Context-aware, Ontology based, Semantic Service Discovery

Tom Broens

Thesis for a Master of Science degree in Telematics from the University of Twente, Enschede, The Netherlands

Graduation Committee:

University of Twente:
Dr. ir. M.J. van Sinderen
P. Dockhorn Costa, MSc.

Telematica instituut:
Dr. ir. J.W. Koolwaaij
S.V. Pokraev, MSc.

Enschede, The Netherlands
July, 2004
Abstract

The internet is evolving from a data repository to a more complex service provisioning system. Besides the upcoming of enabling technologies for internet service provisioning like web services, also business to business integration can be seen as a driving force for this evolution. Therefore, a growing amount of electronic services is deployed on the internet.

Before services can be used, they first have to be found. Service discovery is a mechanism for finding services. Current mechanisms are syntactic methods (i.e. they match service requests and descriptions of service offerings based on keywords) which often lead to a poor quality result. This thesis focuses on how the quality of the service discovery result can be improved. Service discovery quality denotes the extent to which a returned match is relevant for the user.

This thesis consists of two parts. In the first part, a problem analysis on service discovery is presented. The second part uses knowledge from the problem analysis to create a new service discovery approach. Key aspects of this new approach are context-awareness, ontologies and semantic reasoning. Finally, this new approach is implemented in a prototype to be able to evaluate the gained improvement in discovery quality.
# Table of contents

1 Introduction ................................................................................................................. 9
1.1 Motivation .................................................................................................................. 9
1.2 Objectives .................................................................................................................. 11
1.3 Research questions ................................................................................................. 11
1.4 Approach .................................................................................................................. 11
1.5 Structure .................................................................................................................. 13

## Part I: Problem analysis

2 Service discovery ........................................................................................................ 15
  2.1 Introduction to E-Services ....................................................................................... 15
  2.2 Background on service discovery .......................................................................... 16
  2.3 Processes in service discovery .............................................................................. 17
  2.4 Service discovery and information retrieval ......................................................... 18
  2.5 Current service discovery mechanisms ............................................................... 19
  2.6 Conclusions ........................................................................................................... 24

3 Context-aware service discovery ..................................................................................... 27
  3.1 Context and Context-awareness ............................................................................. 27
  3.2 Context-aware services .......................................................................................... 28
  3.3 Context-aware service discovery .......................................................................... 29
  3.4 Nature of context ................................................................................................... 30
  3.5 Related work on CA service discovery .................................................................. 31
  3.6 Conclusions ........................................................................................................... 32

4 Ontology based service discovery ................................................................................. 33
  4.1 Introduction to ontologies ...................................................................................... 33
  4.2 Ontologies in service discovery ............................................................................ 33
  4.3 Ontology technology ............................................................................................ 35
  4.4 Related work on ontology based service discovery ............................................. 37
  4.5 Conclusions ........................................................................................................... 40

5 WASP services platform ............................................................................................... 41
  5.1 Overview of the WASP architecture ...................................................................... 41
  5.2 Service discovery in the WASP platform ............................................................. 43
  5.3 Conclusions ........................................................................................................... 44

## Part II: Approach development and evaluation

6 COSS - Service discovery approach ........................................................................... 47
  6.1 Overview of COSS ................................................................................................. 47
  6.2 Matching degree .................................................................................................... 55
  6.3 COSS matching algorithm .................................................................................... 57
  6.4 CASE: Shopping ................................................................................................... 63
  6.5 Prototype implementation ...................................................................................... 67

7 Approach evaluation .................................................................................................... 73
  7.1 COSS prototype evaluation .................................................................................. 73
  7.2 Precision and Recall evaluation .......................................................................... 78
  7.3 Discussion .............................................................................................................. 80
8  Conclusions ................................................................................................................................................81
8.1  General conclusions ............................................................................................................................81
8.2  Future research ....................................................................................................................................83

References............................................................................................................................................................85
Preface

This thesis is the final piece of research I deliver before graduating from my study Telematica at the University of Twente, the Netherlands. From December 2003 until June 2004, I researched the complex, but also very interesting, area of Service Discovery at the Telematica Instituut. With this thesis, I hope to have contributed to this still developing research area.

When I started my study Telematica in 1999, as one of the first students of this course, the student mentor told us the metaphor that our study career could be compared with a train. This train could be a steam train or a high speed intercity. The train stops at different stations and can make choices to go left or right at certain turning points. However, at some point, everybody comes at his or her final destination. I am very glad I took my changes to start with this new study. With some bumps in the track, I rolled, with this thesis, as an intercity to my final destination.

Of course, I could not have done this without the help of other people. Therefore, I want to first thank my daily supervisor, Stanislav Pokraev, for his ongoing support and for giving me the motivation to go to the limit. Furthermore, I want to thank my other supervisors: Marten van Sinderen, Johan Koolwaaij and Patricia Dockhorn Costa for their support, supervision and guiding remarks. Then, I want to thank my best friend, Joris Mulder, for his interesting comments on my work or just the “gezelligheid”. As the common known proverb says, “Behind every great man is a great woman” this is also the case with me. I want to thank my girlfriend, Miriam Ogink, for her love, unconditional support and motivating ideas. Last but certainly not least, I want to thank my family: my sisters Marloes and Sandra and their partners, Bas and Bas, and of course my parents, Adrie and Ria to which I am very thankful.

Therefore, I dedicate this work to my parents, Adrie and Ria Broens, who made it possible for me to complete my study. They gave me a solid foundation for the future for which I will be eternally grateful.

Enschede, 2nd of July 2004

Tom Broens

Dutch translation


Toen ik in 1999 begon als een van de eerste Telematica studenten vertelde de studiebegeleider ons dat onze studiecarrière te vergelijken is met een trein. Deze trein kan een stoomtrein zijn of een hoge snelheid intercity. Je doet verschillende stationnetjes aan en je kunt verschillende keuze maken bij splitsingen op het spoor. Maar, op een moment, komt iedereen aan op een eindstation. Ik ben erg blij dat ik de sprong in het diepe heb genomen door aan een nieuwe opleiding te beginnen. Het spoor bevatte hier en daar enkele hobbels maar met dit rapport rijd ik als een intercity mijn eindstation binnen.
Natuurlijk was het voor mij niet mogelijk om dit rapport te schrijven zonder de hulp van andere mensen. Daarom wil ik ten eerste mijn dagelijkse begeleider, Stanislav Pokraev, bedanken voor zijn ondersteuning en voor het geven van de motivatie om tot het uiterste te gaan. Verder wil ik ook mijn andere begeleiders bedanken: Marten van Sinderen, Johan Koolwaaij en Patricia Dockhorn Costa, voor hun ondersteuning, begeleiding en sturende opmerkingen. Dan wil ik mijn beste vriend, Joris Mulder, bedanken voor zijn interessante opmerkingen op mijn werk en gewoon de gezelligheid. Zoals het bekende gezegde, “Achter elke grote man staat een grote vrouw”, weergeeft is dit bij mij ook het geval. Ik wil mijn vriendin, Miriam Ogink, bedanken voor haar liefde, onvoorwaardelijke steun en motiverende ideeën. Ten slotte wil ik mijn familie bedanken: mijn zussen Marloes en Sandra en hun partners Bas en Bas en natuurlijk mijn ouders Adrie en Ria die ik veel dankbaarheid verschuldigd ben.

Daarom wil ik dit werk opdragen aan mijn ouders, Adrie en Ria Broens, die het voor mij mogelijk hebben gemaakt om mijn studie af te ronden. Zij hebben mij een solide fundering voor de toekomst gegeven waarvoor ik hun eeuwig dankbaar ben.

Enschede, 2 juli 2004

Tom Broens
1 Introduction

The web has evolved from solely a repository of text and images to a collection of complex and heterogeneous services [FIL03]. Web service technologies [W3C-1] have enabled service provisioning through the internet. In general, they simplify the development of business applications and improve their interoperability. Therefore, it is believed that in the future there will be many different services available on the internet [PIL02].

Web services are distributed over the internet, which makes it difficult for the user to find a service and use it. We believe that contextual information plays a crucial role in the service discovery process. By considering the user's context, more-relevant services (i.e. services that match the needs and interests of that user) can be discovered and provided. However, considering contextual information in the service discovery process introduces challenging issues that will be discussed throughout this thesis.

The remainder of this chapter presents the motivation, objectives, research questions, approach and structure of this thesis. Section 1.1 describes the motivation for this research. It identifies some issues with service discovery that stimulate the need for higher-quality service discovery mechanisms. Section 1.2 presents the objectives of this research. Section 1.3 describes the research questions. Section 1.4 presents the approach to reach the formulated objectives. Finally, section 1.5 describes the structure of this thesis.

1.1 Motivation

Electronic services are software entities that perform actions on behalf of other entities [SUL02]. We believe that in the future there will be many different services available on the internet. These services have to be found before they can be used. Service discovery mechanisms provide functionality to find these services but the quality of current service discovery mechanisms is often poor.

Most of the existing service discovery mechanisms retrieve service descriptions that contain particular keywords from the user's query. In the majority of the cases, this leads to low quality of the retrieved results. The first reason for this is that query keywords might be semantically similar but syntactically different from the terms in service descriptions, e.g. ‘buy’ and ‘purchase’ (synonyms). A second reason for this is that the query keywords might be syntactically equivalent but semantically different from the terms in the service description, e.g. ‘order’ in the sense of proper arrangement and ‘order’ in the sense of a commercial document used to request supply of something (homonyms). This ambiguity of keywords can lead to mismatches between the user query and service descriptions, which leads to a low quality discovery result.

Another problem with keyword-based service discovery approaches is that they cannot completely capture the semantics of the user's query because they do not consider the relations between the keywords (e.g. if the query is “order food”, the relation between these keywords could indicate a need for a restaurant). An envisioned approach to overcome these limitations is to use ontology-based service discovery. In this approach, ontologies are used for classification of the services based on their properties. This enables retrieval based on service types rather than keywords.

Another drawback of the existing service discovery approaches is that the query-service matching score is calculated taking into account only the keywords from the user's query and the terms in
the service descriptions. Thus, regardless of the context of the user and the context of the service providers, the same list of results is returned in response to a query. By definition, context is a situation of an entity (person, place or object) that is relevant to the interaction between a user and an application [DEY00]. Therefore, considering the context in the query-service matching process can improve the quality of the retrieved results. However, contextual information is highly interrelated and has many alternative representations [POK03-1] that make it difficult to interpret and use. Again, the use of ontologies to specify the interrelations among context entities can ensure common, unambiguous representation of these entities.

An appealing future scenario of context-aware service discovery is illustrated in the following context-aware personal car system (see Figure 1). A man is driving in his car on the freeway and he tells his context-aware personal car system that he is hungry.

![Figure 1: Personal car system](image)

The driver (user) has some goal that at the end has to lead to service provisioning by a service provider. This goal has to be acquired and specified in a service request (e.g. requesting a food service providing tasty food in the proximity of where the user is driving at this moment). The service discovery mechanism receives this request from the user and starts discovering advertised services that can fulfill this request.

In traditional service discovery mechanisms, the discovery process matches advertised services with keywords from the user request. The request is full with ambiguous keywords (e.g. a service can advertise with ‘fast food service’ or ‘restaurant’ instead of ‘food service’ while meaning that it sells food). This leads to low quality matches. Furthermore, in traditional non-context-aware service discovery mechanism, contextual information (e.g. time, ‘tasty’ -> food advice, ‘nearby’ -> location) could not be used in the service discovery process because this information would simply not be present. As a result, the outcome of the discovery process is always the same while the environment of the user may require different results (e.g. some restaurants may be closed). By using contextual information, the quality of the returned services is enhanced. The returned services are better tailored to the needs of the user and less non-relevant services are returned. In the described scenario, the result of the discovery process could be a list of opened fast food restaurants (user likes fast food) with individual restaurant descriptions, ordered by proximity with a route description and estimated time of arrival.
1.2 Objectives
The main goal of this research is to improve service discovery in context-aware services platforms. This leads to an improved approach for service discovery in context-aware services platforms.

Improving service discovery is a broad concept. Improvement could stand for enhanced performance, better scalability or increased security, etc. However, in this thesis, improvement is measured by the quality of the discovery result. This leads to sub-goals that contribute to the concept of improvement in this thesis:

- Generate a definition of what improvement on the quality of the service discovery result means.
- Develop an evaluation framework in which comparisons can be made between the improved service discovery approach and current service discovery mechanisms.

This research is performed as part of the WASP project [WAS04]. The goal of this project is to create a services platform to support mobile context-aware applications [EBB02]. Therefore, it is considered an ideal environment to apply the ideas and the concepts of this research. Some further sub-goals are defined which contribute to service discovery in the WASP platform and context-aware services platforms in general:

- Define why the use of context can improve the service discovery process.
- Investigate possibilities to describe requests, context and services such that they can be semantically matched.

1.3 Research questions
Several research questions are defined which assist in the fulfillment of the main and sub-goals presented in the previous section. An important aspect of service discovery is the matchmaking process that returns request-service matches. To be able to improve the service discovery process the notion of what a good match is has to be defined. This leads to the following research questions

1. What types of matches can be identified and when is a match acceptable?
2. What is a close match among a service request, contextual information and a service description?

Furthermore, we propose that by using contextual information the quality of the service discovery result improves. Therefore, we have to research the influence and consequences of introducing contextual information in the service discovery process. This leads to the following research question:

3. Can contextual information be used in the service discovery process to retrieve higher-quality service discovery results?

1.4 Approach
We adopt a research framework (based on [VER00]) which visualizes the approach that is taken to achieve the final goal (see Figure 2).
The first part of this research consists of a **Problem analysis** on service discovery. This part results in the necessary information to create an **approach proposal**. It identifies issues in service discovery from which the novel approach proposal can benefit from. The issues considered in this phase include:

- **Traditional Service discovery**: The current service discovery mechanisms are syntactic methods that often offer poor quality results. By researching this area we are able to identify issues that limit the quality of the result with these traditional service mechanisms (e.g. Corba naming, UDDI, etc.).

- **Context-aware service discovery**: We believe that by introducing contextual information in the service discovery process, the quality of the service discovery result may improve. Researching this area gives insight why contextual information is important for service discovery and what issues we need to consider when using contextual information in the service discovery process.

- **Ontology based service discovery**: Ontologies are a formal and explicit specification of a shared conceptualization that can be used for sharing and reasoning on knowledge [FEN00]. This approach is an envisioned approach to overcome issues with non-common vocabularies [POK03, MEN03] important for service discovery like discussed in the motivation.

- **WASP platform**: The WASP platform is a services platform for supporting context-aware applications. It is researched to investigate the possibilities to introduce the new service discovery approach.

The problem analysis is used as input for the second part if this research: **Approach development and evaluation**. It is used to create a novel **Context-aware, Ontology based, Semantic Service discovery proposal**. To be able to judge if this approach has really improved the quality of the service discovery result, some criteria and evaluation methods need to be defined. These criteria and methods are presented in the **evaluation framework**. This framework is used to evaluate the new service discovery proposal and the existing service discovery methods. This evaluation leads to some conclusions on the quality of the service discovery result. Finally, these conclusions
contribute to the final product: an improved service discovery approach for context-aware services platforms.

1.5 Structure

The remainder of this thesis is structured as follows:

Part I: Problem analysis

- *Chapter 2* introduces services and service discovery in general. It discusses traditional service discovery mechanism (e.g. UDDI) and identifies issues that may limit the quality of the discovery result.

- *Chapter 3* gives an overview of context-aware service discovery. It investigates why the introduction of context in the service discovery process may improve the quality of the discovery result.

- *Chapter 4* reports on ontology based service discovery. It introduces ontologies and investigates why and how ontologies may improve the quality of the discovery result.

- *Chapter 5* introduces the WASP platform. It gives an overview of the platform and indicates why the platform is very well suited as basis for a new service discovery mechanism.

Part II: Approach development and evaluation

- *Chapter 6* reports on our effort towards an improved service discovery approach: Context-aware, Ontology based, Semantic Service discovery approach (coined COSS). It defines the notion of matching degree as a quality measurement to rate service matches. Furthermore, the COSS matching algorithm is discussed here.

- *Chapter 7* presents an evaluation that compares COSS with current service discovery approaches.

- Finally, *chapter 8* presents some conclusions and future work.
2 Service discovery

Before a service can be used, it first has to be found. Service discovery provides important mechanisms for finding services. This chapter introduces the concepts of services and service discovery. Furthermore, it presents current service discovery mechanisms and identifies important issues that need to be considered when developing service discovery approaches.

This chapter is structured as follows: Section 2.1 introduces services and their lifecycle. Section 2.2 discusses the background of service discovery. Section 2.3 discusses the different processes (e.g. matchmaking) involved in service discovery. Section 2.4 presents the relation between service discovery and information retrieval. Section 2.5 provides an overview of existing service discovery mechanisms and finally, section 2.6 gives some concluding remarks.

2.1 Introduction to E-Services

Services are ubiquitous. In everyday life, people use services in their environment. These services can be for instance physical services (e.g. the bakery on the corner, the usage of the microwave at home), social services and electronic services (E-services). Well-known examples of this last type of services are web search engines, route planners and electronic tax statements. Our research focuses on E-services. Therefore, when services are mentioned in this thesis it is a synonym for E-services.

There are many different definitions of services [SUL02, CHA00, HUL03]. We adopt a general definition (based on [SUL02]): A service is a software entity that performs an action on behalf of another entity (see Figure 3).

![Figure 3: Service](image)

A Service is a software entity provided by a Service Provider. It performs an action (based on inputs) on behalf of a Service Requestor and provides a result (output). An example of a service is the use of a web search engine: when a user (service requestor) types a query (input) into the search engine (service provider), the search engine tries to find relevant web sites (action) and returns the findings to the user (output).

