xsd:integer"/>
           </xsd:sequence>
```

```
      </xsd:complexType>
   </xsd:schema>
</types>
```

A.4 WSDL example—observatory binding

For each portType, a binding element can specify how the requestor can invoke operations.

```
<binding name="TelescopeBinding" type="ObservatoryCheckPortType">
<soap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http" />
        <operation name="checkTelescope">
             <soap:operation
                 soapAction="http:www.ManuaKea.com/services/checkTelescope" />
          <input>
            <soap:body use="encoded" namespace="checkTelescope"
             encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
             namespace="http:www.ManuaKea.com/ns/tc/checkTelescope">
         </input>
         <output>
            <soap:body use="encoded" namespace="checkTelescope"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
            namespace="http:www.ManuaKea.com/ns/tc/checkTelescope">
         </output>
</binding>
```

A.5 WSDL example—observatory port

The port element specifies the network address of the endpoint hosting the web service

```
<port name="ObservatoryCheck"
   binding="wc:ObservatoryCheckSOAPBinding">
     <soap:address location="http:www.ManuaKea.com/services/ObservatoryCheck"/>
</port>
```

A.6 WSDL example—observatory service

A WSDL service contains a set of relevant ports.

```
<Service name="ObservatoryCheck">
   <port name="ObservatoryCheck"
     binding="wc:ObservatoryCheckSOAPBinding">
     .........
       <soap:address location="http:www.ManuaKea.com/services/ObservatoryCheck"/>
   </port>
```

A.7 Example of an OWL-S atomic process

```
<process:AtomicProcess rdf:ID="ObservatoryCheck">
  <process:hasInput>
    <process:Input ref:ID="In-StarName">
      <process:parameterType rdf:about="&xsd;#string">
    </process:Input>
  </process:hasInput>
  <process:hasInput>
    <process:Input ref:ID="In-SignInData">
      <process:parameterType rdf:resource="#SignInData">
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output ref:ID="Out-Confirmation">
      <process:parameterType rdf:resource="&xsd;#string">
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

A.8 WSDL Schema for telescope QoS attributes

This schema is based on QWSDL. [1]

```
<xs:complexType name="criteriaCheckObsevatory">
   <xs:complexContent>
   <xs:extension base="qwsdl:Extended">
     <xs:sequence>
       <xs:element name="availability" minOccurs="80" maxOccure="unbounded"/>
     </xs:sequence>
<xs:attribute name="name"  type="xs:NCname" use="required"/>
   </xs:extension>
   </xs:complexContent>
</xs::complexType>
<xs:complexType name="availabilityType">
     <sequence>
       <xs:element name="latency"  type="qwsdl:latencyType" minOccurs="unbounded"
       maxOccure="1/2"/>
     </sequence>
<xs:attribute name="name"  type="boolean"/>
</xs:complexType>
     <xs:complxType name="latency">
     <attribute name="value" type="float">
     <attribute name="unit" type="string"  fixed="msec"/>
     </xs:complexType>
```

[1] http://csrl.unt.edu/~kavi/Research/ICSEA-2012.pdf

Bibliography

[AAA+07] A. Alves, A. Arkin, S. Askary, C. Barreto, et al. Web Services Business Process Execution Language. OASIS Standard, Version 2.0, 2007. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`. Ref: 123.

[AAD11a] M. Anisetti, C. Ardagna, and E. Damiani. Fine-grained Modeling of Web Services for Test-based Security Certification. In *Proceedings of the 8th IEEE International Conference on Services Computing (SCC)*, pages 456–463. IEEE Computer Society, 2011. Ref: 125, 126, 127, 130.

[AAD+11b] M. Anisetti, C. A. Ardagna, E. Damiani, F. Frati, H. A. Müller, and A. Pahlevan. A Framework to Support Assurance-based Service Search. In *First International Symposium on Data-Driven Process Discovery and Analysis (SYMPDA), Campione d'Italia, Italy*, pages 147–162, 2011. Ref: 124.

[AAD+12] M. Anisetti, C. A. Ardagna, E. Damiani, F. Frati, H. A. Müller, and A. Pahlevan. Web Service Assurance: The Notion and the Issues. *Future Internet*, 4(1):92–109, 2012. Ref: 124.

[ABFG04] D. Austin, A. Barbir, C. Ferris, and S. Garg. Web Services Architecture Requirements, W3C Recommendation, 2004. `http://www.w3.org/TR/wsa-reqs`. Ref: 114.

[AC04] A. Armando and L. Compagna. SATMC: A SAT-based Model Checker for Security Protocols. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA)—Lecture Notes in Computer Science*, volume 3229, pages 730–733. Springer, 2004. Ref: 123.

[ACC+08] A. Armando, R. Carbone, L. Compagna, J. Cuellar, and L. Tobarra. Formal Analysis of SAML 2.0 Web Browser Single Sign-on: Breaking the SAML-based Single Sign-on for Google App. In *Proceedings of the 6th ACM workshop on Formal Methods in Security Engineering (FMSE)*, pages 1–10. ACM, 2008. Ref: 123.

[ACKM04] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer, 2004. Ref: 18.

[AJK+03] C. Abela, J. Jenkins, M. Kamil, M. Keshk, R. Masuoka, J. Ortman, J. Peer, M. Sabou, and M. Solanki. DAML-S (and OWL-S) 0.9 Draft Release, 2003. `http://www.daml.org/services/daml-s/0.9`. Ref: 30.

[AMH10] A. Al-Moayed and B. Hollunder. Quality of Service Attributes in Web Services. In *Fifth International Conference on Software Engineering Advances (ICSEA)*, pages 367–372, 2010. Ref: 40.

