



National Technical University of Athens

SCHOOL OF ELECTRICAL AND
COMPUTER ENGINEERING

COMPUTER SCIENCE DIVISION

**Discovery and Integration of Data and Services
in the Semantic Web**

PhD Thesis

of

Dimitrios Skoutas

Dipl. Electrical and Computer Engineering (2003)

Athens, December 2008



NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
COMPUTER SCIENCE DIVISION

Discovery and Integration of Data and Services in the Semantic Web

PhD Thesis

of

Dimitrios Skoutas

Dipl. Electrical and Computer Engineering (2003)

Supervising Committee: T. Sellis
Y. Vassiliou
A.-G. Stafylopatis

Approved by the Examination Committee, 5th December 2008.

...
T. Sellis
Prof. NTUA

...
Y. Vassiliou
Prof. NTUA

...
A.-G. Stafylopatis
Prof. NTUA

...
N. Koziris
Assoc. Prof. NTUA

...
K. Kontogiannis
Assoc. Prof. NTUA

...
M. Koubarakis
Assoc. Prof. UoA

...
P. Vassiliadis
Assist. Prof. UoI

Athens, December 2008

...

Dimitrios Skoutas
Electrical Engineer, PhD

© 2008 - All rights reserved

ABSTRACT

The Web constitutes a universal repository providing a huge amount of information in a variety of topics and formats. At the same time, the number of users has increased significantly, their participation has become more active, and their needs are more complex. Thus, new trends arise, emphasizing on the need for integration and collaboration. To address these new challenges, a lot of research efforts have been devoted to the transition to the Semantic Web, which will enhance the current Web with formal and explicit metadata, promising to facilitate interoperability and to increase the automation in searching, managing, and sharing information.

In this direction, this thesis studies the problem of searching for relevant services and data on the Semantic Web, as well as integrating information from heterogeneous sources to meet specific needs and requirements. First, we study the problem of Web service discovery. We propose a similarity measure for comparing service descriptions, using the semantic information conveyed by the ontologies used to annotate these descriptions. We also develop techniques, drawing from concepts related to skyline queries, for ranking available services under diverse user preferences and multiple matching criteria. Then, we study the search of services and data in distributed environments, considering peer-to-peer networks where the available resources are semantically annotated. We propose an approach for efficient and progressive search of services in a structured peer-to-peer overlay network, and a method to facilitate the sharing of structured data in an ontology-enhanced peer data management system. Finally, we propose techniques to facilitate the conceptual design of Extract-Transform-Load processes, which are critical processes for reconciling information from several heterogeneous sources. These techniques also rely on the use of ontologies to identify correspondences, conflicts, and transformations between the source and target specifications.

ACKNOWLEDGEMENTS

There are several people who helped and supported me during the work on this thesis. First of all, I am grateful to Prof. Timos Sellis for his support and guidance, without which this thesis could not have been accomplished. His inspiration and encouragement, as well as his suggestions and corrections on several issues, were most valuable throughout these years, especially during the most critical periods, at the beginning of my work and during the last months towards its completion. I also need to thank Dr. Alkis Simitsis for his patience and support throughout all the stressful moments during this work, for his valuable comments and advice on various issues, for his constant urging and focus on detail, and for always finding the time to discuss and review all the parts of the work presented in this thesis. I also want to thank Dimitris Sacharidis for the very fruitful collaboration we had, which helped me to significantly improve the quality of this work. In addition, I had the pleasure to collaborate and discuss some of the issues presented in this thesis with Verena Kantere. The committee members, and especially Prof. Panos Vassiliadis, helped me with their constructive comments to improve the final version of the thesis.

I also want to thank all the members of the Knowledge and Database Systems Laboratory for our discussions and for creating a very friendly working environment. The Institute for the Management of Information Systems supported me with a grant during the last period of my work. I am very grateful to them, and for having the chance to work in such a pleasant and inspiring environment.

Finally, I would like to thank my family for their support and patience throughout all these years.