A characteristic of a service is that it has properties, e.g. the action it performs, which in- and outputs it supports and who performs the action. These types of properties are called functional properties. Furthermore, a service also has non-functional properties, e.g. quality of service, cost, performance and security. Functional and non-functional properties characterize the service. Therefore, they can be used to describe a service.

The operational lifecycle of a service consists of three successive phases: Advertisement, Discovery and Delivery. This is a generalization of the model proposed in [SUL02, KLE03]. In the advertisement phase, the service provider creates a service description based on the service properties. This description is used to advertise (i.e. enable the service for use) the service. In the
discovery phase, the requestor tries to find (i.e. manual or automatic discovery) a service that satisfies his need. When a service is found, it is provided in the delivery phase (see Figure 4).

Before the delivery phase, the service requestor and service provider are unassociated with each other (maybe even not aware of each other’s existence). The service provider creates a service and advertises it to the world. The requestor tries to find some service it desires. With the service discovery the service requestor and some service provider are associated (depicted by bold lines in figure 4) with each other (i.e. they become aware of each other’s existence) and both can participate in the delivery phase.

2.2 Background on service discovery

To use a service, it first has to be found. This is often difficult for a service requestor. Services are distributed and heterogeneous. They are offered by different providers using different operating systems and transportation technologies. Furthermore, similar services can have different properties. Which one is the best suited for the requestor’s need and how can we find these heterogeneous services (see Figure 5)? These questions are all related to the area of service discovery.

Service discovery can be defined as the act of finding services. The link between the service provider and the service requestor (as presented in Figure 4) is not that obvious. Without a priori knowledge (e.g., location of the service, IP address) it is difficult for the service requestor to use the service it needs.
The issues with static binding (i.e. a priori knowledge on the location of the service) is already recognized by Remote Procedure Calls (RPC). RPC can be seen as the foundation of the vast majority of middleware platforms [ALO03]. It was introduced in the early 1980's and provided a way to transparently call procedures located on other machines. Before an RPC can be made, first, a binding (i.e. association with the server) has to be established. With static bindings, this association is hard coded, beforehand, knowledge. This mechanism is simple and efficient but it tightly couples the server and client. In this way, flexibility is reduced in a substantial manner. Furthermore, what happens when no a priori knowledge is present? In order to overcome these issues, naming and directory servers were created to enable dynamic binding. With these mechanisms, clients are able to dynamically locate servers and execute RPC's. This was the starting point of the development of current service discovery mechanisms. These mechanisms enabled users to dynamically find services without a priori knowledge of service location and service characteristics.

2.3 Processes in service discovery

The service discovery phase takes a central place in the operational lifecycle of a service, described in section 2.1. It consists of three sub-processes: Discovery request handling, Matchmaking and Discovery result handling (see Figure 6).

**Discovery request handling**

This process performs the knowledge acquisition for the matchmaking process. It consists of retrieving the request of the service requestor and formatting it such that it can be used by the matchmaking process. The retrieved discovery request is an abstraction from the high-level service requestors’ need into machine concepts. With this step, information that cannot be handled by the system is ignored. Often this ignored information is contextual information like location of the user and the time the request is executed. An important aspect of this process is the representation of the request. The way this is done influences how accurate the matchmaking result can become.

**Matchmaking**

The matchmaking process compares service advertisements (i.e. representation provided by service providers) with a discovery request (i.e. representation of the requestors need) and tries to match them.
In the matchmaking process, reasoning about knowledge plays an important role. Two information representations, coming from different parties, service providers and service requestor, are matched. This information has to be correlated and the most accurate match has to be provided.

**Discovery result handling**

This process transfers the matchmaking result to the service requestor. With this result, the service requestor can associate with the service provider. The service delivery phase can then commence.

### 2.4 Service discovery and information retrieval

Service discovery and information retrieval are related areas. The information retrieval paradigm is about a user (physical or not) having a need for information, and a set of information objects from which this need has to be satisfied [WEI01]. This is analogous to the described situation of service discovery in the previous paragraph. The service requestor has a need for a service, and a set of advertised services from which this need has to be satisfied. Therefore, some of the concepts of information retrieval can be applied to service discovery.

In information retrieval, there are two key quality measurements, which can also be applied in service discovery:

- **Recall**: The number of relevant items retrieved, divided by the total number of relevant items in the collection. The highest value of recall is achieved when all relevant items are retrieved.

- **Precision**: The number of relevant items retrieved, divided by the total number of items retrieved. The highest value of precision is achieved when only relevant items are retrieved.

[KLE04] argues that the existing service discovery mechanisms offer low retrieval precision and recall. Therefore, existing service discovery approaches suffer often from the “zero or a million problem”. The discovery result gives no result at all or gives many irrelevant results. To improve service discovery both recall and precision rates should improve (Figure 7).

The circle with the bold line is the set with services of an optimal (100% recall, 100% precision) discovery result. Improved recall rates of a discovery method shift the inner border of the result more to the optimal border while improved precision rates shift the outer border more to the optimal border.

Four types of service retrieval approaches can be distinguished [KLE04]:

![Figure 7: Recall and Precision](image-url)

- **Figure 7: Recall and Precision**
• **Keyword-Based Retrieval**: search based on keywords from the service request. This method is highly sensitive to the ‘zero or a million’ problem because keywords are a poor method to capture the semantics of a request. Keywords can be synonyms (i.e. syntactical different words can have the same meaning) or homonyms (i.e. equal words can have different meanings) leading to low precision and recall. Furthermore, the relationship between different keywords in a request cannot be handled.

• **Table-Based Retrieval**: consists of attribute-value pairs that captures service properties (e.g. output = article name). Services and requests are both represented as tables with attribute-value pairs and then matched. Semantics are more precisely captured in this method than in keyword-based retrieval but still the problems with synonyms and homonyms exist.

• **Concept-Based Retrieval**: defines ontologies for classification of services thereby enabling retrieval on types rather than on keywords. This approach can give increased precision and recall but is difficult to develop and manage (i.e. difficult to define a consistent ontology of the world, how to combine ontologies with contradictory concepts).

• **Deductive Retrieval**: in this approach, service semantics are expressed formally using logic. Retrieval then consists of deducing which service achieves the functionality described in the query. Theoretically, this method can achieve perfect recall and precision. The problems of this method are more practical. Formal modeling of service description and service request is very hard. Furthermore, the matching process in this method (proofing) can have high complexity and therefore operate slowly.

These methods are placed in a recall/precision framework to visualize their capabilities (see Figure 8). As can be seen a blank space appears where new service discovery mechanisms could fill in the need for better service discovery mechanisms. This research focuses on creating a novel service discovery approach within a particular domain (WASP platform, see chapter five) which can be positioned somewhere in this blank space.

![Figure 8: Service retrieval types](Figure 8: Service retrieval types)

### 2.5 Current service discovery mechanisms

The previous sections introduced services and service discovery. These sections already indicated some theoretical issues that limit the quality of the service discovery result. This section elaborates on existing service discovery mechanisms and the issues that limit the quality of the service discovery result in a practical situation. Major approaches like CORBA naming/trading services and UDDI are discussed. Furthermore, upcoming approaches like OSGi and Grid computing (i.e. OSGA) are discussed.
2.5.1 CORBA naming and trading services

The Common Object Request Broker Architecture (CORBA) [OMG03, BRO01] is one of the best-known middleware specifications for creating and managing distributing object-oriented applications. It was developed in the early 1990's by the Object Management Group (OMG) that is a consortium of over 800 companies.

Introduction to CORBA

The main components of the CORBA architecture are the Object Request Broker (ORB), Corba Services and Corba facilities. Together these components support the development of distributed object-oriented applications (see Figure 9).

![CORBA overview](image)

The ORB provides basic object interaction functionality. It is used to access user-defined objects (i.e. application objects / services) in a transparent way (i.e. location and platform transparency). However, before an object can be accessed through the ORB, first its interface has to be declared. This is done using CORBA's Interface Definition Language (IDL). This definition can be compiled in stubs (i.e. client side) and skeletons (i.e. object implementation side) which are used as proxy objects when accessing remote objects.

The other main components of CORBA are CORBA facilities and CORBA services. CORBA facilities [OMG03-2] provide higher-level services needed by applications and not individual objects. Examples are financial services and information services. CORBA services [OMG03-1] are more related to individual objects. They provide functionality commonly needed by most objects. Examples of these services are enabling persistent objects, security features, naming and trading services. These final two CORBA services are related to service discovery and are therefore interesting for this research. These CORBA services will be discussed in more detail in the next paragraphs.

CORBA naming service

To invoke an operation on a remote object the client can use stub code (i.e. generated statically for a particular interface (IDL definition)) to convey the invocation to the remote object. This implies a static binding, because the stub has to be pre-compiled from a shared IDL, before invocations can executed. To overcome this issue, CORBA implemented the Dynamic Invocation Interface (DII) [OMG03]. This mechanism provides a standard interface for creating request to remote objects without using pre-compiled stub code. Although this mechanism provides more flexibility, it still presumes that the user is aware of which service it wants to use (i.e. a reference
to the remote object is needed). To discover and retrieve such object references CORBA has defined the naming service [OMG03-1].

The naming service lets the client retrieve a single object reference using an identifier that functions as a key. To be able to retrieve the reference, first a mapping between a name and the object reference has to be provided. This is called binding and can be seen as the advertisement phase of a service. This mapping is stored in a central repository (i.e. centralized discovery model) and can be accessed via a standard interface (e.g. resolve()). The naming services, matches the name provided by the service requestor with the mappings in the repository. Then, the naming service provides, if exactly matched, the reference to the object. This reference can be used in the delivery phase of the service.

**CORBA trading service**

Another mechanism to retrieve CORBA object references is the trading service [OMG03-1]. It differs from the naming service because this mechanism does not match request and service by a name but by properties the service has.

First, the service providers define the service by creating an IDL interface specification and a service offering. In the service offering, the properties of the service are described. This offering is stored in the central Trading service (i.e. central discovery model). The offerings are classified by service types. So, similar services (i.e. same interface but possibly different properties) can originate from the same service type. In this way, common terminology is used to describe and request a service. The service requestor queries the Trading service by defining which service type it wants to search and which criteria’s (i.e. which properties must be present) have to be fulfilled. The provided result (object references) may be sorted by some defined user preferences. This result can be used in the delivery phase of the service [OMG03-1].

**2.5.2 UDDI**

Universal Description, Discovery and Integration of Web Services (UDDI) [UDD03] is a well-known service discovery mechanism for web services. In 2000, it emerged as a cooperation between Ariba, IBM and Microsoft to improve integration of B2B on the internet. At this moment, the development of UDDI is performed by OASIS and versions three of the UDDI specification is released [UDD03-1]. See Figure 10 for an overview of UDDI for web services.

**Introduction to UDDI**

The main goal of UDDI is to specify a framework for describing and discovering web services [ALO03]. The core of the UDDI architecture is a central business registry (i.e. centralized discovery model) which functions as a naming and directory service. The service provider advertises his service to this central registry and the service requestor discovers services by querying the central UDDI registry and using the result to associating with the service. These steps (publishing and querying) are projected in the two major functionalities of UDDI:
• UDDI defines data structures and API’s for publishing service descriptions.
• UDDI enables the user to query the registry to look for published descriptions.
  o Enable developers to discover information on certain services to make their clients compatible with these services.
  o Enable dynamic binding (i.e. retrieve references to services of interest).

The information contained in an UDDI registry can be categorized using a metaphor of a telephone dictionary [ALO03]. The information can be categorized as:

• **White pages**: listing of organizations, contact information and services these organizations provide.
• **Yellow pages**: classifications of both companies and Web services according to standard or user defined taxonomies (i.e. tModel).
• **Green pages**: information on how a given Web service can be invoked (e.g. WSDL specification).

This classification is reflected by the four types of information encapsulated by a UDDI registry entry (see [UDD03-1, UDD00, ALO03]). These information types are formatted in XML structures. One of these information types is the technical model (tModel) which will be discussed in more detail in the next paragraph.

**Technical model (tModel)**

tModels are abstract specifications of knowledge of web service, which is advertised in the UDDI registry. It can be seen as the ‘yellow pages’ of UDDI. It classifies the business or web service using standard or user-defined taxonomies.

The main content of a tModel consists of a unique key and a pointer to the documentation of the service (see Figure 11). The documentation can reside anywhere and can be written in any available language. The documentation of the service is typically unstructured data meant for human beings and not automated systems. The human service requestor can browse the UDDI registry (UDDI entries and tModels) and can gain insight in what a certain service does and which properties and interfaces it provides.

![Figure 11: tModel](image)

A specified tModel (characterized by its unique key) can be used by multiple services. In this way, it is used to classify services. E.g. tModel 432 describes a standardized transportation interface (e.g. documentation in WSDL). Other services implementing the same interface could also refer to this unique tModel number. A client could search on tModel 432 to discover all services supporting this interface.

**Discovery process in UDDI**

UDDI defines three types of users to whom they expose their API’s: service providers that publish services, service requestors that look for services, and other registries that need to exchange information on services. There are six sets of API’s [UDD03-1, UDD00] that these parties can use. We only address the two that are relevant for this research:
• **Publisher API**: includes operations for service advertisement that can be used by service providers.

• **Inquiry API**: includes operations to find registry entries and to find details about a specific entity.

The publisher API enables service providers to advertise their created services. Service providers can add, modify and delete UDDI entries. Example operations that this API supports are: `save_business`, `save_service`, `save_binding`, `save_tModel` for creating and modifying entries and `delete_{xxx}` for deleting entries. This API is used in the Advertisement phase of the service’s lifecycle. The service provider creates a service definition (e.g. WSDL and tModel) and advertises (i.e. `save_{xxx}`) it to the registry.

The inquiry API can be used for discovering registry entries. Example operations of this API are `find_business`, `find_service`, `find_binding`, `find_tModel`. These functions return an XML document that encapsulates keys to relevant UDDI entries. With the `get_{xxx}` functions (plus document keys) the real XML information structures can be retrieved. This API is used in the Discovery phase of the service’s lifecycle. The service requestor places a discovery request (i.e. `find_{xxx}`). The UDDI registry syntactically matches this request with the advertised services and returns the result.

### 2.5.3 OSGi platform service discovery

The Open Service Gateway Initiative (OSGi) Alliance is an open forum in which major companies participate (e.g. Samsung, Siemens). Their mission is to specify, create, advance, and promote an open Service Platform for the delivery and management of multiple applications and services to all types of networked devices in home, vehicle, mobile and other environments [OSGI03]. The alliance was founded in 1999 and at this moment version three of the specification is released [OSGI03-1].

**Introduction to the OSGi platform**

The mission of the OSGi alliance is to create an open specification for the network delivery of managed services to local networks and devices. The efforts of the alliance resulted in the OSGi platform specification. This specification describes a framework for an open and common architecture for service providers, developers, software vendors, gateway operators and equipment vendors to develop, deploy and manage services in a coordinated fashion [OSGI03-1].

The OSGi specification describes much more than service discovery alone. The complete platform specification is too extensive to describe completely in this section. Furthermore, parts of the framework are out of the scope of this research. Therefore, we will focus solely on the service discovery mechanisms of the OSGi platform in this section.

The OSGi platform defines a java platform for service provisioning. The functionality of services is implemented by **bundles**. Bundles are a collection of java classes that can provide functionality to end users or other services. Services are described by their java **service interfaces**. The interface and the implementation classes together are used as a **service reference**. Furthermore, pairs of key/value can be used to specify some property of the service. The service is registered by the service provider in a central service registry, meaning that a centralized service discovery model is applied (see Figure 12).
After the services are registered in the service registry a service requestor can discovery the services by requesting service references (can be used to invoke the service). The request can be based on the name of the interface (syntactical matchmaking) or based on the defined properties using filters. By using retrieval methods on the service reference, properties of the service can be read and the requestor can determine which service it wants to use.

2.5.4 OSGA service discovery

The Open Service Grid Architecture (OSGA) [GLO03] focuses on integrating grid-computing paradigms with web services technologies. It proposes a specification for the deployment of grid services using web service technology. The latest version of the specification was released in 2003 [TUE03].

Introduction to OSGA

Grid computing is an upcoming approach for integrating dynamic resources in services. This area is too extensive and complex to describe in this section. Therefore, we refer to [FOS02] that gives a good overview. This section focuses solely on the service discovery mechanism in OSGA.

A service provider can provide a grid service and a service instance is created. The service instance has a grid service handle (i.e. reference to the service). This handle can be registered to a central registry (i.e. central service discovery method). This entry in the registry is complemented with service data. This data is formatted as XML structures containing information about the service instance. By using standard methods (e.g. FindServiceData) this information can be retrieved by the service requestor. This information is used to retrieve the correct handle and invoke the service.

2.6 Conclusions

Before service requestors can use a service, they have to find it first. Without the assistance of discovery mechanisms to find services, it is hard (if not impossible) for service requestors to use the service they need. Services are often heterogeneous and distributed and therefore finding services can be a difficult task. Current service discovery mechanisms provide functionality to help finding services for the service requestor making it easier to use services.

Two important aspects in service discovery can be distinguished which highly influence the quality of the service result. The first aspect is knowledge representation. The request of the service requestor and the description of the services are represented in a particular format. These representations are matched. Therefore, if these representations are awkwardly formatted the
quality of the discovery result can be poor. This research does not focus on how to retrieve and how to transform the users need into a usable representation. We consider this future work. The starting point is a ‘pre-defined’ representation of the service description and request.

The second aspect that can greatly influence the quality of the discovery result is reasoning about knowledge. The matchmaking process has to perform reasoning on service knowledge. It takes the request from the service requestor and the service descriptions from the service providers and tries to match them. How this matching (i.e. reasoning) is done is therefore very important for the quality of the discovery result.

This chapter presented several major and some upcoming service discovery approaches. Below they are classified using the service discovery types defined in section 2.4. This gives an impression of their capabilities.