[And02] A. Anderson. XrML 2.0 Technical Overview Version 1.0, (ContentGuard), 2002. `http://www.xrml.org/Reference/XrMLTechnicalOverviewV1.pdf`. Ref: 119.

[And06] A. Anderson. Web Services Profile of XACML (WS-XACML) Version 1.0, (Working Draft 8), 2006. `http://www.oasis-open.org/committees/download.php/21490/xacml-3.0-profile-webservices-spec-v1.0-wd-8-en.pd`. Ref: 119.

[ANI12] Ensuring Trustworthiness and Security in Service Composition (ANIKETOS), 2010–2012. `http://aniketos.eu`. Ref: 123.

[ASO10] Advanced Security Service cERTificate for SOA (ASSERT4SOA), 2010. `http://www.assert4soa.eu`. Ref: 37, 123.

[AVA] Automated Validation of Trust and Security of Service-oriented Architectures (AVANTSSAR). `http://www.avantssar.eu`. Ref: 123.

[AVI06] Automated Validation of Internet Security Protocols and Applications (AVISPA), 2003–2006. `http://www.avispa-project.org`. Ref: 123.

[AZI+08] D. Athanasopoulos, A. V. Zarras, V. Issarny, E. Pitoura, and P. Vassiliadis. CoWSAMI: Interface-aware context gathering in ambient intelligence environments. *Pervasive and Mobile Computing*, 4(3):360–389, June 2008. Ref: 59.

[Bar05] J. E. Bardram. The Java Context Awareness Framework (JCAF)—A Service Infrastructure and Programming Framework for Context-Aware Applications. In *Proceedings of the 3rd International Conference on Pervasive Computing, Lecture Notes in Computer Science*, volume 3468, pages 98–115. Springer, 2005. Ref: 58.

[BBH12] K. Benouaret, D. Benslimane, and A. Hadjali. Selecting Skyline Web Services from Uncertain QoS. In *Proceedings of the 9th International Conference on Services Computing (SCC)*, pages 523–530. IEEE, 2012. Ref: 76.

[BCF04] E. Bertino, B. Carminat, and E. Ferrari. Merkel Tree Authentication in UDDI Registries. *International Journal of Web Services Research*, 1(2):37–57, 2004. Ref: 120.

[BDTC05] X. Bai, W. Dong, W. Tsai, and Y. Chen. WSDL-based automatic test case generation for web services testing. In *Proceedings of the International Conference on Service-Oriented System Engineering (SOSE)*, pages 207–212. IEEE Computer Society, 2005. Ref: 122.

[BGM02] N. Bruno, L. Gravano, and A. Marian. Evaluating Top-k Queries over Web-accessible Databases. In *Proceedings of the 18th International Conference on Data Engineering*, pages 216–226. IEEE Computer Society, 2002. Ref: 77.

[BKS01] S. Borzonyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *Proceedings of the 17th International Conference on Data Engineering*, volume 3, pages 421–430. IEEE, 2001. Ref: 69, 71, 78.

[BLM08] P. Bianco, G. A. Lewis, and P. Merson. Service Level Agreements in Service-Oriented Architecture Environment. *Technical Report CMU/SEI-2008-TN-021*, 2008. Ref: 63.

[Blo02] J. Bloomberg. The Rational edge e-zine for the rational community: Testing Web Services Today and Tomorrow. *IT Professional*, 2002. `http://www.p2080.co.il/go/p2080h/files/4989377677.pdf`. Ref: 122.

[BMPS09] E. Bertino, L. Martino, F. Paci, and A. Scuicciarini. *Security for Web Services and Service-Oriented Architectures*. Springer, 2009. Ref: 114, 115, 116, 119.

[BW03] W.-T. Balke and M. Wagner. Towards Personalized Selection of Web Services. In *Proceedings of the 12th International World Wide Web Conference (WWW)—Alternate Paper Tracks*, 2003. Ref: 30.

[CAP12] CAPEC Common Attack Pattern Enumeration and Classification, 2007–2012. `http://capec.mitre.org`. Ref: 116.

[CCC+04] Y. Chevalier, L. Compagna, J. Cuellar, P. Drieslma, J. Mantovani, S. Mödersheim, and L. Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS)*, pages 193–205. Austrian Computer Society, 2004. Ref: 123.

[CDP06] G. Canfora and M. Di Penta. Testing Services and Service-centric Systems: Challenges and Opportunities. *IT Professional*, 8(2):10–17, 2006. Ref: 122.

[CFV07] P. Castells, M. Fernandez, and D. Vallet. An Adaptation of the Vector-Space Model for Ontology-Based Information Retrieval. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):261–272, 2007. Ref: 29, 69.

[Cha89] C. Chatfield. *The Analysis of Time Series: An Introduction*, volume 1. Chapman and Hall, sixth edition, 1989. Ref: 78.

[CHvRR04] L. Clement, A. Hately, C. von Riegen, and T. Rogers. UDDI Version 3.0.2, UDDI Spec Technical Committee Draft (OASIS), 2004. `http://uddi.org/pubs/uddi-v3.0.2-20041019.htm`. Ref: 18, 24, 25, 44, 120.

[CK11] S. Cheshire and M. Krochmal. DNS-Based Service Discovery, Internet-Draft, Apple Inc., 2011. `http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt`. Ref: 27.

[CKP03] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Evaluating Probabilistic Queries over Imprecise Data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 551–562. ACM, 2003. Ref: 77.