*Dimitrios Skoutas
Athens, December 2008*

Contents

1	Introduction	1
1.1	Challenges and Contributions	2
1.1.1	Service Discovery and Selection	3
1.1.2	Search for Services and Data in P2P Networks	6
1.1.3	Design of ETL processes	8
1.2	Thesis Outline	11
2	Web Service Matchmaking and Ranking	13
2.1	Preliminaries	14
2.1.1	Ontologies on the Web	14
2.1.2	Web Service Description	16
2.2	Related Work	19
2.2.1	Web Service Discovery	19
2.2.2	Skyline Computation	21
2.2.3	Data Fusion	22
2.3	A Similarity Measure for Semantic Web Services	23
2.3.1	Example	23
2.3.2	The Similarity Measure	25
2.3.3	Properties of the Similarity Measure	34
2.4	Selection of Services with Skyline Queries	35
2.4.1	Skyline Services	35
2.4.2	Selection of the Best Candidates	39
2.4.3	Experimental Evaluation	45
2.5	Ranking of Services under Multiple Criteria	47
2.5.1	Motivation and Problem Definition	47
2.5.2	Algorithms	51
2.5.3	Experimental Evaluation	57
2.6	Summary	64
3	Service and Data Selection in Peer-to-Peer Networks	65
3.1	Related Work	65
3.1.1	P2P Service Discovery	65
3.1.2	Semantics-based P2P Data Sharing	66
3.2	Service Discovery in Structured P2P Networks	67
3.2.1	Encoding Service Descriptions	67
3.2.2	Indexing Service Descriptions	70
3.2.3	Managing Services in the P2P Overlay	74
3.2.4	Experimental Evaluation	76

3.3	Ontology-based Data Sharing in a PDMS	78
3.3.1	Problem Description	78
3.3.2	Comparison of Peer Schemas	81
3.3.3	Comparison of Rewritten Queries	85
3.3.4	Extending to Multiple Ontologies	88
3.4	Summary	91
4	Ontology-based Design of Extract-Transform-Load Processes	93
4.1	Related Work	93
4.1.1	Conceptual models for ETL processes and DWs	94
4.1.2	Data Integration and Semantic Schema Matching	95
4.1.3	Mashups	96
4.2	Conceptual Design of ETL Processes	96
4.2.1	Datastore Representation	96
4.2.2	Application Ontology Construction	98
4.2.3	Semantic Annotation of Datastores	103
4.2.4	Identification of ETL Operations	104
4.2.5	Generation of Reports	109
4.3	ETL Design through Graph Transformations	115
4.3.1	General Framework	115
4.3.2	Graph Transformations	117
4.3.3	The Type Graph	118
4.3.4	The Transformation Rules	120
4.3.5	Creation of the ETL Design	126
4.3.6	Illustrative Example	128
4.4	Summary	129
5	Conclusions and Future Work	131
5.1	Conclusions	131
5.2	Future Work	132
	Bibliography	135

List of Figures

1.1	(a) The main axes of our approach; (b) Discovery and integration with ontologies, Web services and ETL processes	2
2.1	The Semantic Web technology stack.....	15
2.2	Overview of Web service discovery	17
2.3	A sample ontology snippet for the motivating example.....	24
2.4	Illustration of (a) static and (b) dynamic skylines.....	36
2.5	Experimental evaluation on real and synthetic data.....	45
2.6	An illustrative example	48
2.7	Search space for <i>TKDD</i> , <i>TKDG</i> , and <i>TKM</i>	53
2.8	Recall-Precision graphs	59
2.9	Effect of parameters under low (left graph of each pair) and high (right graph of each pair) variance <i>var</i>	61
2.10	Effect of <i>corr</i> under low (left) and high (right) variance <i>var</i>	63
3.1	(a) A sample ontology fragment, (b) A service request (<i>R</i>) and three service advertisements (<i>S</i> ₁ , <i>S</i> ₂ , <i>S</i> ₃), (c) Intervals assigned to ontology concepts.....	68
3.2	Interval based search.....	70
3.3	R-trees example	72
3.4	Illustrative SpatialP2P overlay	75
3.5	Search regions	75
3.6	(a) Recall-precision curve, (b) Precision and Success at <i>k</i> (<i>P@k</i> , <i>S@k</i>)	77
3.7	Search cost for: (a) centralized registry, (b) P2P registry.....	77
3.8	Unstructured network of semantic peers with (a) single ontology (b) multiple ontologies.....	80
3.9	A sample ontology and a snippet of the corresponding XML representation	81
4.1	The ontology graph for the reference example	103
4.2	Semantic annotation of the datastores	104
4.3	Identified transformations for the reference example	109
4.4	(a) The type graph and (b) a sample instance graph.....	119
4.5	Rules for inserting LOAD operations in the presence of direct relationship.....	121
4.6	Rules for inserting LOAD operations via <i>isa</i> link	122
4.7	Rules for inserting FILTER operations	123
4.8	Rules for inserting CONVERT operations	123
4.9	Rules for inserting EXTRACT operations	124
4.10	Rules for inserting CONSTRUCT operations	124