<table>
<thead>
<tr>
<th>Method</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corba naming</td>
<td>Keyword-based</td>
</tr>
<tr>
<td>Corba trading</td>
<td>Table-based</td>
</tr>
<tr>
<td>UDDI</td>
<td>Table-based / Concept-based</td>
</tr>
<tr>
<td>OSGi</td>
<td>Table-based</td>
</tr>
<tr>
<td>OSGA</td>
<td>Table-based</td>
</tr>
</tbody>
</table>

Many of the current service discovery approaches are table-based which means that they provide limited recall and precision. This leaves opportunity for new service discovery mechanisms to improve precision and recall.

After examining the existing service discovery mechanisms some issues can be distinguished which prevent them from reaching the full potential in service discovery:

- **Contextual information**: no existing service discovery mechanisms incorporate contextual information into their functionality. This way, information (which is often present) relevant to the request is lost. We believe that by incorporating contextual information into the matchmaking process, a higher precision and recall can be achieved. This concept is elaborated in the next chapter.

- **Common understanding**: the investigated methods lack the support for creating a common understanding on services. The service requestor and service provider are different parties with disjoint knowledge on the service. Without a mechanism to coordinate the knowledge of the different parties, the request and the description can be out-of-sync (e.g. synonyms and homonyms) resulting in a poor quality discovery result.

- **Semantic matchmaking**: the described service discovery mechanisms have limited capabilities to use semantics. This limits them to syntactic matchmaking that may not always lead to the best quality of the discovery result. We believe that by performing semantic matchmaking, a higher precision and recall can be achieved. This concept is elaborated in chapter 6.

The described issues form a challenge for creating new service discovery mechanisms. When overcome, they have the potential to increase the quality of the discovery result. In the next chapter, the concept of introducing contextual information in the service discovery process, context-aware service discovery, will be discussed. It elaborates on why contextual information can improve service discovery.
3 **Context-aware service discovery**

The previous chapter discussed traditional service discovery mechanisms. Among other things, it concluded by saying that traditional service discovery mechanisms do not use contextual information. Furthermore, it stated that including contextual information could improve the quality of the service discovery result. This chapter introduces the concepts of context-aware services (CA services) and context-aware service discovery (CA service discovery) in which contextual information is used in its functionality. This chapter elaborates on what context is, why context-awareness may improve the service discovery result and which issues in CA service discovery play a role.

This chapter is structured as follows: Section 3.1 introduces context and context-awareness. Section 3.2 presents the notion of CA services and the involved parties. Section 3.3 elaborates on CA service discovery and discusses why this method is an improvement over traditional service discovery. Section 3.4 describes the nature of context. Section 3.5 gives an overview of related work on CA services and CA service discovery. Finally, section 3.6 gives some concluding remarks.

### 3.1 Context and Context-awareness

In the previous chapters, an intuitive notion of context and context-awareness is used to describe issues with service discovery. Location and time are well known examples of context. However, many other context entities, maybe not that intuitive, such as stock quotes or weather forecast exist. To give a precise definition of context and context-awareness is challenging and is done by many research areas (i.e. artificial intelligence, human-computer interaction, ubiquitous computing) [DOC03, MEN03, POK03-1]. In this thesis, context and context-awareness are described by definitions presented in [DEY00]. These definitions do not exclude particular issues but are also not too general.

In this thesis **context** is defined as: *information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to interaction between a user and an application including the user and application themselves*. Furthermore, **context-awareness** is defined as: *a property of a system that uses context to provide relevant information and/or service to the user, where relevancy depends on the user’s task.*

Historically, context has been ignored in computer science [LIE00]. The computer science field has strived for context-independence to be able to use abstractions. Many computer systems are treated as black boxes to overcome complexity and to guarantee correct functionality. By improving the computer's access to context, thereby breaking with the tradition of context-independence, the richness of communications in human-computer interactions can be improved and more useful computational services can be produced [DEY00].

Schilit [SHI94] defines three categories of context that can be further refined into sub-categories [MEN03]:

- **Computing context**: contextual information related to computational aspects of the CA system.
  - Application context: email received, websites visited, etc.
  - System context: network traffic, status of resources, bandwidth, QoS, etc.
- **User context**: context information related to the service requestor.
  - Personal context: health, mood, schedule, activity, etc.,
- Social contexts: group activity, social relationship, people nearby etc.
- **Physical context**: contextual information related to physical aspects of the CA system.
  - Physical context: location, time, etc.
  - Environmental context: weather, altitude, light, etc.
  - Informational context: stock quotes, sport scores, etc.

This list is not a systematic approach but merely gives a classification of context by means of examples.

Some of these context entities are already present in current non-CA systems [KRA03]. Consider for instance, a mobile phone. It has information on the time, the network capabilities, maybe even the user’s location, but this information is not used in service provisioning. This provides an opportunity for context-aware services to provide context to create better user-tailored services.

### 3.2 Context-aware services

In general, traditional services perform their functionality based on explicit inputs and ignore context [LIE00]. Consider a simplified traditional telephone service. A telephone call would be made based on the dialed telephone number. The result would be a connection to the recipient. Contextual information, such as whether the recipient is busy (e.g., meeting) is completely ignored.

In a context-aware telephone service, this information could be incorporated in the service functionality. This way, the call would be forwarded to the user only if he is available otherwise the voicemail is activated. Therefore, by using the status of the user (e.g., derived from its calendar) we may be able to provide more value services (see Figure 13).

To be able to provide CA service, these services need to consider also context inputs besides functional inputs. Therefore, the output of CA service does not now also depend on contextual information. Contextual inputs are provided by context providers such as a GPS, weather station, etc. Context providers form a new party in the service provisioning process (see Figure 14).
Besides context inputs, a service exhibits context itself. Consider for instance a shopping service. This can be a store located in the city center. The physical location can be stored in the service description and can be used when discovering this service. E.g., a user requests a shopping service nearby. The location of the user and the location of the shopping services are matched and the nearby services are returned. Services that exhibit context but do not require context inputs are not considered context-aware because they do not use user context to provide tailored information. However, they can be context-aware discovered (see the next section).

3.3 Context-aware service discovery

The issues that play a part when using distributed and heterogeneous services are not overcome by the introduction of CA services. Still a mechanism to find services is needed. With non-CA service discovery mechanisms context cannot be used to discovery service. Therefore, another mechanism has been introduced: context-aware service discovery.

CA service discovery can be defined as: the act of finding services thereby using context. To use a CA service not solely an association between the service requestor and service provider (two parties) need to be created (traditional service discovery) but an association between the service requestor, service provider and context provider (three parties) needs be created (see Figure 15).

When considering the lifecycle of services (see Figure 4) some changes occur with the introduction of CA services. The service provider still creates and advertises a service description. However, this service description can now also contain context inputs and context information.
on the service. The context provider defines his context provisioning service. In the discovery phase, the discovery process matches the service request with the context service descriptions and the service descriptions. In the delivery phase the service requestor, context provider and service provider are associated and the service can be invoked.

The matching process (see Figure 6) is now divided into two sub-processes. When the request is acquired and represented in the correct format, the matching process searches for context relevant to the request and the service. The second phase is the matching of services with the request and the context. Consider the example that a user wants to find open restaurants. First, the matching process finds the current time to determine which restaurants are open. Then, it finds all open restaurants and returns them.

We believe, there are two major reasons why the matching process can initiate the search for context (they can be performed simultaneously):

- **Request completion (applies to service requestor side):** should be performed when the requestor wants to find services based on contextual criteria. For instance, the requestor wants to find shopping services nearby. The matching process looks for location information of the requestor and certain services. This information is used to match the request and the service. By completing the request with contextual information, the precision of the result is improved since the request becomes information-richer and therefore results that are more relevant can be returned.

- **Input completion (applies to service provider side):** should be performed when possible services require context inputs that the service requestor does not provide. For instance, the requestor wants to find open shopping centers (services). The matching process looks for these services and concludes that the requestor does not provide the current time. It then looks for time services that can provide this contextual information. By completing missing inputs, the recall of the result is improved since the services that would be ignored when not using context, are now returned.

Therefore, theoretically, recall and precision rates improve when including context in the service discovery process. The next section presents the nature of context that influence how much context can improve service discovery.

### 3.4 Nature of context

In [POK03-1] several observations are indicated about the nature of context. These issues influence the role that context-aware service discovery can play in improving the quality of the discovery result:

- **Context information exhibits a range of temporal characteristics:** Context can be classified as static or dynamic. Static context is fixed information such as the gender of a person while dynamic context changes often and is for instance location. This provides requirement for context acquisition (e.g., dynamic context has to be often acquired).

- **Context information is imperfect:** Pervasive environments are highly dynamic which means that described information can quickly become out of date. Because of the distributed property of CA systems, high delays can occur that can result in faulty contextual information. Furthermore, derivation algorithm of context producers can produce faulty information when inferring from crude sensor inputs.

- **Context information is highly interrelated:** Several relations are evident between contextual information. For instance, speed can be derived by a time interval and distance and the openness of a store can be inferred using the current time.

- **Context has many alternative representations:** Often context is obtained from sensors. Before this context can be used it has to be processed to generate concepts which can be
used by high level applications. Different applications have different requirement leading to different representations of the context.

These observations indicate some limiting factors on how precision and recall rates can increase by introducing context in the discovery process. When the context is imperfect, the precision and recall decreases because incorrect information is used. Furthermore, alternative representations among the service requestor, providers and context providers results in mismatches and decrease precision and recall.

Some system requirements for CA service discovery mechanisms can be derived from these observations:

- Limit computational delays such that the probability of faulty context is minimized;
- Common understanding on context has to be established for correct matchmaking;
- Context descriptions should be able to encapsulate relations between contexts to use the opportunity for context derivations.

### 3.5 Related work on CA service discovery

This section discusses related work on CA service discovery. The discussed mechanisms give an insight in the issues that play a role in applied CA service discovery. Furthermore, it gives an impression of the state of the art in CA service discovery. The discussion includes the following initiatives: Cooltown project, Context Toolkit, Platform for Adaptive Applications and the CB-Sec project.

#### 3.5.1 Cooltown Project

The Cooltown project [COO03], developed by the Hewlett Packard Lab, focuses on expanding the view of physical entities to a virtual world of web content, in which people, places and things are web-present. The bottom layer of the developed Cooltown architecture consists of mechanisms for obtaining points of interest (i.e. people, places, things) via discovery mechanisms. Discovery can take place by sensing the location of the points of interest (i.e. URL or ID from which a URL can be derived) via beacons [KIN00]. Cooltown's service discovery is location aware. The location of the user enables it to retrieve points of interest. It is up to the user to determine if a certain point of interest is relevant to him.

#### 3.5.2 Context Toolkit

The context toolkit aims at facilitating the development and deployment of context-aware applications [CON03]. The discovery mechanism used in the Context Toolkit is centralized. It uses only a single discoverer (i.e. central registry). When started, all components register with the discoverer at a known network address and port. The discoverer “pings” each registered component at a pre-specified frequency to ensure that each component is functioning correctly. Applications can discovery services by using yellow and white page lookup mechanisms [DEY01].

#### 3.5.3 Platform for Adaptive Applications

This architecture for a services platform is developed at the Lancaster University, England. Its goal is to build a unified architecture that supports multiple contextual attributes coupled with a user driven adaptation control mechanism [EFS01]. Applications using this architecture are adapted by context. Therefore, the discovery mechanism focuses on locating services that provide contextual information. The architecture is mainly based on UPnP (Universal Plug and Play) discovery. Meaning, that when a context service is available it advertises itself to the platform. The platform retrieves a simple XML description of the context service that is stored in the context database. Applications are adapted by rules defined in XML that point to context services.
3.5.4 CB-SeC Project

CB-SeC is a project, developed at the University of Fribourg in Switzerland. This project develops an architecture that allows service discovery and composition to take benefits from the available context [MOS03]. It is based on mobile devices that not only consume web services, but also publish their data through web services. Service discovery is done by the Context-Based Service Discovery module (CSD). The request of the client is decomposed into basic services using UCMs (ubiquitous coordination model) socials laws. Then the broker agent (i.e. another part of the CSD) retrieves context information from the context database. This information is used to discover the requested basic services near the user. Furthermore, user and device capabilities are incorporated in the discovery.

3.6 Conclusions

Traditional service discovery mechanisms do not use context in their functionality. This raises the possibility of using context to improve service discovery. This chapter elaborated on the importance of context in service discovery and investigated why context would improve service discovery and which issues should be considered.

Adding context to service provisioning can provide better user-tailored services. By introducing context in service discovery, the request gets information richer and thereby can make the discovery result more precise (i.e., request completion, see section 3.3). Furthermore, by searching for context the user does not provide explicitly that is needed to let the service execute, more (CA) services can be returned as acceptable matches leading to better recall (i.e., input completion, see section 3.3).

The nature of context introduces some limiting factors on how well precision and recall can improve. This issue introduced some system requirements for CA service discovery mechanisms. Furthermore, some of the issues distinguished in the previous chapter still apply, and even get more important, with CA service discovery:

- **Common understanding**: by introducing context, another party is introduced in the service provisioning process: the context provider. Service provider, service requestor and now also context provider have different knowledge of the services and without mechanism to create a common understanding, poor discovery result may occur.
- **Semantic matchmaking**: context has different representations making them semantically rich. This increases the need for semantic description mechanisms and semantic reasoning.

The CA service discovery area is relatively new. The discussed existing CA discovery mechanisms (see section 3.5) are not full-grown or commercially available. The above-described issues can be recognized in these mechanisms. The next chapter introduces ontologies for creating a common understanding on services among the involved parties in service discovery.
4 Ontology based service discovery

The previous chapters discussed the importance for a common understanding among the involved parties in service discovery. Furthermore, they indicated that (context-aware) service discovery mechanisms lack this common understanding. In this chapter, we discuss ontology based service discovery. These mechanisms use ontologies as means to achieve a shared and common understanding on services.

This chapter is structured as follows: Section 4.1 introduces ontologies. It gives their background and a definition. Section 4.2 presents the place of ontologies in service discovery. It gives a motivation to use ontologies and discusses their responsibilities. Section 4.3 elaborates on the major ontology technologies (i.e. RDF, DAML+OIL, OWL). Section 4.4 discusses related work. Finally, section 4.5 gives some concluding remarks.

4.1 Introduction to ontologies

Historically, ontology is a part of metaphysics. Metaphysics is the branch of philosophy concerned with the nature of ultimate reality. It is customarily divided into ontology, which deals with the question of which fundamentally distinct sorts of entities compose the universe, and metaphysics proper, which is concerned with describing the most general traits of reality [ENC03]. The word ‘ontology’ originated in the Greek language, meaning “the study of being”.

Currently, ontologies have also been used in computer science. In the beginning, they were developed in Artificial Intelligence to facilitate knowledge sharing and reuse. More recently, they have also been used in information retrieval, electronic commerce and knowledge management. They have been developed to provide machine-processable semantics of information sources that can be communicated between entities [FEN00].

Ontologies can be defined as: “a formal, explicit definition of a shared conceptualization” [FEN00]. A ‘conceptualization’ refers to an abstract model of some phenomenon in the world that identifies the relevant concepts of that phenomenon. ‘Explicit’ means that the used concepts and their constraints should be explicitly described and ‘formal’ means that the ontology should be machine-readable.

4.2 Ontologies in service discovery

Service discovery considers multiple parties in its functionality. The service requestor needs a particular service. The service provider offers a certain service and the context provider delivers contextual information about services and service requestors. These parties are distributed and have different knowledge on the service (see Figure 16).
Service discovery deals with the dynamic process of bringing (associating) these distributed parties together. This dynamic and distributed aspect of service discovery together with the lack of common understanding makes service discovery hard.

The current (syntactic) discovery mechanisms (keyword-based) do not consider the semantics of the requestors goals, service and context. They retrieve objects with descriptions that contain particular keywords from the user’s request. In most of the cases, this leads to low recall and precision. The reason for the first is that request keywords might be semantically similar but syntactically different from the terms in object descriptions (synonyms). The reason for the second is that the query keywords might be syntactically equivalent but semantically different from the terms in the object description (homonyms). Another problem of the current discovery mechanisms is that they cannot completely capture the semantics of the user’s request because they do not consider the relations between the keywords. Furthermore, in CA service discovery the contextual information is highly interrelated and has many alternative representations [POK03-1] which makes them difficult to interpret and use. One possible solution for these described problems is to use ontologies (concept-based). In this approach, ontologies are used for classification of the objects based on their properties. This enables retrieval based on object types rather than keywords. Furthermore, they can specify the interrelations among context entities and ensure common, unambiguous representation of these entities (see Figure 17).

Figure 17 depicts the ideal position of ontologies in service discovery. All the involved parties, including the discovery process, commit to a common ontology. This way, a common understanding on services is established leading to better quality of the discovery result.
[GUI03] indicates two major concerns when building an ontology-based application. These have to be considered when creating an ontology based service discovery mechanism:

- **Language**: ontologies must be encoded in some language before they can be used in an application. The languages must be able to encode the concepts in the particular problem domain and they must provide enough expressiveness to deal with the relationships between concepts. Already many initiatives have started to encode ontologies. Some of these initiatives are addressed in section 4.3.

- **Environment**: generating, analyzing, modifying and maintaining ontologies are important aspects for an ontology-based application. Who creates the ontology? What is in the ontology and how to keep the ontology consistent are relevant questions in this area. In service discovery, these aspects could be fulfilled by a service registry/platform. As a central entity, it is a good place to create, maintain an ontology and to enforce commitment to the ontology.

### 4.3 Ontology technology

This section presents the major enabling technologies (i.e. ontology languages) for ontology-based applications. The discussed technologies are RDF/RDFS, DAML+OIL, and OWL.