[CMJ+08] T. Chau, V. Muthusamy, H. Jacobsen, E. Litani, A. Chan, and P. Coulthard. Automating SLA modelling. *Proceeding of 2008 Conference of the Centre for Advanced Studies on Collaborative Research: Meeting of Minds (CASCON)*, pages 126–143, 2008. Ref: 113.

[CMRW07] R. Chinnici, J. Moreau, A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C Recommendation, 2007. `http://www.w3.org/TR/wsdl20`. Ref: 18, 22, 28.

[Com05] L. Compagna. *SAT-based Model-Checking of Security Protocols*. PhD thesis, Università degli Studi di Genova and the University of Edinburgh, 2005. Ref: 123.

[Com10] Committee on National Security System. National Information Assurance Glossary, CNSSI No. 4009, 2010. `http://www.cnss.gov/Assets/pdf/cnssi_4009.pdf`. Ref: 114.

[Cre09] J. W. Creswell. *Research design :qualitative, quantitative, and mixed methods approaches*. Sage Publications, 3rd edition, 2009. Ref: 35.

[CvHH+01] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL Reference Description, 2001. `http://www.w3.org/TR/daml+oil-reference`. Ref: 30.

[CXP+04] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data. In *Proceedings of the 13th International Conference on Very Large Databases*, volume 30, pages 876–887. VLDB Foundation, 2004. Ref: 78.

[CYZ06] I. Chen, S. Yang, and J. Zhang. Ubiquitous Provision of Context Aware Web Services. In *International Conference on Services Computing (SCC)*, pages 60–68. IEEE, 2006. Ref: 59, 69.

[CZC08] M. Crasso, A. Zunino, and M. Campo. Easy Web Service Discovery: A Query-by-example Approach. *Science of Computer Programming*, 71(2):144–164, 2008. Ref: 29, 36, 69.

[DA10] D. A. D'Mello and V. S. Ananthanarayana. A Review of Dynamic Web Service Description and Discovery Techniques. In *First International Conference on Integrated Intelligent Computing (ICIIC)*, pages 246–251, 2010. Ref: 28.

[dAdSBdS+06] D. de Almeida, C. de Souza Baptista, E. da Silva, C. Campelo, H. de Figueiredo, and Y. Lacerda. A Context-Aware System based on Service-Oriented Architecture. In *Proceedings of the 20th International Conference on Advanced Information Networking and Applications*, volume 1. IEEE Computer Society, 2006. Ref: 59, 69.

[DAE09] E. Damiani, C. Ardagna, and N. El Ioini. *Open Source Systems Security Certification*. Springer, 2009. Ref: 122.

[dAH01] L. de Alfaro and T. A. Henzinger. Interface Automata. *ACM SIGSOFT Software Engineering Notes*, 26(5):109–120, 2001. Ref: 45.

[DESS09] E. Damiani, N. El Ioini, A. Sillitti, and G. Succi. WS-Certificate. In *Proceedings of the World Conference on Services*, volume 3395, pages 637–644. IEEE Computer Society, 2009. Ref: 122, 127, 128.

[DHM+04] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Similarity Search for Web Services. In *Proceedings of the 13th International Conference on Very Large Databases*, volume 30, pages 372–383. VLDB Foundation, 2004. Ref: 29.

[DKP+09] D. Davis, A. Karmarkar, G. Pilz, S. Winkler, and U. Yalçinalp. Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2, OASIS Standard, 2009. `http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.pdf`. Ref: 118.

[DM09] E. Damiani and A. Maña. Toward WS-Certificate. In *Proceedings of the ACM Workshop on Secure Web Services (SWS)*, pages 1–2. ACM, 2009. Ref: 112.

[Don07] W. Dong. QoS Driven Service Discovery Method Based on Extended UDDI. In *Third International Conference on Natural Computation (ICNC)*, volume 5, pages 317–324, 2007. Ref: 40.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2 (IETF RFC 5246), 2008. `http://tools.ietf.org/html/rfc5246`. Ref: 117.

[DY06] W. Dong and H. Yu. Web Service Testing Method Based on Fault-coverage. In *Proceedings of the 10th International Enterprise Distributed Object Computing Conference Workshops (EDOCW)*, pages 43–50. IEEE Computer Society, 2006. Ref: 122.

[EHMR10] J. El Hadad, M. Manouvrier, and M. Rukoz. TQoS: Transactional and QoS-Aware Selection Algorithm for Automatic Web Service Composition. *Transactions on Services Computing*, 3:73–85, 2010. Ref: 36, 75.

[ERS+11] D. Eastlake, J. Reagle, D. Solo, F. Hirsch, and T. Roessler. XML Signature Syntax and Processing, second edition (W3C Recommendation), 2011. `http://www.w3.org/TR/xmldsig-core`. Ref: 117.

[FK05] J. Fan and S. Kambhampati. A Snapshot of Public Web Services. *SIGMOD Record*, 34(1):24–32, 2005. Ref: 18, 33, 57.

[FKK11] A. Freier, P. Karlton, and P. Kocher. The Secure Sockets Layer (SSL) Protocol Version 3.0 (IETF RFC 6101), 2011. `http://tools.ietf.org/html/rfc6101`. Ref: 117.

[FLN01] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, pages 102–113. ACM, 2001. Ref: 69, 71, 77, 83.

[Fok07] W. J. Fokkink. *Introduction to Process Algebra*. Springer-Verlag, second edition, 2007. Ref: 46.

[FP12] A. Fox and D. Patterson. *Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing*. Strawberry Canyon LLC, beta edition, 2012. `http://beta.saasbook.info`. Ref: 7, 8.