4.11	Rules for inserting SPLIT operations.....	125
4.12	Rules for inserting MERGE operations.....	126
4.13	“Clean-up” rule.....	126
4.14	Example	128
4.15	Output of the graph transformation process	130

List of Tables

2.1	Service request and advertisements	24
2.2	Domain of an answer to a search	25
2.3	Recall and precision based on the class hierarchy	27
2.4	Recall and precision based on common properties	28
2.5	Recall and precision considering property hierarchy	29
2.6	Recall and precision including value restrictions	30
2.7	Recall and precision based on properties, including hierarchy and restrictions	30
2.8	Preconditions and effects for the motivating example	32
2.9	Recall and precision values for the preconditions and effects	32
2.10	Recall and precision combining class hierarchy and properties	32
2.11	Recall and precision for multiple input/output parameters	33
2.12	Recall and precision combining input, output, preconditions and effects	34
2.13	Matching service I/Os	38
2.14	Illustrative example	40
2.15	Bitmap representation	40
2.16	Dominance check for S_4	40
2.17	Determining the skyline services	41
2.18	Modifying service parameter	44
2.19	Evaluation of retrieval accuracy	46
2.20	Algorithm $TKDD$	54
2.21	Algorithm $TKDG$	55
2.22	IR metrics for all methods	60
2.23	Parameters and examined values	63
3.1	Types of match using the intervals based encoding	70
3.2	Algorithm for index-based service matchmaking	73
3.3	Algorithm for progressively returning matches	73
3.4	OWL constructs and notation used	79
3.5	Semantic annotation of peer schemas	81
3.6	Different cases of subsumption relationship between two restrictions R_1 and R_2	83
3.7	Algorithm for measuring the semantic similarity for rewritten queries	86
4.1	Sample source and target schemas	98
4.2	OWL features used in our approach	100
4.3	Notation used for the ontology graph	101
4.4	Algorithm for graph representation of the ontology	102

4.5	Algorithm for creating the definition for an internal labeled node of the datastore graph	105
4.6	Generic types of conceptual transformations frequently used in an ETL process	106
4.7	Algorithm for provider node selection	107
4.8	Algorithm for deriving transformations between two labeled leaf nodes	108
4.9	A set of provided built-in functions	110
4.10	A set of provided built-in macros for datastore annotations	111
4.11	A set of provided built-in macros for generic ETL operations	112
4.12	Source and target schemata for the example	128

Chapter 1

Introduction

Nowadays the Web has been established as a universal information repository, providing unprecedented wealth and diversity of information. It has penetrated essentially every aspect of our every day lives, revolutionizing the way people educate, work, communicate, inform, and entertain. Over the past years, the Web has evolved from a collection of static pages, displaying the same information to all users, at all times and in all contexts, to an interactive and dynamic environment, where the contents of the pages change in response to different requests and conditions. At the same time there has been an enormous growth of the amount of Web users, as well as of the diversity of their technical background and needs. Yet, the full potential of the Web is still to be explored, while current trends and people needs seem to point to a direction where *integration* and *collaboration*, based on *semantics*, lie at the heart of the new landscape.

The term Web 2.0 has been coined to refer to these new trends in the use of the Web, characterized primarily by the active participation of users in online communities. Users share information, collaborate, and contribute by creating new content and/or creating tags to annotate and categorize existing content. A characteristic example is mashups, a new type of Web applications where third-party content is drawn through public interfaces and it is combined in a way to increase its value for the end users. Still, since social tagging operates without terminological control, it can often be unreliable or even inconsistent. On the other hand, researchers pursue the semantic enhancement of Web content, towards the so-called Semantic Web, through the development of standards and tools, that will allow to formally describe the concepts and their relationships in a given domain. This will enable software agents to automatically understand, process, and reason about available information and services.