#### 4.3.1 RDF/RDF-S

The Resource Description Framework (RDF) specification provides a lightweight ontology system to support the exchange of knowledge on the Web [W3C-4]. It is developed by the W3C consortium and is one of the building blocks of the so-called “Semantic Web” [FEN03].

RDF is an extension of XML. While XML only provides mechanisms to create structured data with elements and attributes, RDF also describes the data. It can be used to describe statements like “Tom’s gender is male” (see Example 1).

```xml
<?XML version="1.0"?>
<rdf:RDF
// general RDF namespace
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
// namespace points to RDF schema (RDF-S)
xmns:example=http://www.example.com/exampleschema>
// Described resource is “Tom”
<rdf:Description rdf:about="Tom">
  //“Tom” has predicate “Gender” which has value “Male”
  <example:gender>Male</example:gender>
</rdf:Description>
</rdf:RDF>
```

**Example 1: XML and RDF document**

In the described example, the resource that is described in RDF is “Tom”. A predicate of Tom is that he has a gender. The value of his gender is “Male”. These three elements (resource, predicate, object (i.e. value)) form the basic building blocks of an RDF document. By pointing to RDF schemas, the semantics of “gender” is defined. The namespace mechanism (i.e. xmlns, URI) of XML is used to refer to the RDF schemas. RDF schema (RDF-S) is the schema specification of RDF. While RDF is used to describe data, RDF-S is a domain-neutral way of describing the
metadata that RDF uses to describe data [POW03]. This schema can be used to validate a created RDF document.

### 4.3.2 DAML+OIL

DAML+OIL [DAM03] is a combination of the DARPA Agent Markup Language (DAML) and the Ontology Interface Layer (OIL). Both initiatives recognized their common goal and in 2000 they merged in the combined initiative DAML+OIL.

DAML+OIL extends existing standards (i.e. XML, RDF) to describe the structure of a particular domain in terms of classes and properties. A DAML+OIL description consists of a set of axioms that asserts relationships between classes or properties (i.e. intersectionOf, unionOf, complementOf, etc). In this way the semantics of the document is captured (see Example 2).

```xml
<rdf:RDF
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs=http://www.w3.org/2000/01/rdf-schema#
  xmlns:example=http://www.example.com/exampleschema>
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
>
  // This document defines the ontology
  <daml:Ontology rdf:about=""/>
</daml:Ontology>

  // Class "Gender"
  <daml:Class rdf:ID="Gender"/>

  // Male is "part of" Gender
  <daml:Class rdf:ID="Male">
    <rdfs:subClassOf rdf:resource="#Gender"/>
  </daml:Class>

  // Female is "part of" Gender but a female cannot be a "Male"
  <daml:Class rdf:ID="Female">
    <rdfs:subClassOf rdf:resource="#Gender"/>
    <daml:disjointWith rdf:resource="#Male"/>
  </daml:Class>

  // Assign properties to classes
  <daml:DatatypeProperty rdf:resource="#Gender">
    <rdfs:comment>
      Gender is a DatatypeProperty whose range is xsd:String.
      Gender is also a UniqueProperty (can only have one gender)
    </rdfs:comment>
    <rdf:type rdf:resource="http://www.daml.org/2001/03/daml+oil#UniqueProperty"/>
    <rdfs:range rdf:resource="http://www.w3.org/2000/10/XMLSchema#decimal"/>
  </daml:DatatypeProperty>
</rdf:RDF>
```

**Example 2: DAML+OIL document**

### 4.3.3 OWL

“The OWL Web Ontology Language is designed for use with applications that need to process the content of information instead of just presenting information to humans. OWL facilitates greater machine interpretability of Web content than that supported by XML, RDF, and RDF Schema (RDF-S) by providing additional vocabulary along with a formal semantics.” [W3C-5].

OWL is based on DAML+OIL. Generally, it uses the same language structure plus extensions. It is now being proposed as W3C standard for ontology and metadata representation. Just like
DAML+OIL, OWL is based on XML and RDF/RDF-S. It uses XML namespaces and URI’s, and furthermore concepts like disjointWith, subclassOf and unionOf are present.

4.4 Related work on ontology based service discovery

This section describes related work on ontology based service discovery. It describes COBRA, Ontomat-Service, CBSDP and OWL-S.

4.4.1 COBRA

The context broker architecture (COBRA) aims at developing an architecture that reduces the cost and difficulty of building context-aware agents. The COBRA architecture is a broker-centric agent architecture to provide runtime support for context-aware systems in an Intelligent Meeting Room environment [CHE03].

COBRA divides the world into several application domains (see Figure 18). Each domain is a knowledge model about a partial world in which people, devices and non-computing physical objects interact.

An application of the COBRA mechanism is presented the previous figure. It presents a directory service agent. This agent discovers the directories of a particular domain in a certain radius (i.e. location aware.). For instance, when two colleagues meet in the hall (i.e. ad-hoc meeting domain) the agent detects the directory of the colleagues such that they can share information.

4.4.2 Ontomat-Service

Ontomat-Service [AGA03] tries to combine the strength of the World Wide Web with the strength of semantic web services. They envision a service web that uses XHTML/XML/RDF to transport information and a web service framework to invoke operations and a framework to bind these two aspects together. The latter is developed in the Ontomat-Service project and is called the OntoMat-Service-Surfer.
In Figure 19 the complete process of Ontomat-Service is presented. It starts with creation of a Machine-understandable web service description (e.g. WSDL) which is compatible with the service provider ontology (i.e. using for instance RDF). This description is hard to understand for users therefore, in step one, this service description is transformed in a human readable description (e.g. HTML page). This description contains links to concepts of the service provider’s ontology with their semantic description. In step two, the annotator/user links concepts from his ‘client ontology’ to the service provider concepts described in the human readable description. This results in some mapping rules. In step three, several logical web service flows are generated using these mapping rules. In step four, the user can determine which flow he wants to execute.

4.4.3 CBSDP

The context based service discovery protocol (CBSDP) \cite{KHE03} is a service discovery protocol developed to overcome the issues with service discovery in highly dynamic ad hoc communications. It tries to solve the problems with current service discovery mechanism that are not capable of enabling services in a spontaneous networking environment. It provides mechanisms that supply initial service configuration and it enables service composition.

CBSDP provides users with the possibility to obtain services and information according to its context. The services are automatic detected by sensors. CBSDP uses a common ontology for the different services that behave as a basic communication between the agents in the environment. The ontology (i.e. uses RDF as encoding technology) is used to interpret the data of the service into meaningful information for the lookup service that is invoked when a user requests a service.

4.4.4 OWL - S

OWL-S \cite{OWL03} recognizes that not only content but also services are offered by Web resources. Users should be able to discovery, invoke, compose and monitor these services. OWL-S develops an ontology for services that makes this functionality possible. OWL-S is the successor of DAML-S and is part of the DARPA agent markup language program. Of all the discussed mechanism in this chapter, OWL-S is the most accepted and applied.

One of the big tasks OWL-S tries to enable is automatic service discovery. This involves the automatic location of web services that provide a particular service that adheres to requested constraints. By using the OWL-S markup of services a common description of services is created that can be matched more easily in the discovery process. Below, the top-level ontology of a service description is presented (see Figure 20).
Figure 20: Top level of the service ontology

It answers fundamental questions of a service like: what does the service do (ServiceProfile), how does the service work (ServiceModel) and how can the service be accessed (ServiceGrounding). The service Profile provides a way to describe the service offering by the provider and the needed service by the requestors. An OWL-S profile describes a service as a function of three basic types of information: what organization provides the service (e.g. contact information), what functions does the service computes (required inputs, produced outputs, pre-post conditions), and a set of features that specify characteristics of the service (e.g. service type, QoS). The service model describes how the service works. This is done by modeling the service as a process (see Figure 21).

Figure 21: Top level of the process ontology

The ServiceGrounding specifies the details on how to access the service. The details mainly describe transportation protocols and message formats, serialization, transport, and addressing. A grounding can be thought of as a mapping from an abstract to a concrete specification of those service description elements that are required for interacting with the service. OWL-S grounding is consistent with the WSDL [WSD03] binding mechanism. By using the extensibility elements already provided by WSDL, along with new extensibility element proposed in OWL-S, it is easy to ground an OWL-S process. In OWL-S, both the ServiceProfile and the ServiceModel are thought of as abstract representations. Only the ServiceGrounding deals with the concrete level of specification [OWL03].
4.5 Conclusions

Traditional and CA service discovery mechanisms have limited quality results because these mechanisms lack a common understanding on services. Matchmaking based on different knowledge levels is hard, if not even impossible. Ontology is an envisioned mechanism to overcome this limitation. In this chapter ontologies where introduced and their role in service discovery was indicated.

The main goal of ontologies is to create a common understanding on a specific domain. By creating this common understanding, the sketched problems with keyword-based service discovery mechanisms can be overcome (see chapter 2). Furthermore, in CA service discovery, the context can have different representation (see chapter 3). By using ontologies context can be unambiguously defined leading to better matchmaking. Therefore, ontologies may overcome the problem with non-common understanding in (CA) service discovery that leads to better quality discovery result.

Additionally, ontologies enable semantic matchmaking. Ontologies define concepts and their relations. Concepts and relations specify semantics of information better than non-related keywords. These semantic definitions can be used to match service and request based on properties and their semantics rather than on keywords (e.g. syntactic matchmaking). This can lead to more relevant discovery results.

Research on ontologies in computer science is relatively new. The discussed ontology technology and related work (see section 4.3 and 4.4) are not full-grown or commercially available. They present interesting concepts that could be applied in new ontology based discovery mechanisms (see chapter 6). The next chapter will discuss the WASP platform as an opportunity to introduce a new context-aware, ontology based, semantic service discovery approach. Furthermore, it discusses the current service discovery mechanism in this platform.

5 WASP services platform

The WASP platform is developed in the Web Architecture for Services Platform (WASP) project [WAS04]. The goal of this project is to create a supporting platform for mobile context-aware applications [EBB02]. It is a cooperation of the Telematica instituut, University of Twente and Ericsson. This chapter introduces the WASP services platform. It indicates why the platform provides a good opportunity to apply a context-aware, ontology based, semantic service discovery mechanism.

This chapter is structured as follows: Section 5.1 gives a general overview of the WASP architecture. It introduces the involved parties and the main components of the architecture. Section 5.2 focuses on the service discovery aspect of the platform. In section 5.3, some concluding remarks are given.

5.1 Overview of the WASP architecture

The WASP architecture provides a web-service ([W3C-1]) based application environment that enables context-aware applications [EBB02-1]. There are four main parties involved in the WASP architecture: service requestors, service providers, context providers and the WASP platform. These parties collaborate in context-aware service provisioning (see Figure 22).

Figure 22: Overall overview of the WASP architecture

The WASP architecture consists of three components: third party services, WASP platform and Context-Aware applications. These components will be discussed successively.

5.1.1 Third party services

Third party services are (web) services offered by parties other than the WASP platform. This component can be divided into three categories: 3G Network Services, Business Services (offered by Service providers) and Context Services (offered by Context providers).
• 3G Network Services provide network access capabilities, such as user identification, charging, call setup, messaging, etc. These network capabilities are accessible via web service interfaces and offered by mobile operators.
• Business services offer content and services for an application built on the WASP platform. Examples of these services are shopping, hotel reservation, and restaurant services.
• Context services provide information about the context of a user, e.g., the user status, the user location, etc. Furthermore, they can give personal users’ information of such as shopping lists or personal schedule. In addition, information that is not relevant to one particular user such as time or weather information can be offered by these services. In some cases, context can also be offered by 3G Network services. For example, location may be provided by a network operator by using the mobile device location. In that situation a 3G Network service is also a context service.

5.1.2 WASP platform
The WASP platform forms the system environment for the context-aware applications. The WASP platform is divided into sub-components responsible for supporting these context-aware applications.

• The request dispatcher is responsible for forwarding requests from the user to the 3G-network. For instance, transparently switching from a network operator to another. Furthermore, seamless roaming aspects and switching messaging services (e.g. SMS, MMS) are handled by this component.
• The notification manager provides functionality to subscribe to notifications when the context of a particular user changes. For instance, when a user is moving, his location changes. If the CA application is subscribed to the notification manager, it is notified of this context change and can react on his change.
• The context manager retrieves information about the user’s context by contacting the appropriate context providers. It is responsible for aggregating the context or deriving new context based on domain-specific rules (e.g., speed can be derived by dividing distance by the corresponding time interval). The context manager is also responsible for updating the notification manager on changes in the context.
• The service registry is a repository that contains information (i.e., service description) about the services provided by the 3rd party and context services. It enables searching for relevant services. In the advertisement phase, the service/context provider will advertise its service description in the registry.
• The matchmaker uses the service registry to discover the service that matches the request from the application. The discovery result is forwarded to the CA application. The final two components (service registry and matchmaker) form the current service discovery mechanism of the WASP platform. In section 5.2 this mechanism will be discussed in more detail.

5.1.3 Context-Aware application
The WASP platform acts as a supporting framework that enables mobile context-aware applications. These applications are deployed on top of the platform and they use the capabilities of the platform. The WASP CA-applications run on mobile devices such as PDA’s or smartphones. The user interacts with the application using the interaction manager. This can be activities like service request, network switch request, call setup, etc. The service retrieval component is responsible for forwarding a discovery request to the platform. Furthermore, it is also responsible for receiving the result and applying a personalization step defined in the personalization component [SET04] (e.g., score/sort the result based on predefined and/or historical defined user
interest). The real service invocation is performed outside the platform by using the service discovery result. An example of a context-aware application that is deployed on top of the WASP platform is “Tourist Compass”. This application is also developed in the WASP project.

**Tourist compass**

Tourist COMPASS is an acronym for Tourist COntext-aware MOBILE Personal ASSistant. The application provides tourists with information and services needed in his specific context (i.e. context-aware) that are interesting to him (i.e. personalization [SET04]). The main component of the application is a map that shows the current location of the user. Depending on the user’s profile and goals (e.g. his profile is that he likes architecture and his goal is that he wants to go to a hotel) a selection of nearby objects (e.g. buildings and hotels) are presented on the map. The map with objects updates when the user moves or other context changes occur. By clicking on a presented object, a provided services by that object can be executed (e.g. reserve a room at a hotel). In addition, routes can be displayed on the map. This creates a digital city walk. The user may also add his friends to the system. Making use of proper authorization mechanisms, the friends can see each other on the map and communicate via the platform using for instance GSM, MMS or SMS.

### 5.2 Service discovery in the WASP platform

One of the characteristics of the WASP platform is that it should intelligently search (i.e. service discovery) for relevant services in a broad and dynamic range of services. Two components in the WASP architecture are particularly interesting for the current WASP service discovery mechanism: the matchmaker and the service registry (see Figure 22). The discovery process of the WASP platform and the role of the service discovery components will be discussed according to the service lifecycle presented in chapter two. The discussion will be limited to the advertisement and discovery phase since these are part of service discovery.

#### 5.2.1 Advertisement phase

In the advertisement phase, the service/context provider creates a RDF description of the (context) service it wants to advertise. With a public method (publishAdvertisement) of the service registry (web service) the service provider can advertise (using the RDF advertisement) its service. This advertisement (see Example 3) is stored in the registry’s database as a ‘Service ID’/’RDF service description’ pair.

```xml
<rdf:RDF
  xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
  xmlns:service='http://www.freeband.nl/rdf/wasp/service#'
  xmlns:platform='http://www.freeband.nl/rdf/wasp/platform#'>
  <platform:WASPPlatformService>
    <service:name>UserLocationService</service:name>
    <service:description>GPS based location service</service:description>
    <service:url>http://foo.bar.org/UserLocationService</service:url>
  </platform:WASPPlatformService>
</rdf:RDF>
```

**Example 3: RDF service advertisement of a user location service (context service)**

In the previous example, an advertisement of a location service is presented. First, the RDF defines the appropriate namespaces (i.e. pointer to RDF schema). Then, it defines the WASP platform service that contains a name, a description and a URL where the service can be invoked. In this example, the WASP platform service is called a UserLocationService that is a GPS based location service for deriving the location of service requestors. To give an impression of the ontologies that forms the basis of the previous advertisement, the partial service ontology is
presented below (see Example 4). Although the ontologies are not enforced by the platform, they act as templates for service descriptions.

```xml
<rdfs:Class rdf:ID="http://www.freeband.nl/rdf/wasp/service#Service"/>
<rdfs:Property rdf:ID="http://www.freeband.nl/rdf/wasp/service#name">
  <rdfs:domain rdf:resource="http://www.freeband.nl/rdf/wasp/service#Service"/>
</rdfs:Property>

<rdfs:Property rdf:ID="http://www.freeband.nl/rdf/wasp/service#location">
  <rdfs:domain rdf:resource="http://www.freeband.nl/rdf/wasp/service#Service"/>
  <rdfs:range rdf:resource="http://www.freeband.nl/rdf/wasp/location#Location"/>
</rdfs:Property>

...</rdfs:RDF>
```

**Example 4: Partial Service Ontology (RDF-S)**

The first part of the ontology defines the general rdf and rdfs namespaces. It then defines that resource “Service”, defined by URI: http://www.freeband.nl/rdf/wasp/service, is a subclass of Resource. The ID is then a unique identifier to relate to the ‘Service’ concept. The following parts define properties of Service. The first statement defines that service has a name and the second defines that service has a location.

### 5.2.2 Discovery phase

The discovery phase of a service starts when the interaction manager requests service discovery. How often and when this occurs depends on the context-aware application. The matchmaker receives this request and queries the service registry using RDQL queries (i.e. RDQL [RQL04]). The service name and the service type are the concepts (related to a common ontology) on which is queried. Relevant services are returned to the application as an ‘ID’, ‘Advertisement’ pair that can be used to invoke a service.