[fSITS07] F. I. for Secure Information Technology SIT. Simple Homomorphism Verification Tool (SHVT) Manual, 2007. `http://publica.fraunhofer.de/documents/N-47349.html`. Ref: 123.

[FTdV06] L. Frantzen, J. Tretmans, and R. de Vries. Towards Model-based Testing of Web Services. In *Proceedings of the International Workshop on Web Services—Modeling and Testing (WS-MaTe)*, pages 67–82, 2006. Ref: 126.

[FTW04] L. Frantzen, J. Tretmans, and T. Willemse. Test Generation Based on Symbolic Specifications. In *Proceedings of the 4th International Workshop on Formal Approaches to Software Testing (FATES)*, pages 1–15. Springer-Verlag, 2004. Ref: 126.

[GAHL00] P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science and Engineering*, 15:277–290, 2000. Ref: 122.

[Gan] S. Gandhi. A Service-Oriented Approach to B2B Integration using Web Services. `http://www.dreamscape.co.in/download/B2B_Integration_WebServices_WhitePaper.pdf`. Ref: 21.

[GBK00] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing Multi-feature Queries for Image Databases. In *Proceedings of the 26th International Conference on Very Large Databases*, pages 419–428. VLDB Foundation, 2000. Ref: 77.

[GDD+05] S. Graham, G. Daniels, D. Davis, Y. Nakamura, S. Simeonov, P. Brittenham, P. Fremantle, D. Koenig, and C. Zentner. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Sams Publishing, 2005. Ref: 17, 19.

[GHH+02] B. Galbraith, W. Hankinson, A. Hiotis, M. Janakiraman, D. Prasad, R. Trivedi, and D. Whitney. *Professional Web Services Security*. Wrox Press, 2002. Ref: 111.

[GHH+11] B. Galbraith, W. Hankinson, A. Hiotis, M. Janakiraman, D. Prasad, R. Trivedi, and D. Whitney. Securing Web Services for Army SOA. *Software Engineering Institute*, 2011. `http://www.sei.cmu.edu/solutions/softwaredev/securing-web-services.cfm`. Ref: 112.

[GOR02] S. Gürgens, P. Ochsenschläger, and C. Rudolph. Role Based Specification and Security Analysis of Cryptographic Protocols Using Asynchronous Product Automata. In *Proceedings of IEEE International Workshop on Trust and Privacy in Digital Business (DEXA)*, pages 473–482. IEEE Computer Society, 2002. Ref: 123.

[GP99] E. Guttman and C. Perkins. Service Location Protocol (IETF RFC 2608), 1999. `http://tools.ietf.org/html/rfc2608`. Ref: 27.

[GPZ05] T. Gu, H. K. Pung, and D. Q. Zhang. A Service-Oriented Middleware for Building Context-Aware Services. *Journal of Network and Computer Applications*, 28(1):1–18, 2005. Ref: 59.

[GR04] S. Gürgens and C. Rudolph. *Formal Aspects of Security*, volume 2629, chapter Security Analysis of (Un-)Fair Non-repudiation Protocols, pages 97–114. Springer-Verlag, 2004. Ref: 123.

[GRQ07] J. C. Goodwin, D. J. Russomanno, and J. Qualls. Survey of Semantic Extensions to UDDI: Implications for Sensor Services. In *Proceedings of the International Conference on Semantic Web & Web Services (SWWS)*, pages 16–22, 2007. Ref: 40.

[GRS+07] S. Gürgens, C. Rudolph, D. Scheuermann, M. Atts, and R. Plaga. Security Evaluation of Scenarios based on the TCG's TPM Specification. In *Proceedings of the European Symposium on Research in Computer (ESORICS)*, pages 438–453, 2007. Ref: 123.

[HDWX08] J. Huang, D. Ding, G. Wang, and J. Xin. Tuning the Cardinality of Skyline. In Y. Ishikawa, J. He, G. Xu, Y. Shi, G. Huang,

C. Pang, Q. Zhang, and G. Wang, editors, *Advanced Web and Network Technologies and Applications*, pages 220–231. Springer-Verlag, 2008. Ref: 78, 79, 98.

[Her02] D. Herrmann. *Using the Common Criteria for IT Security Evaluation.* Auerbach Publications, 2002. Ref: 122.

[HIMB05] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam. Middleware for Distributed Context-Aware Systems. *OTM Confederated International Conferences*, pages 846–863, 2005. Ref: 36.

[HJSY08] B. Han, W. Jia, J. Shen, and M.-C. Yuen. Context-Awareness in Mobile Web Services. *Parallel and Distributed Processing and Applications*, pages 519–528, 2008. Ref: 36.

[HKK06] J. Han, R. Kowalczyk, and K. Khan. Security-oriented Service Composition and Evolution. In *Proceedings of the 13th Asia Pacific Software Engineering Conference (APSEC)*, pages 71–78. IEEE Computer Society, 2006. Ref: 124.

[HLOT06] Z. Huang, H. Lu, B. Ooi, and A. Tung. Continuous Skyline Queries for Moving Objects. *IEEE Transactions on Knowledge and Data Engineering*, 18:1645–1658, 2006. Ref: 78, 79.

[HM07] S. Hanna and M. Munro. An Approach for Specification-based Test Case Generation for Web Services. In *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications (AICCSA)*, pages 16–23. IEEE Computer Society, 2007. Ref: 122.

[HM08] M. C. Huebscher and J. A. McCann. A Survey of Autonomic Computing Degrees, Models, and Applications. *ACM Computing Surveys*, 40:1–28, 2008. Ref: 62.

[HZ07] Y. Hao and Y. Zhang. Web Services Discovery Based on Schema Matching. In *Proceedings of the 30th Australasian conference on Computer science*, volume 62, pages 107–113, 2007. Ref: 86.