In this emerging environment, searching for a single piece of information in a particular Web page is no longer sufficient. Instead, the users typically have in mind some high-level, complex task that they need to fulfil, and for this purpose they likely need to use one or more services that provide a specific functionality or to find several pieces of information, from various sources independent from each other, and then combine them appropriately to produce a final result. To that end, two primary challenges can be identified:

<i>discover and select potentially useful data sources and services</i>

and

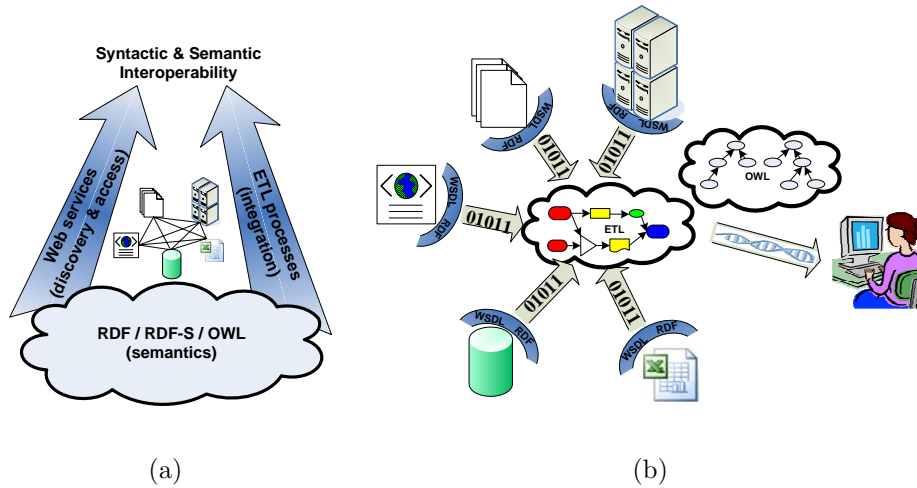


Figure 1.1: (a) The main axes of our approach; (b) Discovery and integration with ontologies, Web services and ETL processes

combine information from multiple, distributed, and heterogeneous sources.

This thesis deals with the above issues exploiting ontologies, Web services, and Extract-Transform-Load (ETL) processes (see Figure 1.1(a)). The use of Web services facilitates the interoperability between heterogeneous platforms, providing the means to discover and access, in a uniform way, available data sources, as well as operations that can be executed on these data. The use of ETL processes allows to appropriately reconcile and combine information from these sources, so as to satisfy the user requirements and to increase the quality and value of the original data. Ontologies provide the semantics that support and drive the whole process. Our work assumes an environment where data sources are made available to the network through Web service interfaces, and then ETL processes are employed to homogenize, combine, and aggregate incoming data to provide the user with the requested information, as depicted in Figure 1.1(b). In addition, to address the need for distributed architectures, we study also the search for data and services in peer-to-peer networks. Summarizing, in this thesis we have identified and addressed the following three main problems:

1. *how services can be discovered and selected;*
2. *how services and data can be searched in distributed environments;*
3. *how to facilitate the design of ETL processes.*

Next, we discuss in more detail these topics, and we present the specific challenges that arise and the contributions made by this thesis to address them.

1.1 Challenges and Contributions

Towards the overall goal, which is the discovery and integration of data and services in the Semantic Web, this thesis identifies and focuses on three major issues: a)

the matchmaking and ranking of Web services, b) the search for services and data in peer-to-peer networks, and c) the conceptual design of Extract-Transform-Load processes. In the following, we describe the challenges that arise in each one of these topics, and we present the contributions of the thesis.

1.1.1 Service Discovery and Selection

Web services are software components that are accessible over the Web and are designed to perform a specific task, which essentially comprises either returning some information to the user (e.g., a weather forecast service or a news service) or performing an action that alters the world state (e.g., an online shopping service or an online booking service). During the last years, Web services have become increasingly popular as a key technology for realizing Service Oriented Architectures, allowing for interoperability among heterogeneous systems and integration of inter-organization applications. They are described by a well-defined interface, which provides the number, the names, and the types of the service input and output parameters, and is expressed in a standardized language. By standardizing the interfaces and the exchanged messages between communicating systems, they can significantly reduce the development and, even more importantly, the maintenance cost for large-scale, distributed, heterogeneous applications.