On paper, the matchmaker and the service registry are complete separate components. In reality, these components can be seen as highly correlated component implemented by the Jena framework [JEN04] on top of a MySQL database. The Jena framework enables the context-aware application to query (RDQL) the database. So, in a sense the used Jena framework can be seen as a limited matchmaker.

### 5.3 Conclusions

The WASP platform forms a good opportunity to apply service discovery mechanisms that are context-aware and ontology based. The basic concepts used in the platform are similar to the ideas discussed in the problem analysis in the previous chapters. The platform should support intelligent searching of relevant services. It supports context and context-awareness and ontologies are used for common understanding.
Although these concepts can be applied in the platform, they are not yet used to their full potential:

- **Limited matchmaker:** The matchmaker solely consists of mechanisms (Jena) to query the registry using RDQL. This is kind of a static process where service name and service type, are used to query the service registry. It does not take into account concepts that describe the service the requestor needs (e.g. requested output, needed input, context criteria). Furthermore, the power of relations between concepts in the ontology is not used for semantic reasoning.

- **No CA service discovery:** The discovery process of the WASP platform is at this moment not context-aware. It does not consider context in its functionality. The CA application is at this moment responsible for filtering the discovery result using context.

- **Implements limited context types:** The implementation of WASP uses at this moment only ‘user location’ and ‘user interest’ as context type. More context types could be useful in service discovery.

- **No grounding reference:** In the advertisement no pointer is present to the grounding of the service (e.g. WSDL description), preventing ‘unaware’ applications (i.e. no stub code available) from dynamically invocating the service.

- **Ontologies are not enforced:** The platform uses ontologies, defined in RDF-S, but they are not enforced when the service is advertised. In this way, a common understanding is not guaranteed. Because the discovery phase is the focus of this research and enforcement of ontologies should take place in the advertisement phase, ontology enforcement will not be addressed in this thesis. Although it is not addressed in this thesis, it still is an important aspect which, if not applied properly, harms the quality of the service discovery result. We therefore consider it future work.

These issues create opportunities for a new service discovery mechanism in WASP. It should focus on extending the matchmaker into a more intelligent component that can match service and request based on dynamic semantic descriptions. Furthermore, it should incorporate context in its functionality. Other issues like: just one context type and referencing to grounding, are considered future work.

The following chapter will continue with the second part of this thesis. That part will focus on designing a new discovery approach that incorporates the discussed issues from the problem analysis to provide a higher quality discovery result.
6 COSS - Service discovery approach

This chapter describes a new service discovery approach that is our attempt to improve the service discovery quality of current service discovery techniques discussed in the previous chapters of this thesis. This new mechanism is called Context-Aware, Ontology based, Semantic Service discovery (COSS).

This chapter is structured as follows: section 6.1 gives a general overview of COSS. Section 6.2 introduces the concept ‘matching degree’ that is a quality measurement for discovered matches. It is used to rate matches for their relevancy for the service requestor. Section 6.3 elaborates on the main aspect of COSS, the matching algorithm. Section 6.4 illustrates a case to explain the mechanisms of COSS in a practical situation. Finally, section 6.5 presents the implementation of the COSS service discovery prototype.

6.1 Overview of COSS

We start our discussion by positioning the COSS approach with respect to the problem analysis described in the previous part of this thesis. Furthermore, we go into the details of the COSS approach by analyzing its general architecture and the issues that need to be considered when using this method.

6.1.1 Positioning of COSS

So far, we have discussed the issues that limit the quality of the service discovery result in several service discovery methods. COSS tries to improve the quality of the service discovery result by addressing the distinguished issues from the problem analysis. The problem analysis is summarized in the table below (see Table 2). The table also presents the position that COSS takes.

<table>
<thead>
<tr>
<th>Issue</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service requestors want to use services but have difficulty finding them.</td>
<td><em>Traditional service discovery</em></td>
</tr>
<tr>
<td></td>
<td>- Syntactic matching (keyword/table based).</td>
</tr>
<tr>
<td>Low recall/precision:</td>
<td><em>Context-Aware service discovery</em></td>
</tr>
<tr>
<td>- does not use context.</td>
<td>- Theoretical higher quality (request/input completion).</td>
</tr>
<tr>
<td>- does not provide a common understanding.</td>
<td></td>
</tr>
<tr>
<td>- does not do semantic matchmaking.</td>
<td></td>
</tr>
<tr>
<td>- does not provide a common understanding.</td>
<td><em>Ontology based service discovery</em></td>
</tr>
<tr>
<td>- does not do semantic matchmaking.</td>
<td>- create common understanding.</td>
</tr>
<tr>
<td>Gets more important.</td>
<td>- enables semantic matchmaking.</td>
</tr>
<tr>
<td>The current WASP platform has a limited service discovery mechanism.</td>
<td><em>COSS</em></td>
</tr>
<tr>
<td></td>
<td>- incorporate context.</td>
</tr>
<tr>
<td></td>
<td>- use ontologies for common understanding and do reasoning.</td>
</tr>
<tr>
<td></td>
<td>- does semantic matchmaking.</td>
</tr>
</tbody>
</table>

The main aspects of COSS that improve the quality of the service discovery result are:

- **Context-awareness**: COSS collects and uses contextual information to return more relevant matches (i.e. request and input completion like discussed in chapter three).
- **Ontologies**: COSS makes use of ontologies to create a common understanding and do reasoning.
- **Semantic matchmaking**: COSS performs semantic matchmaking thereby using ontologies.

The following figure (see Figure 23) visualizes the position of COSS with respect to the main information retrieval technologies introduced in chapter two.

![Position of COSS related to main information retrieval methods](image)

**Figure 23: Position of COSS related to main information retrieval methods**

We position COSS in the space between the concept-based approaches and deductive retrieval approaches. The deductive retrieval approaches offer higher recall and precision, however, modeling service functionality by the means of formal logic is sometimes an extremely difficult task [KLE04]. Another disadvantage of the deductive approach is that the search process is usually very slow due to the high computation complexity that this process involves.

### 6.1.2 The COSS approach

The COSS approach is a service discovery mechanism should be able to be easily incorporated in the WASP platform (see Figure 24).

![Application of COSS in the WASP platform](image)

**Figure 24: Application of COSS in the WASP platform**

Incorporating the COSS approach has some consequences for the current WASP platform (see chapter 5). In general, the main components of the current WASP system are preserved except for the matchmaker that will be extended to perform the COSS matchmaking algorithm that will be discussed in section 6.3. Furthermore, the context manager needs to be included in the service discovery mechanism to provide contextual information to enable context-aware service discovery. The service registry is enhanced such that it can provide service descriptions with extensive information for the COSS matchmaking algorithm (see section 6.3).
Figure 25 gives an abstract overview of the COSS method.

![Diagram](image.png)  
*Figure 25: Abstract overview of the COSS system*

The inputs of COSS are: the service request, a set of advertised services, a set of context providers and ontologies. Common understanding on services is achieved by using the shared ontology to which the requestor, service provider and COSS commit.

Context-aware service discovery is enabled by the association between the service requestor and possible context providers (e.g., a user has a context provider that can deliver his location). This association will be used to do input and request completion as discussed in chapter 3 (e.g., the context provider can be contacted to provide users’ preferences as added contextual information to the COSS approach. Furthermore, the context provider may be requested to supply inputs that were not directly provided by the user that is needed for the delivery of a service). We assume that the platform is aware of the associations between the context providers and the user.

In order to do semantic matchmaking COSS uses ontologies. Within an ontology, concepts and their relations to each other are described. By reasoning on concepts and relations, COSS can derive matches that syntactically would not match. Consider, for instance, a shop that defines that it sells ‘music products’ (e.g. cd’s, dvd’s). When the user specifies that he wants to buy a ‘cd’, a syntactic mismatch occurs. When we use an ontology (that is shared by both parties) to derive that ‘cd’ is a type of ‘music product’, we can infer that this is a semantic match. This requires that the service request and the service descriptions are not described by using keywords but by their properties which are related to the concepts from the shared ontology (see Figure 26).

![Diagram](image.png)  
*Figure 26: Request and service modeling*
Service properties can be of different types that have to be handled differently by the matching algorithm. COSS distinguishes four types of properties:

- **Service type:** Indicates the type of services that the user requests, and classifies the services in categories. This property is used for rough filtering (see section 6.3).
- **Outputs:** This is the information that the user requests and the service provider possibly can provide.
- **Inputs:** These are the inputs the user possibly can provide and the service provider needs.
- **Attributes:** This is contextual information that is provided by the user (e.g. nearby, open) that may require the use of other contextual information (e.g. userlocation). It specifies some property the user likes the service to have.

Figure 27 shows the role of these properties for the request and the service. These roles are similar but not equal for both parties (e.g. an output for the service \(S_o\) means that the service can provide this output while an output for the request \(R_o\) means that the requestor wants to discovery a service that can provide this output, etc.). The naming conventions presented in this picture are used throughout this chapter to indicate the property types of the service or request.

Intuitively, the inputs needed by the service \(S_i\) can be divided into two categories: mandatory and optional. Mandatory inputs are inherent to the provisioning of the outputs while optional inputs may enhance the outputs (e.g. provide filtered outputs using the optional input). We do not model this difference by separating optional and mandatory inputs, but we create separate services to encapsulate optional inputs (see Figure 28).

The figure presents an example of a music service that has artist as mandatory input and PriceRange and Category as optional inputs. The top level is the service with only the mandatory inputs. For every optional input, a new service (i.e. subclass) is created with the optional input as mandatory input. In this way, three additional services (also combinations of optional inputs) are created with only mandatory inputs. COSS can then discovery every individual service without bothering on optional inputs.
6.1.3 Grounding

To be able to invoke a service after its discovery, we use a WSDL grounding mechanism. WSDL [30] defines services as collections of network endpoints. The abstract definition of an endpoint, called interface, is separated from its concrete network deployment, protocol and data encoding through reusable bindings. Interfaces are abstract collections of operations that contain input and/or output messages that consist of message parts. Figure 29 presents the mapping between our service model and the WSDL metamodel.

![Figure 29: WSDL metamodel and COSS mapping](image)

In our service model, each service has a service type. This service type is mapped to a WSDL interface. The service itself maps to an operation in this interface (e.g. SellMusicCD). The inputs and outputs of the service map to messages in WSDL, whereas concepts map to message parts. The following example presents an example of our grounding mechanism.

```xml
...<operation name="SellMusicCD">
  <input message="credit_card"/>
  <output message="CD" />
</operation>
...<message name="credit_card">
  <part name="type" payontology:output="payontology:#CreditCardType" />
  <part name="card" payontology:output="payontology:#Card"/>
  <part name="expire" payontology:output="payontology:#ExpireDate" />
</message>
...
```

6.1.4 COSS ontologies

Ontologies are one of the main components of the COSS system. Besides providing mechanisms for common understanding on services, ontologies also enable semantic reasoning. This section describes the developed ontologies and their role in the COSS system. The application that is currently deployed on top of the WASP platform is a tourist application. Therefore, the developed ontologies used by COSS are tourist-related. There are seven ontology types, namely: service, service type, product, attribute, product, context and payment ontology. Figure 30 presents an overview of the COSS ontologies and their mutual relation.
Concepts are marked *italic and underlined* and a relation is marked *italic*. In Figure 30, ‘subclass’ relations are depicted with non-dotted lines while special relations are depicted with dotted lines. Relation-subtypes are marked with rounded rectangles.

**Service ontology**

The service ontology models the general structure of a COSS service. As discussed in the previous section, a service is described by its properties. A *service* provides *outputs*. These outputs can be concepts derived from the general concept *Thing* (derived from the product ontology). Furthermore, a *service* needs certain *inputs*. These inputs are again derived from the *Thing* concepts (derived from the payment or product ontology). A *service* has a certain *type* that is a concept from the service type ontology. Furthermore, the *service* may have some *attributes*. The attribute ontology presents subclass relations of this *attribute* relation. In general, the realization of *Thing* concept linked to the *attribute* relation will be a Boolean indicating if a property is exhibited by the service. Finally, a service has a *servicename*, which is merely used for identification of the service from the requestors’ perspective.

**Service type ontology**

The service type ontology models the possible types a service can have in our application scenario. First, it defines that a service can be a *selling service*, *Rental service*, *Manufacturing service*. The selling service _sells_ some _product_. There are several types of selling services: *Shop* and *Restaurant*. The shop is divided into *Souvenirshop*, *Musicshop* and *Foodshops*. The restaurants are divided into *Dutch restaurant*, *Italian restaurant* and *Vegetarian restaurant*. The sell relation restrictions are specified to define the semantics of the several shops and restaurants. For instance, a souvenir shop sells souvenirs and a vegetarian restaurant sells vegetarian food. These two examples can be represented in first order logic and corresponding semi-OWL in the following manner:
A **Manufacturing service** and a **Rental service** can both be a **Shop**. This is a definition of homonyms. All the shops are syntactically defined the same (i.e. with the word shop), but have different concepts in the ontology.

**Product ontology**

The product ontology specifies the physical products that are offered by the services. A **Product** has a **name** and a **price** that are respectively a string and a double. Furthermore, the ontology defines that **Souvenir**, **Food**, and **Music** are types of **Product**. **Food** and **Music** are further refined by defining that there are several types of food and music (**Dutch**, **Italian**, **Vegetarian** and **CD**, **LP** and **DVD**).

**Payment ontology**

The payment ontology specifies the payment methods supported by the services. Practically these concepts are often used as input concepts. The ontology defines that payment can be **Cash** or **Card** or **Customerpoints**. Card payment can be **PIN**, **Credit card** or **Debitcard**.

**Attribute ontology**

Some attributes (i.e. **attribute** relation) can be applicable to services. Therefore, the attribute ontology specifies the attributes the user can define. These properties are defined as subclass relations of the **attribute** relation to be able to restrict this relation. We researched tourist information sites for commonly used criteria in tourism applications. We defined ten different attributes that are useful in our tourist case:

- **nearby**: A service is near to the user location.
- **open**: A service is open (i.e. depends on time and date).
- **parking**: A service has parking places.
- **train**: There is a train station nearby.
- **non-smoking**: The service applies a non-smoking policy.
- **indoor**: Defines if the service is indoors or outdoors.
- **kidscare**: The service has some facilities for kids.
- **bus**: There is public transport nearby (i.e. bus).
- **price range 1**: The range of product offered by the service is between EUR 0 – 50.
- **price range 2**: The range of product offered by the service is between EUR 50 – 200.
- **price range 3**: The range of product offered by the service is EUR >200.

**Context ontology**

The context ontology specifies the contextual information that is used by the COSS approach. There are three types of context:

- **Location**: Physical location of an object. It consists of latitude (**Lat**) and longitude (**Long**) values.
- **Time**: Specifies the notion of time for an object. It consists of a definition of a time **Interval**. A time interval has a **begintime** and **endtime**.
- **Date**: Specifies the current data of an object.
6.1.5 COSS matching documents

This section gives an overview of COSS matching documents. These documents are the primary source of information for COSS to do service discovery. There are two types of matching documents used in COSS:

- **Service advertisement**: represents the offer of the service provider.
- **Service request**: represents the need of the service requestor.

These documents are defined using OWL [OWL03] and specify a certain instance of the service ontology described in the previous section.

**Service advertisement**

This document encapsulates the offering of the service provider. Consider for instance the following example of a definition of a music shop in semi-OWL followed by the same music shop defined in first order logic.

```owl
Musc_Shop = intersectionOf(
  Service,
  restriction onProperty output toValue Music
  restriction onProperty input toValue name
  restriction onProperty type toValue Musicshop
  restriction onProperty property toValue intersectionOf(
    Attribute,
    restriction onProperty parking toValue TRUE
  )
)
```

The OWL advertisement starts by defining that the advertised shop is an intersection of all services with certain restriction on their properties. It then restricts its properties. It defines that it outputs Music, it needs a name as input and it is service of type ‘music shop’. It then defines the attributes that it has: parking space for its customers.

**Service request**

This document encapsulates the need of the user. Consider for instance the following example of a request definition for a music shop in semi-OWL followed by the same request in first order logic.

```owl
Request_Music_Shop = intersectionOf(
  Service,
  restriction onProperty output toValue CD
  restriction onProperty input toValue name
  restriction onProperty type toValue Shop
  restriction onProperty property hasValue intersectionOf(
    Attribute,
    restriction onProperty Parking toValue TRUE
    restriction onProperty PriceRange1 toValue TRUE
  )
)
```

The OWL advertisement starts by defining that the advertised shop is an intersection of all services with certain restriction on their properties. It then restricts its properties. It defines that it outputs Music, it needs a name as input and it is service of type ‘music shop’. It then defines the attributes that it has: parking space for its customers.
First, this request defines that it searches for all services that have the defined restriction on their properties (i.e. intersectionOf). It restricts the output to CD. This means that the user is interested in services that can output CD’s. Furthermore, the request specifies that it searches services of type Shop. Then it defines that the user can provide input name. Finally, it defines the attributes: parking space and price range of zero to fifty euros.

6.1.6 COSS service attributes

The attributes used in COSS are defined by the user. They provide contextual information that is implicitly available for the service discovery process. An example of an attribute is ‘nearby’. This attribute specifies that services that are physically near to the user are important to him. This information is used by COSS to enrich the service discovery request to return higher quality matches.

The attributes by the user, such as nearby, cannot be directly handled by COSS. The semantics of this attribute has to be defined to enable COSS to reason on this attribute. Therefore, a simple rule based method is used:

\[ \text{Attribute} \rightarrow \text{statement} \]

This general rule states that reasoning on an attribute can be done by evaluating the statement. This statement then consists of normal math and Boolean operation together with concepts from the ontology (marked italic in the examples). Consider for instance the following rule that can be specified for ‘nearby’.

\[ \text{nearby} \rightarrow \text{distance(userposition, serverposition)} < \text{maxdistance} \]

This rule states that the attribute ‘nearby’ can be evaluated by determining whether the distance between the serviceposition and the userposition is smaller than a certain value (maxdistance). To evaluate this expression, context providers have to be contacted to retrieve the location of the service and the user. In general, it is often the case that to evaluate these rules, context providers have to be queried. Querying context providers to retrieve the needed information to evaluate the statements is the responsibility of COSS.