[IBS08] I. F. Ilyas, G. Beskales, and M. A. Soliman. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Computing Surveys*, 40:1–58, 2008. Ref: 71, 77.

[IC02] IBM and M. Corporation. Security in Web Services World: A proposed Architecture and Roadmap, 2002. `http://msdn.microsoft.com/en-us/library/ms977312.aspx`. Ref: 117.

[IDS02] T. Imamura, B. Dillaway, and E. Simon. XML Encryption Syntax and Processing (W3C Recommendation), 2002. `http://www.w3.org/TR/xmlenc-core`. Ref: 117.

[JÖ2] J. Jürjens. UMLsec: Extending UML for Secure Systems Development. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML)*, pages 412–425. Springer-Verlag, 2002. Ref: 123.

[Jas11] Y. Jasemian. Sensing of Vital Signs and Transmission Using Wireless Networks. *In Clinical Technologies: Concepts, Methodologies, Tools and Applications*, pages 717–743, 2011. Ref: 32, 34.

[JDS09] M. Jokhio, G. Dobbie, and J. Sun. Towards Specification Based Testing for Semantic Web Services. In *Proceedings of the 20th Australian Software Engineering Conference (ASWEC)*, pages 54–63. IEEE Computer Society, 2009. Ref: 122.

[JF05] R. Jurca and B. Faltings. Reputation-Based Service Level Agreements for Web Services. In *International Conference on Service-Oriented Computing (ICSOC), Lecture Notes in Computer Science*, volume 3826, pages 396–409. Springer, 2005. Ref: 121.

[JK02] K. Järvelin and J. Kekäläinen. Cumulated Gain-based Evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, October 2002. Ref: 87.

[KC03] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003. Ref: 60.

[KCO05] H. Kim, Y.-J. Cho, and S.-R. Oh. CAMUS: a Middleware Supporting Context-Aware Services for Network-based Robots. In *IEEE Workshop on Advanced Robotics and its Social Impacts*, pages 237–242, 2005. Ref: 18, 33, 59.

[KD09] K. Kyriakos and D. Dimitris. Requirements for QoS-Based Web Service Description and Discovery. *IEEE Transactions on Services Computing*, 2:320–337, 2009. Ref: 75.

[Kis11] R. Kissel. Glossary of Key Information Security Terms NIST IR 7298, 2011. `http://csrc.nist.gov/publications/nistir/ir7298-rev1/nistir-7298-revision1.pdf`. Ref: 114.

[KKK+06] C. Keum, S. Kang, I. Ko, J. Baik, and Y. Choi. Generating Test Cases for Web Services using Extended Finite State Machine. In *Proceedings of the 18th International Conference on Testing of Communicating Systems (TestCom), Lecture Notes in Computer Science*, volume 3964, pages 103–117. Springer, 2006. Ref: 126.

[KKKR05] G. Kramler, E. Kapsammer, G. Kappel, and W. Retschitzegger. Towards using UML2 for Modelling Web Service Collaboration Protocols. *Interoperability of Enterprise Software and Applications*, 15:227–238, 2005. Ref: 123.

[KKR08] U. Küster and B. König-Ries. Evaluating Semantic Web service Matchmaking Effectiveness Based on Graded Relevance. In *Proceedings of the 2nd International Workshop SMR on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC)*, volume 416. CEUR-WS.org, 2008. Ref: 36, 86.

[KKR09] U. Küster and B. König-Ries. Relevance Judgments for Web Services Retrieval—A Methodology and Test Collection for SWS Discovery Evaluation. In *Seventh European Conference on Web Services (ECOWS)*, pages 17–26. IEEE, 2009. Ref: 36, 86, 88.

[KP09] K. Kritikos and D. Plexousakis. Requirements for QoS-based Web Service Description and Discovery. *IEEE Transactions on Service Computing*, 2(4):320–337, 2009. Ref: 11, 23, 35, 45, 57.

[KR04] S. M. Kim and M. C. Rosu. A survey of public web services. *E-Commerce and Web Technologies*, pages 96–105, 2004. Ref: 17, 57, 124.

[LBD+02] Lodderstedt, Basin, Doser, et al. SecureUML: A UML-based Modeling Language for Model-driven Security. In *Proceedings of the 5th International Conference on the Unified Modeling Language (UML)*, pages 426–441. Springer-Verlag, 2002. Ref: 123.

[LCS97] D. L. Lee, H. Chuang, and K. E. Seamons. Document Ranking and the Vector Space Model. *IEEE Software*, 14(2):67–75, 1997. Ref: 29, 69.

[LH05] S. Lipner and M. Howard. *The Trustworthy Computing Security Development Lifecycle*. Microsoft Corporation, 2005. `http://msdn.microsoft.com/en-us/library/ms995349.aspx`. Ref: 115.

[LJL+03] K. Lee, J. Jeon, W. Lee, S.-H. Jeong, and S.-W. Park. QoS for Web Services: Requirements and Possible Approaches, W3C Working Group, 2003. `http://www.w3c.or.kr/kr-office/TR/2003/ws-qos`. Ref: 11, 35.

[LKYL07] D. Lee, J. Kwon, S. Yang, and S. Lee. Improvement of the Recall and the Precision for Semantic Web Services Search. In *Proceedings of the 6th IEEE/ACIS International Conference on Computer and Information Science (ICIS)*, pages 763–768, 2007. Ref: 86.

[LN09] M. Lung and Y. Nikos. Multi-dimensional Top-k Dominating Queries. *VLDB Journal*, 18:695–718, 2009. Ref: 79.