In a broader sense, any dynamic Web site can be thought of as a (collection of) Web service(s). Thus, Web services provide also a means for querying the hidden Web. In addition, they are often used as wrappers for databases, in order, for example, to allow data access through firewalls, to hide the details regarding how the underlying data is stored or to enforce data access policies. Another use is met in mashup applications; e.g., DAMIA [145], Yahoo Pipes [173], and so on. These constitute a recently emerging trend, where users select and combine building blocks (essentially, services) to create applications that integrate information from several Web sources; e.g., [173].

Consequently, it becomes apparent that the Web services paradigm constitutes an integral part of many modern real-world applications, and that Web services rapidly gain popularity and are becoming a critical resource on the Web. This fact stresses the need for enhancing existing search engines with effective techniques for retrieving and ranking Web services. However, current standards for describing and locating services, i.e., WSDL and UDDI, focus on describing the syntactic aspects of a service and on performing keyword-based matching. Even though interoperability at the syntactic level is a necessary requirement, the identification and selection of appropriate services should be done in terms of the semantics of the requested and offered capabilities. To this direction, the Semantic Web, through the use of ontologies, provides the means to enrich the service descriptions with semantic information, allowing software agents to reason about the terms in these descriptions. This is a significant step for increasing the precision of the discovery process, as well as for minimizing the required human intervention. In particular, three main approaches have been recently proposed to semantically enhance Web service descriptions: SAWSDL [75], OWL-S [31], and WSMO [60]. The main idea underlying all these approaches is to use appropriate ontologies to semantically annotate several aspects of a Web service, such as inputs, outputs, preconditions, and effects, as well as non-functional parameters (e.g., QoS aspects).

Consider the following typical Web service discovery scenario. The user provides a complete definition of the *desired* service and poses a query on a system maintaining a repository of *advertised* service descriptions. Alternatively, the user could identify a service, e.g., from some previous query, and request similar results. Then, the search engine employs a *matchmaking* algorithm in order to identify advertisements that are relevant to the user's request. According to the traditional principle for Web service discovery, the matchmaking process involves two generic phases. First, a criterion for assessing the similarity of service parameters is selected and applied. Then, individual parameter scores are aggregated to obtain the overall degree of match between the request and the advertisement.

In the first phase, the degree of match between parameters of the request and the advertisement is estimated. Matching can take into account a variety of criteria, ranging from string matching between parameter names and/or associated comments to matching using auxiliary information from a domain thesaurus [129]. There are two major paradigms for assessing this degree of match. The first treats the parameter descriptions as documents and employs basic Information Retrieval techniques to extract keywords; e.g., [46]. Next, a string similarity measure is used to compute the degree of match. The second paradigm follows the Semantic Web vision. Services are enriched by annotating their parameters with semantic concepts taken from domain ontologies; e.g., [118, 93]. Then, estimating the degree of parameter match reduces to a problem of logic inference: a reasoner is employed to check for equivalence or subsumption relationships between the concepts of the parameters under investigation. Both paradigms share their weaknesses and strengths. Regarding the former, since service descriptions are essentially very short documents with few terms, keyword-based matchmaking fails to properly identify and extract semantics. The latter is also plagued with several limitations, most noteworthy the lack of available ontologies, the difficulty in achieving consensus among a large number of involved parties, and the considerable overhead in developing, maintaining an ontology and semantically annotating the available data and services. More recently, hybrid techniques for estimating the degree of parameter match have appeared, e.g., [84], taking into account both paradigms. Still, the common issue with all approaches is that there is no *single* matching criterion that is optimal for determining the similarity between parameters. Instead, different similarity measures may be more suitable, depending on the particular domain or the particular pair of request and offer.