6.2 Matching degree

The main goal of COSS is to return relevant matches (i.e. high precision and recall). The matched services have to be close to the user’s request. However, what does a close match mean? When is a match close? These questions relate to the quality of a match. This section elaborates on how to measure the quality of a match. It introduces a concept to rate the quality of a match, called ‘matching degree’.

We define matching degree as: a quality measurement that indicates the closeness of a service description to a service request. This measurement can be used to distinguish matches on their relevancy for the user (i.e. service discovery quality). Based on this measurement, COSS can decide to return, reject or possible conditionally return a service (e.g., service that can be used if additional manual inputs are provided by the user) as a relevant match to the user request.
The request of a user can be modeled as an intersection of sets of services that have certain properties (see Figure 31). In the figure, the service request represents a need for a service with output “output”, additionally the user must provide input “input”, and the user wants a service that has “attribute 1” and “attribute 2”. All the available services (in this case Service{0,1,2,3,4}) in the service repository can be positioned somewhere in this framework. For instance, service 0 is positioned in the same intersection (i.e., it has the same properties) as the request. Intuitively, it is clear that this is an exact match. Service 1 is positioned in the intersection of the output, input and one attribute. Therefore, service 1 misses one requested attribute namely: attribute 2. Therefore, this is not an exact match. Service 2 and 3, both miss two properties. Service 2 misses the properties while service 3 misses the input and output. Intuitively, it is clear that service 2 is a better match than service 3 because it confirms with the input and the output. As can be seen, a match can be rated by the position (i.e. number of properties the service misses) the service takes related to the service request. However, there is a need for a hierarchy of property types which are weighted by their importance for the request, to be able to rate matches more precisely (see section 6.3).

The above presented model can be simplified to similar models for every particular match. A match (i.e. request vs. service) can be modeled in two sets of properties that the request (R) and service (S) have. Different types of matches can then be distinguished [LI03] (see Figure 32). The number of missing properties (m) between the request and the service indicate in which category the match fits:
The first type indicates an exact match. The request is equal to the service; there are no missing properties. This is the best possible match. A ‘plugin match’ is the second best match. It indicates that the service is capable of more properties than the requestor asks. This can also be seen as an exact match because the requirements of the user are met. The third and fourth match types indicate that the service can only provide a part of what the requestor wants. The number of missing properties is bigger than zero. The fifth match type indicates a disjoint match. No properties of the request can be provided by the service. The number of missing properties equals the total number of requested properties. This is the worst possible match. This type of match is not returned to the user.

COSS adapts this model such that it consists of three types of matches that are useful for the requestor:

- **Precise match**: Exact and Plugin matches. The services are capable of providing the requested functionality or more.
- **Approximate match**: Subsume and intersection matches. The services are capable of part of the requested functionality.
- **Mismatch**: Disjoint match. The services are not capable of the requested functionality and will not be returned to the user.

### 6.3 COSS matching algorithm

This section elaborates on the COSS matching algorithm. The previous section presented a distinction of three types of matches. Furthermore, it indicated that a hierarchy of weighted service properties types is needed to rate matches more precisely. The first part of this section elaborates more on this property hierarchy. The second part presents a model to handle properties. The final section presents the COSS matching algorithm that uses the property hierarchy and the property model to classify services into the distinguished match types (i.e. precise, approximate and mismatch).

#### 6.3.1 Property type hierarchy

The section on matching degree (see section 6.2) indicated that the quality of a match depends on the number of missing properties. However, not all properties are equally important. Intuitively, it is clear that a service that is missing an attribute is better than a service that is missing an output.
Therefore, COSS classifies its service properties by importance for the quality of the discovery result:

1. **Service type** \((S_t, R_t)\): services type the user requests.
2. **Requested output** \((R_o)\): service output the user requests.
3. **Required input** \((S_i)\): input the service needs to provide a requested output.
4. **Attributes** \((R_a)\): criteria the service may conform to.

The first two property types are the most important for the quality of the match. They can be seen as hard criteria. If the requested service type or one of the requested outputs are not present at the service, the service will be rated as a mismatch and will not be returned to the requestor.

The third property type (required input) is a soft criterion. If the user cannot provide a required input, it is queried at the context providers. If it is found the match is rated as a precise match, otherwise it is rated as an approximate match because the user could perhaps manually provide the input when using the service (i.e. choice of the requestor). If an input cannot be provided, we assume that it does not matter for the requestor if multiple inputs are missing. Therefore, the approximate matches are not ordered by the number of missing inputs but by the attributes they have. ‘Attributes’ is the final property type. The number of missing attribute indicates how high or low a certain service is positioned in the list of precise and approximate services. The next section elaborates more on the role of these attributes in COSS.

### 6.3.2 Concept lattices attribute model

COSS attributes are used to order the lists of precise and approximate services. To be able to do this rating, a mechanism is needed that clusters services by their attributes. The mechanism we use is called concept lattices. The first part of this section gives a brief overview of concept lattices while the second part discusses the application of concept lattices for COSS attributes.

#### Overview of concept lattices

Concept lattices [GAN03] is a mechanism used in formal concept analysis. It can be used to study how objects can be hierarchically grouped together according to their common attributes. The starting point is a lattice model (i.e. called formal context but for clarity, this term is not used in this thesis) which consists of a triple \((G, M, I)\). \(G\) is set of objects, \(M\) is a set of attributes and \(I\) is a binary relation between them \((I \subseteq GxM)\). A common used representation of this model is a cross table (see Figure 33, all the examples are created using Concept explorer [CON04])

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Obj1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Obj2</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Obj3</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 33: Lattice table**

Each object is one row in the table while the attributes are the columns. The binary relation is presented by a cross at the intersection of a row and a column. For instance, when considering the previous figure, you can see that Object 3 has Attribute 1 and Attribute 2 etc. The lattice table is the basis for a lattice line diagram that visualizes the attribute communality of the objects (see Figure 34).
This is a hierarchical diagram which presents the most generic objects at the top while getting down in the diagram the objects get more specific (i.e. have more attributes). A node in the diagram is called a *concept*. It can contain objects that share the same attributes. Such a concept shares the *attributes* from its parents in the diagram. The top node is a set of objects that contains no attributes. One level down object 1 is encapsulated by a concept that contains attribute 1. Again one level down we see that object three has a relation with the concept containing attribute 1 and with a concept containing attribute 2. Therefore, object 3 has attribute 1 and attribute 2 and shares attribute 1 with object 1. Object 2 has attribute 3 and shares attribute 2 with object 3. The bottom node contains objects that have all attribute (in this case empty).

There are several reasons why concept lattices can be applied to COSS attributes:

- Concept lattices offers clustering of object by their similar attributes. Therefore, by relating the requested COSS attributes with the attributes the services have, the desired clustering can be reached.
- The division of objects, attributes and binary relation in concept lattices is analogues to the modeling of services (objects) that conform to (relation) COSS attributes (attributes).
- Concept lattices are a mathematical model with a sound mathematical basis. This enables COSS to reason on a position of objects in the lattice to rate the sets of precise and approximate matches (i.e. rate matches).
- Concept lattices can be represented by a cross table. This simple mechanism makes the creation of an attribute reasoning model relatively easy.
- Besides that creating the model is simple, reasoning is also relatively simple. The lattice clusters objects with common attributes. By investigating parent and child relations, some conclusions can be made on the quality of a service match based on its attributes.
- The Telematica Instituut developed a java lattice library that can be enhanced and used easily.

**Application of concept lattices for COSS attributes**

After the initial classification of services in sets of precise and approximate matches, these sets are ordered using the requested attributes. Therefore, all attributes specified by the requestor are added as attributes (columns) in the lattice table. The request itself is added as an object (row) which has a binary relation (cross) with all the added attributes. Then, all the services are added as objects in the table and the attribute rule is evaluated for this service and the request. The result will be a (cross) relation if the attribute statements evaluates to true (see Figure 35a).
Consider the previous figure. After evaluation, the binary relations are inferred: service 4 is not open and nearby, service 1 is open, service 2 is nearby and service 3 is both open and nearby. The resulting table is then used to create a lattice diagram (see Figure 35b). In the lattice diagram, the request and the services are encapsulated by some concept. By reasoning on the position of a service in this diagram related to the request, an ordering of services can be derived. The request is always the bottom concept in the lattice because it always has all the attributes. Services that are encapsulated by this concept have all the attributes. They are therefore placed at the top of the list. The higher you get in the diagram the more attributes are missed by the service. These services are then successively rated lower (see Figure 36 for the ordering of service from the previous example)

<table>
<thead>
<tr>
<th>Service discovery result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service 3</td>
</tr>
<tr>
<td>Service 2</td>
</tr>
<tr>
<td>Service 1</td>
</tr>
<tr>
<td>Service 4</td>
</tr>
</tbody>
</table>

Figure 36: Ordering of services

The user can then decide if the service that misses properties is attractive to use. An extension to this mechanism would be to let the user assign weights to attributes. This enables an even more personalized ordering. However, this is not the focus of this research and is considered future work.

With only a limited amount of attributes and objects it seems cumbersome to use concept lattices to create an ordering. However, as can be seen in the case in section 6.4, when the number of attributes and objects increases, it becomes very difficult to order the objects in an efficient manner. The power of concept lattices to cluster objects with common attributes is then used to do the ordering process effectively.

6.3.3 Matching algorithm

The main component of the COSS method is the COSS matching algorithm (i.e. matchmaker). This algorithm categorizes services into ordered perfect and imperfect matches based on the user request. Informally, the algorithm can be divided into five main phases (see Figure 37).
The starting point of the matching process is a set of all services available to the COSS matchmaker (e.g. \( n \)). The first step will filter this set for services that are of the requested service type. This results in a smaller set of services (e.g. \( n-k \)) with service type \( R_t \). The second step will filter the result set from the previous step for requested outputs. Again, this results in a smaller set of services (e.g. \( n-k-m \)) that can provide the requested output \( R_o \). The services of this set are then queried for the inputs they need. If the needed inputs \( (S_i) \) are provided or can be found at the context providers the service is classified as precise. Else, the service will be classified approximate. The final step orders the two sets using the user defined attributes like discussed in the previous section. As can be seen, the matchmaking process of COSS analysis a big set of services (i.e. relevant and not relevant services), and along the way it reduces this big set to lists of services that are relevant for the requestor.

The next part of this section will give a more formal and exact definition of the COSS matching algorithm. This definition corresponds with the previously explained informal description. The COSS matching algorithm takes as input a request \( R \) that contains the requested outputs \( R_o \), provided inputs \( R_i \), requested service type \( R_t \) and requested attributes \( R_a \) from the service requestor. The second input is a set of service advertisements \( S \) stored in the service registry. Every service \( s (s \in S) \) in this repository consists of provided outputs \( S_o \), needed inputs \( S_i \) and service type \( S_t \). First the COSS matching algorithm is presented in natural language followed by a description using pseudo code:

**COSS Matching algorithm:**

1. Query \( S \) using the required service type \( R_t \). Result is a set of services \( S' \) of type \( R_t \). Implicitly, services that do not have type \( R_t \) are rated as mismatches.
2. Query \( S' \) using the requested outputs \( R_o \). Result is a set of services \( S'' \) which all have outputs \( R_o \). Implicitly, services that do not have output \( R_o \) are rated as mismatches.
3. Query \( R' \) for the inputs needed for all \( s'' \in S'' \). The results are two sets of service \( P \) (precise services) and \( I \) (approximate services).
   a. If all inputs are provided by \( R \), the service is classified as Precise (added to set \( P \)).
   b. If an input is not provided by \( R' \) then search for it at the context providers.
      i. If all the missing inputs are found, the service is classified as Precise (added to set \( P \)).
      ii. If one of the missing inputs is not found, the service is classified Approximate (added to set \( I \)).
4. Reason on a concept lattice of services in \( P \)
   a. Add \( R_a \) as attributes in the lattice
   b. Add \( R \) as an object in the lattice and add relations.
c. Add all services from P as objects in the lattice.
   i. Determine if attribute is applicable to service.
      1. If the attribute is applicable then add relation.
   d. Order set P using the lattice. Result is an ordered list of service P'.
5. Reason on a concept lattice of services in I.
   a. Add Ra as attributes in the lattice
   b. Add R as an object in the lattice and add relations.
   c. Add all services from I as objects in the lattice.
      i. Determine if attribute is applicable to service.
         1. If the attribute is applicable then add relation
   d. Order set I using the lattice. Result is an ordered list of service I'.
6. Present result (P', I').

```java
Coss_Matching(R) {
    S' = queryRegistry(R_o, S)
    S'' = queryRegistry(R_o, S')
    forall s in S'' do {
        s_i = queryInputs(s)
        if provided(s_i, R_i) then {
            P.append(s)
        } else {
            if queryContextProviders(userID, missingInputs(s_i, R_i)) then {
                P.append(s)
            } else {
                I.append(s)
            }
        }
    }
    P_initlattice = createInitialLattice(R_a)
    forall p in P do {
        P_initlattice.addObject(p)
        forall attr in R_a do{
            if evalAttrRule(p, attr) then {
                initlattice.addrelation(p, attr)
            }
        }
    }
    P_lattice = createLattice(P_initlattice);
    P' = orderservices(P_lattice);
    I_initlattice = createInitialLattice(R_a)
    forall i in I do {
        I_initlattice.addObject(p)
        forall attr in R_a do{
            if evalAttrRule(i, attr) then {
                initlattice.addrelation(i, attr)
            }
        }
    }
    I_lattice = createLattice(I_initlattice);
    I' = orderservices(I_lattice);
    return result(P', I')
}
```
The first step roughly filters the service repository for relevant services using the requested service type. The second step uses the filtered set of services from the previous step and queries it for services that can provide the requested outputs. The services in the resulting set are then queried for the inputs they need. COSS then checks if the needed inputs can be provided by the request or can be found at the context provider. If found the matches are rated precise else approximate. Then, the two resulting sets are ordered by relevancy for the user using concept lattices. In the final step, the ordered lists of precise and approximate services are presented to the user. The dynamic behavior of COSS is presented in the time sequence diagram in Figure 38.

![Figure 38: COSS dynamic behavior](image)

The process starts by a request from the user. The request and a unique user id are delivered to the matchmaker. The matchmaker starts the matching algorithm. It will query the service registry for services with the requested service type. The following step is querying the service registry for service that can deliver the requested output. Then, for every service capable of delivering the requested output, it is checked if the request provides the inputs. Therefore, the registry is contacted to deliver the inputs of a certain service. If the request does not provide the input, the context manager is contacted to search for implicit context input the user can provide. Then, the matchmaker validates for every remaining service if the found attributes of the user are applicable to the service. Therefore, again the context manager is contacted to return the result of the evaluation of the attribute rules. The result is used to order the sets of precise and approximate services. Finally, the result is returned to the requestor.

### 6.4 CASE: Shopping

This section discusses a case that presents the working of the COSS matching algorithm and the rating of different matches. The case is a tourist application that allows users to search for shops and restaurants. This application will also be the focus of the implemented prototype (see section 6.5).

#### 6.4.1 Case specification

Consider a male tourist who is walking in an unfamiliar city. He is looking for a Music store but cannot find one. He takes his mobile phone and issues a request with the intended result to find a Music store. After acquiring this request using some kind of GUI, the following semi-OWL description is created to represent the request:
SearchCDShop = intersectionOf(
  Service,
  restriction onProperty output toValue Music
  restriction onProperty input toValue Music
  restriction onProperty type toValue Shop
)

This document has to be matched with the services advertisements available in COSS. Consider the following services that are available in the COSS service repository (i.e. attributes are dealt with later one):

Service type: Shop
- Service 1 - input music, output music, [attributes]
- Service 2 - input music, output music, [attributes]
- Service 3 - input location, output music, [attributes]
- Service 4 - input locations and time, output music, [attributes]
- Service 5 - input CD and time, output CD, [attributes]
- Service 6 - input time and location, output Souvenir, [attributes]
- Service 7 - input time and location, output Vegetarian, [attributes]
- Service 8 - input music and location and time, output music, [attributes]
- Service 9 - input CD and time, output CD, [attributes]
- Service 10 - input music and time, output music, [attributes]
- Service 11 - input music, output music, [attributes]
- Service 12 - input CD, output music, [attributes]
- Service 13 - input Souvenir, output Souvenir, [attributes]

Service type: Restaurant
- Service 14 - output Dutch restaurant, [attributes]
- Service 15 - output Italian restaurant, [attributes]

6.4.2 COSS matching process

The first step roughly filters all the available service (S) using the service type (S_t). This results in a set (S') with thirteen services (service 1 until 13) that possibly are relevant to the user. Implicitly service 14 and 15 are rated disjoint.

The second step, of the algorithm is to query the result set with the requested outputs (R_o). Because CD is a subtype of Music, inputs and outputs CD are also valid. This result in a set (S'') that consists of ten services (service 1 until 5 and service 8 to 12). Service 6, 7 and 13 are implicitly marked disjoint and will therefore be ignored.