[LNZ04] Y. Liu, A. Ngu, and L. Zeng. QoS Computation and Policing in Dynamic Web Service Selection. *Proceedings of the 13th International Conference on the World Wide Web (WWW)*, pages 66–73, 2004. Ref: 56, 63, 64.

[LRMD08] P. Leitner, F. Rosenberg, A. Michlmayr, and S. Dustdar. Towards a Flexible Mediation Framework for Dynamic Service Invocations. In *Proceedings of the 6th European Conference on Web Services (ECOWS)*, pages 45–59. IEEE, 2008. Ref: 57, 76.

[LW07] N. W. Lo and C.-H. Wang. Web Service QoS Evaluation and Service Selection Frameworks: A Proxy-oriented Approach. In *TENCON, IEEE Region 10 Conference*, pages 1–5, Taipei, 2007. Ref: 39.

[LZ09] T. Liu and G. S. Zeng. Adaptation of Mismatching Services Based on Labelled Interface Automata. In *Proceeding of 5th International Conference on Semantics, Knowledge and Grid (SKG)*, pages 326–329, 2009. Ref: 45.

[Mal06] V. Malleret. Value Creation through Service Offers. *European Management Journal*, 24(1):106–116, 2006. Ref: 9, 10.

[Mao09] C. Mao. Towards a Hierarchical Testing and Evaluation Strategy for Web Services System. In *Proceedings of the 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*, pages 245–252. IEEE Computer Society, 2009. Ref: 122.

[Mer89] R. C. Merkle. A Certified Digital Signature. In *Proceeding of Advances in Cryptology (CRYPTO)*, pages 218–238. Springer-Verlag, 1989. Ref: 120.

[Mic05] Microsoft. *The STRIDE Threat Model*. Microsoft Corporation, 2005. `http://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx`. Ref: 115.

[Min08] A. Mintchev. Interoperability among Service Registry Implementations: Is UDDI Standard Enough? *IEEE International Conference on Web Services (ICWS)*, pages 724–731, 2008. Ref: 40.

[MMD+03] J. D. Meier, A. Mackman, M. Dunner, S. Vasireddy, R. Escamilla, and A. Murukan. *Improving Web Application Security: Threats and Countermeasures*. Microsoft Corporation, 2003. `http://msdn.microsoft.com/en-us/library/ff648636.aspx`. Ref: 115.

[MMW05] J. D. Meier, A. Mackman, and B. Wastell. *Threat Modeling Web Applications*. Microsoft Corporation, 2005. `http://msdn.microsoft.com/en-us/library/ff648006.aspx`. Ref: 115.

[MPG07] M. Morse, J. Patel, and W. Grosky. Efficient Continuous Skyline Computation. *Information Sciences*, 177:3411–3437, 2007. Ref: 80.

[MRLD10] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. End-to-End Support for QoS-Aware Service Selection, Binding, and Mediation in VRESCo. *IEEE Transactions on Services Computing*, 3(3):193–205, 2010. Ref: 35, 57, 76.

[MRS08] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. Ref: 29.

[MSZ01] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16:46–53, 2001. Ref: 2.

[MvH04] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language Overview, W3C Recommendation, 2004. `http://www.w3.org/TR/2004/REC-owl-features-20040210`. Ref: 1.

[NGG+07] A. Nadalin, M. Goodner, M. Gudgin, A. Barbir, and H. Granqvist. WS-SecureConversation 1.3. OASIS, 2007. `http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.3/ws-secureconversation.html`. Ref: 118, 125.

[NKMHB06] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker. Web Services Security: SOAP Message Security 1.1. OASIS, 2006. `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`. Ref: 118, 125.

[NR99] S. Nepal and M. V. Ramakrishna. Query Processing Issues in Image (Multimedia) Databases. In *Proceedings of the 15th International Conference on Data Engineering*, pages 22–29. IEEE, 1999. Ref: 77.

[NS08] S. Noikajana and T. Suwannasart. Web Service Test Case Generation Based on Decision Table. In *Proceedings of International Conference on Quality Software (QSIC)*, pages 321–326. IEEE Computer Society, 2008. Ref: 122.

[O'S06] J. J. O'Sullivan. *Towards a Precise Understanding of Service Properties*. PhD thesis, Queensland University of Technology, 2006. `http://eprints.qut.edu.au/16503`. Ref: 113.

[OVS+06] P.-O. Osland, B. Viken, F. Solsvik, G. Nygreen, J. Wedvik, and S. E. Myklbust. Enabling context-aware applications. *Proceedings of ICIN2006: Convergence in Services, Media and Networks*, 2006. Ref: 36.

[PCM10] A. Pahlevan, M. Cheng, and H. A. Müller. A Dynamic Framework for Quality Web Service Discovery. In *Proceedings of the 4th International Workshop on Maintenance and Evaluation of Service-Oriented Systems MESOA, Workshop at 26th IEEE International Conference on Software Maintenance (ICSM)*, pages 73–89. SEI Press, 2010. Ref: 41, 76.

[PCTM11] A. Pahlevan, S. Chester, A. Thomo, and H. A. Müller. On Supporting Dynamic Web Service Selection with Histogramming. In *Proceedings of International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems*, pages 1–8. IEEE Computer Society, 2011. Ref: 76.

[PDTM12] A. Pahlevan, J.-L. Duprat, A. Thomo, and H. A. Müller. DYNAMIS: Effective Context-Aware Web Service Selection Using Dynamic Attributes. *Transactions on Services Computing*, Submitted, 2012. Ref: 69, 88.