The second phase of matchmaking regards the computation of the *overall* degree of match for a pair of requested and advertised services, taking into consideration the individual scores of corresponding parameters. Since there is no consensus on how to aggregate scores, there are various approaches to this phase, as well. One direction is to assign weights, calculated from user feedback, to the degrees of parameter match [46]. However, choosing the appropriate weights for each contributing factor requires either a-priori knowledge of the users' preferences and of which similarity measure is more appropriate for each case, or turning to machine learning techniques. Both alternatives face serious drawbacks, and raise a series of other issues to be solved. More often, methods are pessimistic adopting a worst-case scenario, where the overall service similarity is derived from the worst degree of match among parameters. This, however, leads to loss of information that may significantly affect the quality of the retrieved results. For example, services, having a single bad

matching parameter may be excluded from the result set, even though they are potentially good alternatives.

Typically, very few services constitute perfect matches, while a large number of services only *partially* match the request. It is crucial in such scenarios that the retrieved results are presented as a ranked list sorted by decreasing similarity to the user's request, so that better candidates can be identified quickly.

Summarizing, we can identify the following main challenges for the matchmaking and ranking of (Semantic) Web services: (a) how to assess the similarity between service requests and descriptions; (b) how to aggregate the similarities between individual parameters in order to obtain the overall degree of match between a service request and a service advertisement; and (c) how to obtain a ranking of the offered services in the presence of multiple matching criteria. In this thesis we address these issues, making the contributions outlined below:

- We propose a matching function for Semantic Web services, based on a semantic similarity measure between the parameters of service requests and service advertisements. The matching is inspired by the well-known evaluation measures used in the area of Information Retrieval: recall and precision. Both measures are adapted appropriately to express the extent of match between a service request and a service advertisement. This allows for the matching function to be asymmetric, so as to distinguish whether the service capabilities are a superset or a subset of the request. The proposed matching function considers the semantic information encoded in the associated domain ontology, including both the class hierarchy and the properties of classes, as well as restrictions possibly defined on these properties. Thus, it is applicable to both taxonomies and more expressive ontologies, such as OWL ontologies. Moreover, the matching takes into consideration both input and output parameters, as well as service preconditions and effects. In addition, it is flexible and customizable, and may be adapted to support specific application needs or user requirements and preferences.
- We use the concept of skyline queries to determine the overall degree of match between service requests and and service advertisement, without requiring a-priori knowledge of user preferences. For this purpose, we formalize the problem of discovering and selecting Semantic Web services as a skyline computation problem. Based on a state of the art skyline computation algorithm, we provide an effective and efficient way to handle the service selection process, dealing both with the requester's and the provider's perspectives. We evaluate experimentally the performance of the proposed approach using real and synthetic data.
- We propose a method for ranking service descriptions under multiple matching criteria. In particular, we introduce the notion of top- k dominant Web services, specifying three criteria for ranking Web service descriptions with respect to service requests, using multiple similarity measures. We present efficient algorithms for selecting the top- k matches for a service request, based on these criteria. Also, we experimentally evaluate our approach, both in terms of retrieval effectiveness, using real requests and relevance sets, as well as in terms of efficiency, using synthetically generated scenarios.

1.1.2 Search for Services and Data in P2P Networks

In today's world, information is a valuable asset. However, a piece of information is of little or no use if not seen within a specific context. Thus, modern applications do not (or will not) often operate in isolation. This leads to a world of interconnected devices, services, and applications. In such a highly dynamic environment, centralized architectures fail to provide efficiency and scalability; instead, architectures are required that better reflect the inherently distributed nature of such applications. Thus, peer-to-peer (P2P) networks have emerged as an alternative paradigm.