In the third step, every service in the resulting set from the previous step is queried for their needed inputs (S_i). Then COSS checks if such a needed input is explicitly provided by the request (R_i). If it is provided, the match is rated precise; else, the input is searched at the context providers. If found at the context providers, the match is rated precise else it is rated approximate. Consider, for instance, service 3. This service is in the resulting set S'' and it needs location as input. This input is not provided by the requestor therefore it is searched at the context providers. The context manager is contacted and it checks whether a context provider can provide this input for the requestor. Assume that the requestor has an association with a location provider but not with a time provider. Therefore, service 3 is marked precise while service 4 is marked approximate (i.e. time cannot be provided). The same process is applied to the other services. This leads to the following categorization of services in two sets (P and I):
Table 3: Classification of service in Precise and Approximate matches

<table>
<thead>
<tr>
<th>Precise (P)</th>
<th>Approximate (I)</th>
<th>Disjoint (implicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service 1</td>
<td>Service 4</td>
<td>Service 6</td>
</tr>
<tr>
<td>Service 2</td>
<td>Service 5</td>
<td>Service 7</td>
</tr>
<tr>
<td>Service 3</td>
<td>Service 8</td>
<td>Service 13</td>
</tr>
<tr>
<td>Service 11</td>
<td>Service 9</td>
<td>Service 14</td>
</tr>
<tr>
<td>Service 12</td>
<td>Service 10</td>
<td>Service 15</td>
</tr>
</tbody>
</table>

The fourth and fifth step, order the two resulting sets P and I. First the request and its corresponding attributes are added to a lattice table. Then, every service in the set is added to the table thereby evaluating the attribute for that particular service (e.g. determine if a service is near to the user location). If the attribute evaluates to true a relation (cross) in the table is added. This results in the following two lattice tables for the precise and approximate matches.

By reasoning on the position of services with respect to the request, an ordering of services can be determined. For set P, service 2 is encapsulated by the same concept as the request. This means that service 2 exhibits the same attributes as the user request and is therefore placed at the top of the list. Other services are encapsulated by parents (i.e. higher nodes in the hierarchy) of the concept encapsulating the request. This means that they miss attributes that the user requests. Therefore, they are positioned lower in the list. The result of this ordering is two lists (P’ and I’) of services that are presented to the requestor.
Table 4: Service discovery result

<table>
<thead>
<tr>
<th>Matches</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>+Perfect matches</td>
<td>5 match(es)</td>
</tr>
<tr>
<td>-Service 2</td>
<td>Misses 0 attribute(s).</td>
</tr>
<tr>
<td>-Service 11</td>
<td>Misses 7 attribute(s): - Price range 2, price range 1, kids care, train, indoor, bus, price range 3</td>
</tr>
<tr>
<td>-Service 1</td>
<td>Misses 7 attribute(s): - Price range 2, price range 1, kids care, train, indoor, non-smoking</td>
</tr>
<tr>
<td>-Service 12</td>
<td>Misses 8 attribute(s): - Price range 2, price range 1, kids care, train, parking, price range 3, bus, non-smoking</td>
</tr>
<tr>
<td>-Service 3</td>
<td>Misses 9 attribute(s): - Price range 2, price range 1, kids care, train, indoor, bus, price range 3, parking, nearby</td>
</tr>
<tr>
<td>+Imperfect matches</td>
<td>5 match(es)</td>
</tr>
<tr>
<td>Service 5</td>
<td>Requires 1 additional input(s): - time Misses 1 attribute(s): - Parking</td>
</tr>
<tr>
<td>Service 10</td>
<td>Requires 1 additional input(s): - time Misses 5 attribute(s): - Price range 1, price range 3, parking, nearby, non-smoking</td>
</tr>
<tr>
<td>Service 9</td>
<td>Requires 1 additional input(s): - time Misses 6 attribute(s): - Price range 1, price range 3, indoor, kids care, bus, non-smoking</td>
</tr>
<tr>
<td>Service 8</td>
<td>Requires 1 additional input(s): - time Misses 8 attribute(s): - Price range 1, price range 3, indoor, kids care, bus, train, nearby, parking</td>
</tr>
<tr>
<td>Service 4</td>
<td>Requires 1 additional input(s): - time Misses 9 attribute(s): - Price range 1, price range 3, indoor, kids care, bus, train, nearby, open, non-smoking</td>
</tr>
</tbody>
</table>
6.5 Prototype implementation

This section discusses the implementation of the COSS matching algorithm in the COSS service discovery prototype. The implementation is using Java technology. This sections starts with a general overview of the implemented components followed by a more detailed presentation of each of the components. Figure 42 presents the high-level overview of the COSS prototype.

The service requestor initiates the service discovery by contacting the COSS service discovery server. The server retrieves the request and matches it with the services that are advertised in the service registry. It uses the context registry and possible associated context services to do context-aware service discovery. The COSS prototype can be divided into several subcomponents (see Figure 43) that will be successively discussed.

6.5.1 GUI

The service requestor initiates the service discovery by issuing a request. Therefore, the requestor invokes the GUI component. The GUI component consists of the graphical user interface in which the requestor can specify its request and in which the results of this service discovery are presented. The technology used in the GUI component is Java Servlets and Java JSP pages. The JSP pages are used for the presentation while the servlets are used as logic to retrieve information that has to be displayed in the JSP pages.

The service requestor defines its request in two phases. It starts by specifying the types of services the requestor wants to discover. The second step is the definition of the desired outputs and the inputs he can provide. In addition, he can define some properties he would like the service to have. These phases are handled by the several servlets and JSP’s (see Figure 44).
The GUI is invoked by a standard HTTP request to the Request servlet. This servlet queries the ontology (see 6.1.4) for all subclasses of the ‘Servicetype’ concept. This information is displayed to the user in the RequestJSP.jsp page. We choose to use a tree structure [TRE04] to display the subclass relation of the service types. In this way the semantics of the subclass relation is visualized.

The requestor can then select multiple types of services he wants to discover. The Request2 servlet retrieves the selected types and adds all subclasses (by querying the ontology) of the selected types to the request (e.g. if the user is interested in a music store, he is also interested in a CD store). Furthermore, it traverses the ‘sell’ relation in the ontology to determine which outputs and inputs correspond with the selected type. This information is presented in the Request2JSP.jsp page. Again, a tree is used to visualize subclass relations of the inputs and the outputs. The user can then select inputs, outputs and attributes of the service he wants to discover. This information is used by the Request3 servlet to invoke the matchmaker to discover services. The result of this invocation is displayed in the Result.jsp page (See the following figures for some partial screenshots of the JSP pages in the COSS prototype).
6.5.2 Matchmaker

The matchmaker component is the core of the COSS prototype. This component is the implementation of the matching algorithm, presented in the beginning of this chapter. The matchmaker is invoked by the GUI component. It first retrieves the request and initiates the discovery algorithm (see Figure 48).

The matchmaker then executes the algorithm. It first queries the service registry with a MySQL query for services with the specified service types. The query returns OWL [W3C-5] service advertisements with the specified service types. These advertisements are merged into one ontology model using the Jena framework [JEN04]. First, the requested outputs are expanded with their subclasses (e.g., if a user is interested in music he is also interested in CDs). Then the model is used to query (using RDQL [SEA02] statements) for services that have the requested and expanded outputs. These services are stored and they are queried, one by one, for the inputs they need. If all inputs are provided by the inputs specified in the request, the service is stored as a precise match. If one or more inputs are missing, the algorithm checks, by invoking the context manager, if an input can be provided by a context provider. If this is the case for all missing inputs, the service is stored as a precise match; otherwise, it is stored as an approximate match. In the final step, the two sets of services are ordered by their properties. To do this, two lattice models
(see 6.3.2) are created. The request is added as an object in the models and its requested properties are added as attributes. Then the services are added as objects in the corresponding lattice model. For every service and every property from the request, the property rule is evaluated by invoking the context manager. If the result of this invocation is positive, a relation of the service (object) with the corresponding property (attribute) is made. Finally, the lattice models are used to primary order the services by the number of properties they have and secondly by the physical distance between a service and the requestor (to further demonstrate the applicability of context-awareness In this case, location-aware service discovery).

The results of the ordering are then returned to the GUI that can display the result to the user. In this prototype, we have chosen to implement a graphical user interface to invoke and display the result of service discovery. However, the matchmaker is designed to be deployed as a webservice. The method ‘discoverServices’ executes the matching algorithm that takes a request and returns the precise and approximate matches.

6.5.3 Context manager

The context manager is responsible for the context-awareness aspect of the COSS prototype. It maintains the associations with the context providers and provides means for the matchmaker to retrieve contextual information from the context providers. At this moment, the context manager has two major tasks:

1) Evaluate attribute rules.
2) Check if missing inputs can be provided by the context providers.

The first task is to evaluate attribute rules. The matchmaker can invoke the context manager to evaluate if an attribute holds for a certain service. The context manager then evaluates the corresponding attribute rule (e.g. nearby distance(service,user) < maxdistance). It contacts the corresponding context providers to retrieve needed contextual information. The result (true, false) is returned to the matchmaker that can use it to build the lattice property models.

The second task is to check whether missing inputs can be provided by context providers. Therefore, the matchmaker invokes the context manager with a missing input and the user ID. The context manager then queries the context registry for associations between context providers (providing certain contextual information) and the particular user. The result (true, false) is returned to the matchmaker, which can then decide whether a particular match is a precise or approximate.

6.5.4 Service and context registry

The main source of information for the COSS service discovery algorithm is the service and the context registry. Both are MySQL [MYS04] databases that can be distributed on different servers.

The service registry contains the advertisement of the service. This is the main information with which a request is matched. The structure of the service registry is a MySQL table with an ID, Service URI (without namespace), Service type, Service advertisement (OWL) and Service URL. An example of a service registry record is given below.

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>URI</th>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
</table>
The context registry contains the association of a user with a context provider. It is used to determine if missing inputs can be implicitly provided by a context provider. The structure of the context registry is a MySQL table with an ID, User ID, Context ID and Context provider service URL. The context ID is corresponding with the ontology. An example of a context registry record is given below.

Table 6: Context registry record

<table>
<thead>
<tr>
<th>ID</th>
<th>UserID</th>
<th>ContextID</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Tom</td>
<td>Location</td>
<td><a href="http://foo.bar/LocProv">http://foo.bar/LocProv</a></td>
</tr>
</tbody>
</table>
7 Approach evaluation

The main goal of this research was to improve service discovery by creating a new service discovery approach. To be able to conclude that COSS is an improvement or deterioration, it has to be evaluated. This chapter presents the result of this evaluation of the developed COSS service discovery approach. It makes use of the developed prototype that was discussed in the previous chapter.

This chapter is structured as follows: section 7.1 discusses the evaluation of the COSS prototype. It shows the working of the prototype and discusses the significance of the distinguished issues from the problem analysis (first part of this report) for the COSS service discovery approach. Section 7.2 analyzes the COSS approach by calculating precision and recall rates. Section 7.3 briefly discusses the result of the evaluation.

7.1 COSS prototype evaluation

To be able to evaluate COSS, we developed a prototype. This prototype should demonstrate the improvement of the quality of the discovery result of our approach compared to discovery results of current approaches (syntactic methods). To illustrate the functionality of the prototype, this section will present a short prototype run. The result of this run will also be used in the second part of this section to analyze the consequences of the application of our approach for the issues discussed in the problem analysis.

7.1.1 Working of the prototype

Consider a tourist who is searching for a store that sells music. He takes his mobile phone and he browses to the COSS server. There he can select the service types he is looking for by browsing the service type tree (SellingService -> Shop -> Musicshop). If he does not want to select a service type, he can also go to the next step by submitting nothing. Then all types are selected. In the second step only the relevant outputs (e.g. a music shop sells music products and not food) and relevant inputs are displayed. He can select multiple outputs he wants the service to have, and he can select multiple inputs he can provide. Furthermore, he can select some attributes he would like the service to have. For instance, the service has to be open and nearby. Furthermore, it must have a parking space and must sell product with a price until 200 euro. After he specified this information by simply clicking checkboxes, the matcher starts matching the advertised services with the user’s request and returns the result to the user (see Figure 49).

![Figure 49: Service discovery result of the tourist’s request](image-url)
The top of the resulting JSP page (see Figure 49) summarizes the request of the user. The tourist has searched for a Musicshop that has as output ‘Music’ and he can provide payment by ‘Cash’ or by ‘Card’. Furthermore, he would like the services to be ‘nearby’, ‘open’, have ‘parking’ spaces and sell goods until 200 euro (i.e. range 1 & range 2). The bottom of the page presents the matched services. First, it displays the precise services (services that have the specified outputs) and orders them by the number of attributes they have. If the number of attributes is equal, the distance between the user and the service is the ordering criterion. The second section displays the approximate matches (services that must have certain inputs, which are not provided by the user, to provide the requested outputs). Again, these services are ordered by the number of properties they have and then the distance between the user and the service. The user can then select a particular match and view the match information (see Figure 50).

![Figure 50: Match information of CD_gigant](image.png)

The user can view the reason a match is rated precise or approximate and which inputs, outputs and properties are missing or used. Based on this detailed information and the information from the previous screen, the user is able to invoke a service that is relevant to his request.

The next box presents generated information from the matchmaker that further elaborates the internal working of the matchmaker (comments on this information are prefixed with //):

```plaintext
Received a request ... matching ...

// This is a summary of the request received by the matchmaker from the user.
----------- COSSrequest: Tom-----------
Requested service type = [Musicshop]
Requested outputs = [Music]
Requested inputs = [Cash, Card]
Requested properties = [nearby, open, parking, pricerange1, pricerange2]
---------------------------------

// In this first step the service registry is queried for services with the
// requested types.
*** 1 *** Query the registry for service with the requested type.

// The servicetypes are first expanded (subclassrelation down). In this case
// music shop is the lowest node in the service type ontology so the types
// are not expanded.
// For instance, when the selected service type would be Sellingshop the
// expanded service types would be [Sellingshop, Foodshop, Musicshop,
// Souvenirshop] (see the COSS ontology)
Service with types (expanded) [Musicshop] are queried.

// In the second step the service registry is queried for services that have
// the requested outputs.
*** 2 *** Query the registry for services with the requested outputs.
```
// Again the outputs are first expanded by traversing the sell relation and
// the possible subclass relation (down).
Expanded outputs: [Music, CD, DVD, LP]

// These services have the requested outputs.
Services with the requested outputs:
[http://www.telin.nl/wasp/coss#Van_Leest,
 http://www.telin.nl/wasp/coss#Free_Record_Shop,
 http://www.telin.nl/wasp/coss#CD_gigant,
 http://www.telin.nl/wasp/coss#LP_gigant]

// The third step checks if the services that have the requested outputs need
// certain inputs.
*** 3 ** Query every service for their needed inputs and check if they can be
// provided.

// First the user inputs are expanded by traversing the subclass relations
// (up).
Expanded user inputs: [Payment, Cash, Card]

// For every service is checked if it needs inputs.
* Matching service: Van_Leest
Needed inputs for this service: [PIN, Cash, Location]
Missing inputs: [PIN, Location]
// There are two missing inputs that can not be explicitly provided by the
// user. For each of these missing inputs is checked if it can be provided by
// a context provider.
Inputs: PIN CANNOT be implicitly provided.
Inputs: Location CAN be implicitly provided.
// One input can be provided but one input cannot -> approximate match
This service is an approximate match.

* Matching service: Free_Record_Shop
Needed inputs for this service: [PIN, Cash]
Missing inputs: [PIN]
Inputs: PIN CANNOT be implicitly provided.
This service is an approximate match.

* Matching service: CD_gigant
Needed inputs for this service: [Cash, Location]
Missing inputs: [Location]
Inputs: Location CAN be implicitly provided.
// The missing inputs can be implicitly provided by a context provider this
// means the service is rated precise
This service is an indirect precise match.

* Matching service: LP_gigant
Needed inputs for this service: [Cash]
This service is a direct precise match.

*** Classification before ordering.......  
-- Precise services: [CD_gigant, LP_gigant]  
-- Approximate services: [Van_Leest, Free_Record_Shop]

// The final step orders the two sets of matches.
*** 4 ** Order sets of services using concept lattices.
// Create two lattice model, one for the precise matches and one for the
// approximate matches.
// The property rules are evaluated and when true a ‘cross’ is added to the
// lattice model.
Precise services lattice:
7.1.2 Reflection on the problem analysis versus the COSS prototype

The problem analysis has discussed some issues with current service discovery mechanisms that limit the quality of their result. This section presents a reflection of the issues discussed in the problem analysis for the COSS prototype. In general, the problem analysis distinguished the following issues that will be discussed successively:

- Homonyms (see chapter 2).
- Synonyms (see chapter 2).
- Contextual information (see chapter 2 and 3).
- Common understanding (see chapter 2 and 3).
- Semantic matchmaking (see chapter 2, 3 and 4).
- Application in the WASP platform (see chapter 5).

Homonyms

One of the issues with syntactic service discovery is that they based their matching on keywords. These keywords can be syntactically equivalent but semantically different (e.g. ‘order’ in the sense of proper arrangement and ‘order’ in the sense of a commercial document). Such keywords are called homonyms. If the requestor wants to retrieve a commercial order then also services that provide ordering (arranging) services are returned (false-positive results), which decreases the quality of the discovery result.

In COSS, the issues with homonyms are overcome by using an ontology in which unique concepts are specified. Therefore, homonyms have unique concepts that differentiate them. Furthermore, a concept has a label that consists of human readable text. In COSS, homonyms can have the same labels but have different concepts. In addition, the COSS ontology provides subclass relations that give additional information on the semantics of a concept. See the picture below for an example of a homonym (i.e. Shop) in COSS.

![Figure 51: Homonyms in COSS](image)

The word ‘Shop’ can have different meanings. It can be an entity that sells or manufactures products, or it can be an entity that rents out products. The user can still use the familiar keyword...
‘Shop’ in its request. Furthermore, the user can derive from the additional information (subclass relation in the tree structure) which shop is the correct concept to specify its request.

**Synonyms**

Similarly, to the issues with homonyms, there are issues with synonyms. There can be keywords that are syntactically different but semantically equivalent (e.g. order and purchase). Such keywords are called synonyms. If the requestor wants to ‘order’ something services that specify ‘purchase’ are not returned (false-negative result). This limits the quality of the result.