[PKPS02] M. Paolucci, T. Kawamura, T. R. Payne, and K. P. Sycara. Importing the Semantic Web in UDDI. In *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, pages 225–236. Springer-Verlag, 2002. Ref: 30, 34.

[PM09] A. Pahlevan and H. A. Müller. Static-Discovery Dynamic-Selection (SDDS) Approach to Web Service Discovery. In *Proceedings of the 3rd International Workshop on Service Intelligence and Computing, World Conference on Services*, pages 769–772. IEEE Computer Society, 2009. Ref: 40, 41, 76.

[PM10] A. Pahlevan and H. A. Müller. Self-Adaptive Management of Web Service Discovery. In *Proceedings of the PhD Symposium at the 8th European Conference on Web Services (ECOWS)*, pages 21–24. IEEE Computer Society, 2010. Ref: 76.

[PTC07] G. N. Prezerakos, N. D. Tselikas, and G. Cortese. Model-driven Composition of Context-aware Web Services Using ContextUML and Aspects. In *International Conference on Web Services (ICWS)*, pages 320–329. IEEE Computer Society, 2007. Ref: 36.

[PTFS05] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems*, 30:41–82, 2005. Ref: 79.

[Ran03] S. Ran. A Model for Web Services Discovery with QoS. *SIGecom Exchanges*, 4:1–10, 2003. Ref: 124.

[RHPM06] N. Ragouzis, J. Hughes, R. Philpott, and E. Maler. Security Assertion Markup Language (SAML) Version 2.0 Technical Overview (Working Draft-10), 2006. `http://www.oasis-open.org/committees/download.php/20645`. Ref: 118.

[Ris10] E. Rissanen. eXtensible Access Control Markup Language (XACML) Version 3.0, (OASIS Standard), 2010. `http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cs-01-en.pdf`. Ref: 119.

[RJB04] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley Professional, 2004. Ref: 123.

[RK09] T. Ruokolainen and L. Kutvonen. Managing Non-Functional Properties of Inter-enterprise Business Service Delivery. *International Journal of Web Services Research, Service Oriented Computing (ICSOC) Workshops, Lecture Notes in Computer Science*, 4907(2):1–12, 2009. Ref: 113.

[Sar08] T. Saracevic. Effects of Inconsistent Relevance Judgments on Information Retrieval Test Results: A Historical Perspective. *Library Trends*, 56(4):763–783, 2008. Ref: 86.

[Sch99] B. Schneier. Attack Trees. *Dr. Dobb's Journal*, 24(12):21–29, 1999. Ref: 116.

[Sch02] J. C. Schlimmer. Web Services Description Requirements (W3C Working Draft), 2002. `http://www.w3.org/TR/ws-desc-reqs`. Ref: 1.

[SDHS05] M. Serhani, R. Dssouli, A. Hafid, and H. Sahraoui. A QoS broker based architecture for efficient Web services selection. In *Proceedings of the International Conference on Web Services (ICWS)*, pages 113–120. IEEE Computer Society, 2005. Ref: 124.

[SER+10] M. Sama, S. Elbaum, F. Raimondi, D. S. Rosenblum, and Z. Wang. Context-Aware Adaptive Applications: Fault Patterns and their Automated Identification. *IEEE Transactions on Software Engineering*, 36(5):644–661, 2010. Ref: 46.

[SGS04] D. Skogan, R. Gronmo, and I. Solheim. Web Service Composition in UML. In *Proceedings of the International Enterprise Distributed Object Computing Conference (EDOC)*, volume 15, pages 47–57. IEEE Computer Society, 2004. Ref: 123.

[SL98] R. Sengupta and S. Lafortune. An Optimal Control Theory for Discrete Event Systems. *SIAM Journal on Control and Optimization*, 36:519–528, 1998. Ref: 55.

[SMJ00] R. Strum, W. Morris, and M. Jander. *Foundation of Service Level Management*. SAMS Publishing, 2000. Ref: 121.

[SOA07] SOAP Version 1.2 Specification Assertions and Test Collection (Second Edition), W3C Recommendation, 2007. `http://www.w3.org/TR/2007/REC-soap12-testcollection-20070427`. Ref: 26, 28.

[SPA12] Secure Provision and Consumption in the Internet of Services (SPaCIoS), 2010–2012. `http://www.spacios.eu`. Ref: 123.

[SPS04] N. Srinivasan, M. Paolucci, and K. P. Sycara. An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In *Semantic Web Services and Web Process Composition, 1st International Workshop (SWSWPC)*, pages 96–110, 2004. Ref: 25.

[SRE10] J. Skene, F. Raimondi, and W. Emmerich. Service-Level Agreements for Electronic Services. *IEEE Transactions on Software Engineering*, 36(2):288–304, 2010. Ref: 121.

[SSS+09] D. Skoutas, D. Sacharidis, A. Simitsis, V. Kantere, and T. Sellis. Top-k Dominant Web Services under Multi-Criteria Matching. In *Proceedings of the 12th International Conference on Extending Database Technology*, pages 898–909. ACM, 2009. Ref: 37, 71, 75.

[ST09a] M. Salehie and L. Tahvildari. Self-adaptive software. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, 2009. Ref: 57, 63.

[ST09b] A. Segev and E. Toch. Context-Based Matching and Ranking of Web Services for Composition. *IEEE Transactions on Services Computing*, 2(3):210–222, 2009. Ref: 86.

[SW05] E. Stroulia and Y. Wang. Structural and Semantic Matching for Assessing Web-Service Similarity. *International Journal of Cooperative Information Systems*, 14:407–437, 2005. Ref: 30, 34, 86.

[SWY75] G. Salton, A. Wong, and C. Yang. A Vector Space Model for Automatic Indexing. *Communications of the ACM*, 18(11):613–620, 1975. Ref: 29.

[SZW+07] L. Shao, J. Zhang, Y. Wei, J. Zhao, B. Xie, and H. Mei. Personalized QoS Prediction for Web Services via Collaborative Filtering. In *International Conference on Web Services (ICWS)*, pages 439–446. IEEE, 2007. Ref: 40.

[TAH06] V. Tsetsos, C. Anagnostopoulos, and S. Hadjiefthymiades. On the Evaluation of Semantic Web Service Matchmaking Systems. In *Proceedings of the 4th European Conference on Web Services (ECOWS)*, pages 255–264. IEEE Computer Society, 2006. Ref: 86, 87.

[TD09] H.-L. Truong and S. Dustdar. A Survey on Context-aware Web Service Systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 5(1):5–31, 2009. Ref: 36, 57, 58, 69, 75.

[TDB+08] H.-L. Truong, S. Dustdar, D. Baggio, S. Corlosquet, C. Dorn, G. Giuliani, and e. a. R. Gombotz. inContext: a Pervasive and Collaborative

Working Environment for Emerging Team Forms. *The 2008 International Symposium on Applications and the Internet (SAINT)*, pages 769–772, 2008. Ref: 35.

[TN08] R. Tabein and A. Nourollah. Dynamic Broker-based Service Selection with QoS-driven Recurrent Counter Classes. In *International Conference on Service System and Service Management*, pages 1–6. IEEE, 2008. Ref: 40.

[TPC+03] W. Tsai, R. Paul, Z. Cao, L. Yu, A. Saimi, and B. Xiao. Verification of Web Services Using an Enhanced UDDI Server. In *Proceedings of the 8th International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS)*, pages 131–138. IEEE Computer Society, 2003. Ref: 124.

[TRPA06] D. Tsesmetzis, I. Roussaki, I. Papaioannou, and M. Anagnostou. QoS Awareness Support in Web-Service Semantics. In *Advanced International Conference on Internet and Web Applications and Services (AICT-ICIW)*, page 128. IEEE, 2006. Ref: 35, 64.

[TZT08] X. Tian, D. Zhang, and Y. Tao. On Skyline with Flexible Dominance Relation. In *Proceedings of the 24th International Conference on Data Engineering*, pages 1397–1399. IEEE, 2008. Ref: 69, 71, 78.

[UPN12] UPnP Forum, 2012. `http://upnp.org/sdcps-and-certification/resources`. Ref: 27.

[US 85] US Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria (Orange Book)*. Auerbach Publications, 1985. Ref: 122.

[VL08] S. Vargo and R. Lusch. From Goods to Service(s): Divergence and Convergence of Logics. *Industrial Marketing Management*, 37(3):254–259, 2008. Ref: 11.

[VM10] N. Villegas and H. Müller. Managing Dynamic Context to Optimize Smart Interactions and Services. In M. Chignell, J. Cordy, J. Ng, and Y. Yesha, editors, *The Smart Internet, Lecture Notes in Computer Science*, volume 6400, pages 289–318. Springer, 2010. Ref: 57.

[VOH+07] A. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, P. Yendluri, T. Boubez, and U. Yalçinalp. Web Services Policy 1.5—Framework. World Wide Web Consortium (W3C), 2007. `http://www.w3.org/TR/ws-policy`. Ref: 125.

[VR88] S. Vandermerwe and J. Rada. Servitization of business: Adding value by adding services. *European Management Journal*, 6(4):314–324, 1988. Ref: 9, 11.

[VSS+05] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management, Special Issue on Universal Global Integration*, 6(1):17–39, 2005. Ref: 30, 34, 69.

[WBvB+09] K. Wac, M. Bargh, B.-J. van Beijnum, R. Bults, P. Pawar, and A. Peddemors. Power- and delay-awareness of health telemonitoring services: the mobihealth system case study. *Selected Areas in Communications, IEEE Journal on*, 27(4):525 –536, may 2009. Ref: 34.

[Weg05] M. Wegdam. AWARENESS: A project on Context AWARE mobile NEtworks and ServiceS. In *Fourteenth Mobile and Wireless Communication Summit, Lecture Notes in Computer Science*, volume 4278. Springer, 2005. Ref: 58.

[Wol94] P. Wolfgange. *Design Patterns for Object-Oriented Software Development*. Addison Wesley, 1994. Ref: 116.

[WR88] W. Wonham and P. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems (MCSS)*, 1:13–30, 1988. Ref: 55.

[WS-03] Web Services Policy Assertions Language (WS-PolicyAssertions) Version 1.1, May 2003, 2003. `http://xml.coverpages.org/ws-policyassertionsV11.pdf`. Ref: 119.

[WS-07] Web Services Policy 1.5—Attachment (W3C Recommendation, September 2007), 2007. `http://www.w3.org/TR/ws-policy-attach`. Ref: 119.

[WS03] Y. Wang and E. Stroulia. Flexible interface matching for web-service discovery. In *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE)*, pages 147–156, 2003. Ref: 29, 34, 86.

[WSS+01] A. Westerinen, J. Schnizlein, J. Strassner, et al. Terminology for Policy-Based Management (IETF RFC 3198), 2001. `http://tools.ietf.org/html/rfc3198`. Ref: 119.

[ZBD+03] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality Driven Web Services Composition. In *Proceedings of the 12th International Conference on World Wide Web*, pages 411–421. ACM, 2003. Ref: 35, 64.