In COSS, the issues with synonyms can be easily overcome by using an ontology in which a synonym relation is defined. By specifying that a particular concept is a synonym of another concept, the COSS algorithm can consider this. COSS can display two synonyms as one concept to the user while they are specified as two unique concepts in the ontology. The user can then still select its familiar synonym concept without influencing the quality of the discovery result.

**Contextual information**

Current service discovery mechanisms do not consider contextual information in their functionality. This information is often available (e.g. built-in time function in a mobile phone) but not used. COSS uses contextual information to complete the user request by giving him the opportunity to specify properties (e.g. nearby, open) he would like the service to have. Furthermore, COSS completes the missing inputs (inputs not provided by the requestor) by searching for context providers that implicitly can provide these inputs. Finally, COSS uses the location of the services and the user to order matches by distance. It thereby gives extra information to the requestor that can help him to decide which service to invoke.

As said in section 3.4, context has many alternative representations (e.g. physical location can be denoted as ‘place’, ‘location’ or ‘position’ and can have different formats lat-long, x-y). COSS overcomes this issue by specifying context information with concepts from the ontology. These concepts are shared by and known to all parties. In this way, the alternative representations of contextual information are combined in unique concepts and do not form a source for mismatches.

To be able to do context-aware service discovery, COSS assumes that it has knowledge of the association between context providers and requestors. This implies that a unique userID is necessary to identify users. Furthermore, when context providers advertise their context services to COSS, they have to provide the associations of users that are associated with it. How this has to be done and how to keep these associations up to date is considered future work.

**Common understanding**

A fundamental aspect of matching is the information that has to be matched. If the representation of this information is not correct (e.g. synonyms or homonyms) the discovery result is poor. Current service discovery mechanisms do not create a common understanding on services. This can lead to mismatches because requestor and service provider talk in a different ‘language’.

COSS overcomes this issue by defining a general ontology that creates a common understanding (common language) between the service requestors, service providers and context providers. COSS is the central entity that has access to this ontology. At this moment, the ontology is not enforced to the service providers and context providers. However, by the central nature of the COSS service discovery mechanism enforcement can be applied. How to enforce such an ontology is considered future work.
Semantic matchmaking

Current discovery mechanisms are syntactic matching mechanisms (i.e. they match on keywords). As discussed, these mechanisms have limited quality. An envisioned approach to improve the service discovery quality is semantic matchmaking. So, match service and request on meaning not on keywords.

Complete semantic matchmaking is still a very difficult task. COSS implements a very limited form of semantic matchmaking that gives higher quality discovery results. The basis of COSS semantic matchmaking is the ontology. It defines unique concepts and relations between these concepts. At this moment, COSS uses only subclass relations. This is used for types, outputs and inputs. By traversing these relations for concepts in the request, the request gets more complete (e.g. when a user is interested in output ‘music’ he is also interested in output ‘CD’, ‘LP’ and ‘DVD’).

A challenge is to apply more types of relations and then to adapt the COSS matching algorithm to traverse these relations in a meaningful manner to retrieve even higher quality results. This is considered future work.

Application in WASP

Chapter 5 introduced the WASP platform. It concluded that there are some issues that limit the quality of its current service discovery approach and that this creates opportunities for a new service discovery approach. COSS takes some of these opportunities to produce higher-quality service discovery results. First, COSS implements a much more sophisticated service discovery approach. It performs a limited form of semantic matchmaking, it uses ontologies to create a common understanding and it performs context-aware service discovery. These aspects improve service discovery as discussed in the previous sections. Secondly, COSS enables the use of more contextual information by specifying and using attributes in its functionality. For example, time (contextual information) is added to COSS. Finally, COSS is designed to function as a webservice. This means that the COSS service discovery mechanism can easily be plugged into the WASP platform without drastic changes to the platform.

7.2 Precision and Recall evaluation

This section evaluates the COSS approach by calculating precision and recall rates. Consider a set of relevant services (R) within a big set of advertised services (A) \((R \subseteq A)\). We define:

- **Recall**: The number of relevant items retrieved, divided by the total number of relevant items in the collection. The highest value of recall is achieved when all relevant items are retrieved.
- **Precision**: The number of relevant items retrieved, divided by the total number of items retrieved. The highest value of precision is achieved when only relevant items are retrieved.

For this evaluation, we created a registry with thirty advertised services. We queried this registry with multiple requests using the COSS prototype. With the result, we calculated the recall and precision rates. We compared these calculations with queries using current syntactic mechanisms (i.e. keyword and table based approaches).
Query 1: “Discover all services that sell Music”
This means that in COSS all service types are selected and that “Music” is selected as output.

<table>
<thead>
<tr>
<th></th>
<th>Relevant services</th>
<th>Keyword-based mechanisms</th>
<th>Table based mechanisms</th>
<th>COSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery result (services)</td>
<td>19</td>
<td>6 relevant</td>
<td>6 relevant</td>
<td>19 relevant</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>-</td>
<td>75%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>-</td>
<td>32%</td>
<td>32%</td>
<td>100%</td>
</tr>
</tbody>
</table>

In the set of advertised services, there are 19 relevant services that sell Music. The keyword-based services retrieve only the services that have the word “Music” in their advertisements (8). Two of these services have “Music” as their input and not as output so these matches are irrelevant. The table-based mechanisms do not select the services with input “Music” and therefore does not have irrelevant matches. COSS selects only the services with output “Music” and the subclasses of “Music” like services that sell CD, DVD and LP. Therefore, it has full precision and full recall.

Query 2: “Discover all Music shops that sell Music”
This means that in COSS only “Musicshops” are selected and that “Music” is selected as output.

<table>
<thead>
<tr>
<th></th>
<th>Relevant services</th>
<th>Keyword-based mechanisms</th>
<th>Table based mechanisms</th>
<th>COSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery result (services)</td>
<td>4</td>
<td>2 relevant</td>
<td>2 relevant</td>
<td>4 relevant</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>-</td>
<td>34%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>-</td>
<td>50%</td>
<td>50%</td>
<td>100%</td>
</tr>
</tbody>
</table>

In the set of advertised services, there are 4 relevant services that are both “Musicshop” and sell “Music”. Again, the keyword-based mechanisms only return the services that have “Music” in their advertisement (8). They ignore the requested service type “Musicshop”. The table-based mechanisms consider the service type but miss the services with outputs of subclass “Music”. COSS incorporates the type and the outputs (plus subclass outputs) and therefore has full recall and precision.

Query 3: “Discover all Shops that sell “Music”
Shop has multiple homonyms like Manufacturingshop and Rentalshop. The requestor can decide in COSS which shop he wants to discover by traversing the service type tree. This means that in COSS only “Sellingshops” are selected and that “Music” is selected as output.

<table>
<thead>
<tr>
<th></th>
<th>Relevant services</th>
<th>Keyword-based mechanisms</th>
<th>Table based mechanisms</th>
<th>COSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery result (services)</td>
<td>11</td>
<td>2 relevant</td>
<td>2 relevant</td>
<td>11 relevant</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>-</td>
<td>28,7%</td>
<td>28,7%</td>
<td>100%</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>-</td>
<td>50%</td>
<td>50%</td>
<td>100%</td>
</tr>
</tbody>
</table>

In the set of advertised services there are 11 services that are “Sellingshops” and that sell “Music”. The keyword-based approaches follow the same pattern as the previous query and return the services with Music in their advertisement. The table-based mechanisms perform worse now because the ambiguity (homonyms) of the word “shop”. They return all the shop even if they are
manufacturing or rental shops. COSS provides means for the user to select the right shop and therefore has full recall and precision.

**Query 4: “Discover all shops that sells Compact Discs”**
This means that in COSS “Sellingshop” is selected as service type and that “Compact Disc” is selected as output.

<table>
<thead>
<tr>
<th></th>
<th>Relevant services</th>
<th>Keyword-based mechanisms</th>
<th>Table based mechanisms</th>
<th>COSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery result (services)</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>7 relevant</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>-</td>
<td>0%</td>
<td>0%</td>
<td>100%</td>
</tr>
</tbody>
</table>

In the set of advertised services, there are 7 relevant services that are a Sellingshops and sell “Compact discs”. Because compact disc is a synonym of CD, COSS matches also services with output CD. The other mechanisms cannot infer that CD is semantically equal to Compact disc and therefore do not return relevant matches.

**Query 5: “Discover all shops that sell Music and that are nearby (250m)”**
This means that in COSS “Sellingshop” is selected as service type and that “Music” is selected as output. Furthermore, “nearby” is selected as service property.

<table>
<thead>
<tr>
<th></th>
<th>Relevant services</th>
<th>Keyword-based mechanisms</th>
<th>Table based mechanisms</th>
<th>COSS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Discovery result (services)</td>
<td>5</td>
<td>1 relevant 7 irrelevant</td>
<td>1 relevant 7 irrelevant</td>
<td>5 relevant</td>
</tr>
<tr>
<td>Precision (%)</td>
<td>-</td>
<td>15%</td>
<td>15%</td>
<td>100%</td>
</tr>
<tr>
<td>Recall (%)</td>
<td>-</td>
<td>20%</td>
<td>20%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Both keyword-based and table-based mechanism cannot handle context in their discovery process. Therefore, nearby is ignored and many irrelevant services are returned. COSS evaluates all matches on the nearby rule and orders them by this property.

### 7.3 Discussion
The previous sections showed that by using the COSS matching algorithm the quality of the service discovery result improves. Section 7.1 indicated the theoretical improvements while section 7.2 supported these statements with a case study using the COSS prototype. The recall and precision rates from COSS, where with all queried requests higher than the rates from the keyword- and table-based approaches. This means that the user receives a better subset of services that are relevant for him and furthermore that the user receives fewer services that are irrelevant for him.

Some remarks have to be made on the form of this quantitative analysis. Due to the fact that this research is conducted in a delimited application domain, the set of advertised services, query and ontology are highly adapted to each other. Therefore, COSS retrieves 100% matches. In reality, this correlation will not be that high. Therefore, this analysis does not indicate improvements in all possible situations but merely gives an overview of certain situation in which the COSS approach improves the quality of the service discovery result. The development of COSS is just an improving step in the quest for high-quality service discovery results.
8 Conclusions

This chapter presents the conclusions of this research. Furthermore, it indicates some areas for future research. This chapter is structured as follows: Section 8.1 presents general conclusions and summarizes the main contributions of this research and Section 8.2 identifies future research.

8.1 General conclusions

We have proposed a novel service discovery approach (called Context-aware, Ontology based, Semantic Service discovery (COSS)) that improves the quality of the service discovery result. Our efforts towards this approach included (i) the identification of limiting factors for the quality of service discovery in current approaches (ii) the study of the WASP platform as a good opportunity to apply our approach (iii) the identification of types of service discovery matches and the definition of matching degree (iv) the development of a novel service discovery approach that improves the service discovery result (v) the development of a prototype as an aid for evaluating the novel approach and finally (vi) the evaluation of the novel approach.

The first part of this thesis presented a problem analysis on aspects of current service discovery mechanisms that limit the quality of the discovery result. This part was divided into four sub parts that analyze the current service discovery mechanisms.

The first analysis discussed traditional service discovery mechanism like CORBA naming and trading, and UDDI. It indicated that although these mechanisms were capable of providing mechanisms for the service requestors to find services, the quality of the results was low. Traditional service discovery approaches perform syntactic matching, that is, they retrieve services descriptions that contain particular keywords from the user’s query. This often leads to poor discovery results, because the keywords in the query can be semantically similar but syntactically different (homonyms), or syntactically similar but semantically different from the terms in a service description (synonyms). Another drawback of the traditional service discovery mechanisms is that the query-service matching score is calculated considering only the keywords from the user’s query and the terms in the service descriptions. Thus, regardless of the context of the service user and the context of the services providers, the same list of results is returned in response to a particular query.

The second analysis discussed context-aware service discovery, which is a mechanism that uses contextual information in its functionality. Such mechanisms provide theoretically higher quality of result because the request query can be completed with extra contextual information such as user location or weather conditions etc. Furthermore, implicit contextual information can be used as inputs for services.

The third analysis discussed ontology-based service discovery. Such mechanisms use ontologies to create a common understanding on services and context concepts. Common understanding on services in service discovery is necessary because many independent parties are involved in the discovery process (service requestor, service provider and context provider). They all have different knowledge on services. Matching requests with advertisements that do have shared knowledge is very difficult and often leads to a poor quality result. Furthermore, the use of ontologies enables semantic reasoning. Ontologies define concepts and their relations. By inferring and traversing the defined relations, higher quality matching results can be achieved.
The final analysis presented the WASP platform. This platform provides a supporting platform for mobile context-aware applications. It presents a good opportunity to apply our novel service discovery approach: (i) its goal is to perform intelligent service discovery and (ii) it support contextual information and in a limited fashion, ontologies are applied. The WASP service discovery mechanism is very limited and therefore could benefit greatly from the COSS approach.

The second part of this thesis presented the development and implementation of the novel service discovery approach (COSS). This approach incorporates the discussed issues of the problem analysis to improve the quality of discovery results. The major aspects applied in this novel approach are (i) context-awareness, (ii) ontologies and (iii) semantic reasoning.

First, COSS collects and uses contextual information to complete the user’s query. It provides opportunity for the user to specify attributes which he would like the service to have, such as nearby, open etc. In the rating of the service match, these rules are evaluated and used to order several service matches. This results in a higher precision match. Furthermore, COSS collects and uses contextual information to complete required inputs for services that the user does not explicitly provide. This results in higher recall rate because services that formerly could not be returned are now available.

Secondly, COSS uses ontologies to create common understanding. This way, service requestor, service provider and context provider have an unambiguous understanding on services. This overcomes most of the issues with synonyms and homonyms. Furthermore, by using ontologies the nature of contextual information that it has multiple representations is countered. One common representation is specified in the ontology to which the involved parties commit.

Thirdly, by using the ontologies semantic reasoning is performed. The ontology defines concepts and their relations. COSS reasons on one type of relation (subclass relation) and therefore can infer matches that would not be retrieved otherwise.

For demonstration and evaluation purposes, the approach is implemented in a prototype. The prototype provides a graphical user interface (web) for the user to specify its request. The results are ordered in the two distinguished match types: precise (the services that are classified precise provide the requested outputs and the user provides enough inputs to invoke the service) and approximate matches (the services that are classified approximate provide the requested outputs but the user does not provide enough inputs to invoke the service). Furthermore, they are ordered primarily by the requested properties they have and then by the distance between the user and the service.

The prototype is used to evaluate the novel service discovery approach. Besides giving a proof of concept by demonstrating the prototype, some quantitative evaluations have been done. Some queries have been executed and the precision and recall rates have been calculated. On all accounts, COSS had better recall and precision rates than current service discovery mechanisms. This means that the user receives a subset of services that are more relevant for him. Furthermore, the user receives fewer services that are irrelevant for him. This demonstrates that COSS has improved the quality of the service discovery result, which was the main goal of this research.
8.2 Future research

Although the COSS approach is a major improvement for the quality of the service discovery result, before it can be used to its full potential, some challenging future research areas have to be addressed:

- Considerations on scalability and performance requirements. In this report, we have taken a limited amount of advertised services in the registry and a simple ontology. In reality, the number of advertised services grows enormously and the ontology must be capable of expressing more than just the small tourist scenario used in this report. Therefore, research has to be conducted on the performance and scalability of the COSS approach. A possible direction could be distributed service discovery to separate computational complexity.

- Research on ontology development and maintenance. Chapter 4 reports that ontology development and ontology maintenance are difficult issues. Due to the central nature of the COSS approach in a services platform, ontology development and maintenance shall probably be performed by the service platform (i.e. broker role). How to do this in an efficient and consistent manner has to be researched.

- The developed approach and prototype do not force the service advertisement to be consistent with the ontology. Therefore, to assure the proper applicability of the COSS approach, research has to be performed on the enforcement of ontologies. Again, due to the central nature of the COSS approach in a service platform, it is likely that this task is performed by the service platform.

- Another aspect that could improve the quality of the service discovery result is to let the user weigh the properties he likes the service to have (i.e. nearby = 4 while open = 3). This may results in a better ordering of services.

- Besides explicitly weighing the properties by the user (like discussed in the previous point), this could also be done automatically by a personalization engine. Some efforts in this direction have been investigated in [SET04].

- Our research focused on the matching of a service request and a service advertisement. How this request is retrieved from the user is not considered. A major research area is how a user can express his need to a request. The area of human computer interaction can contribute to provide a higher quality requests that may lead to higher-quality service discovery results.

- The usage of contextual information is a major aspect in our approach. We assume that this information is available, complete and unambiguous. However, in reality this is not often the case. Therefore, research has to be conducted on how to handle missing, redundant, or incomplete contextual information.

- Research other ontology relations for enhanced semantic reasoning. Our approach supports only the subclass relation, which is a transitive relation. In reality, there are many more types of relations. For example, non-transitive relations such as a ‘similarity’ relation. This relation could indicate that concepts are similar but not equal. When considering more types of relations better matches may be derived, which may lead to higher-quality service discovery results.
References

[EBB02] P. Ebben et al., “Blueprint and design of the WASP application platform”, WASP/D2.2, December 2002
[GUI03] D. McGuinness, “Ontologies come of Age”, ch. 6 from Spinning the Semantic web –


[KHE02] M. Khedr, “Enhancing service discovery with Context Information”, ITS’02, Brazil, 2002


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>[SET04]</td>
<td>M. van Setten et al.,”Context-Aware Recommendation in the Mobile Tourist Application Compass, 2004</td>
</tr>
<tr>
<td>[SUN02]</td>
<td>V. Sundramoorthy et al., “Service discovery at home”, 2002</td>
</tr>
<tr>
<td>[WEI01]</td>
<td>P. van der Weide, “Information Discovery”, April 2001</td>
</tr>
</tbody>
</table>