Peer-to-peer systems are typically used for massive, large-scale sharing and exchange of data. In these systems, content and, more generally, resources, are exchanged directly, rather than requiring the intermediation of a centralized server or authority [9]. This makes them more flexible and robust, as they can avoid bottlenecks and adapt to failures. Essentially, a P2P network is a network comprising peer computers (nodes) and connections (edges) between them, and it is referred to as "overlay", since it is formed on top of (and independently from) the underlying physical network. Overlay networks can be categorized according to their centralization and structure. In the so-called "pure" P2P architectures, all the nodes are completely equivalent in terms of the functionality and the tasks they perform. On the other hand, in "hybrid" P2P architectures a small subset of the peers is selected (probably automatically and dynamically) to operate as super-peers. These super-peers typically maintain indexes of the content of other peers, so as to facilitate search on the network. Moreover, the overlay network may be created either non-deterministically (i.e., ad hoc) as new nodes are added, or according to specific rules. In the first case, the P2P network is characterized as "unstructured". In this type of systems, there is no correlation between the content of a peer and its position in the network topology. Although this significantly reduces any administrative efforts, difficulties arise in locating content efficiently (in fact, in the absence of appropriate routing indexes, queries are propagated to all peers, thus causing the flooding of the network). On the other hand, in "structured" P2P networks, there is a specific mechanism that specifies how peers are placed in the overlay and how content is mapped to peers. This creates a distributed routing table so queries can be answered efficiently.

Recently there has been a growing interest in research issues overlapping the field of P2P computing and the Semantic Web, raising a series of new opportunities and challenges [157]. As the number of services on the Web increases, the efficiency and the scalability of service discovery techniques becomes a critical issue. In addition, several applications are inherently distributed. Consider, for example, a network of businesses or institutions, each providing its own services; creating and maintaining a centralized registry would not be desirable. Still, the majority of current service discovery approaches focuses on centralized architectures, i.e., a single service registry or multiple service registries synchronizing periodically. Existing approaches for service discovery in P2P environments typically rely on the use of ontologies to partition the network topology into concept clusters, and then forward requests to the appropriate cluster. However, constructing concept clusters in a fully automated way is not straightforward, as well as providing guarantees regarding search times and load balancing. Consequently, there is a need to explore alternative methods for service discovery in P2P settings.

On the other hand, Peer Data Management Systems (PDMS's) -e.g., [11, 66]- constitute emerging applications of the P2P paradigm, holding a leading role in sharing semantically rich information. PDMS's consist of autonomous sources that store and manage structured data locally, revealing part of their local data schema to the rest of the peers. Pure P2P systems -i.e., without super-peers- are considered to operate in lack of a global schema. Without a reference schema, peer databases express and answer queries based on their local schema. In particular, peers that are directly linked, i.e., *acquainted*, establish a common way of exchanging and comprehending each others' data. Usually this is realized in the form of mappings between the peer schemas. Using the peer mappings and some suitable rewriting algorithm, two acquainted peers can propagate queries to each other. The nature of structured data stored in the overlay enforces strict methods of querying and query rewriting. However, frequently, the user intends to obtain information that is semantically relevant to the posed query, rather than information that strictly complies to structural constraints. The available rewriting algorithms for structured data target the classic data integration problem [63] and consider only queries that can be completely rewritten to the target schema under a set of mappings. Still, such approach is not enough for a P2P environment where peers seek and are satisfied with information semantically similar, but not necessary identical, to their requests (as in the case of popular P2P file sharing applications). An example application where the semantic similarity plays a significant role, is the creation of *social networks*. Recently, new social networking services have been emerging, that are similar to human social networks. Services such as MySpace [110] and Orkut [114], to mention a few, form virtual communities, with each participant setting his/her own characteristics and interests. Their goal is to allow members to form relationships through communication with other members and sharing of common interests. In these applications, the search for identical information among the users is not realistic.

Hence, there is a necessity for investigating the notion of semantic similarity of peer schemas, and, furthermore, between peer queries and their rewritten versions. Using such similarity criteria, users can identify peers sharing similar interests to theirs. For each specific query they pose, the system can decide which peers can rewrite it better and, thus, give more satisfying answers. Peer schemas and query rewritings can be ranked according to their semantic relativeness to a reference schema or to an original query, respectively. Nevertheless, it is not straightforward to encounter the semantic similarity problem in the context of structured data without any additional semantic information [78]. Database schemas and respective mappings cannot capture sufficient semantic metadata so that a qualitative solution for the semantic similarity problem can be anticipated. Instead, the use of ontologies can help to deal with this deficiency.

Summarizing, this part of the thesis addresses two main challenges: (a) the search for services in structured P2P networks; and (b) the search for structured data in unstructured P2P networks. Below we outline our main contributions to these problems:

- We propose an efficient method for service discovery in P2P environments. For this purpose, we employ a novel encoding of the service descriptions, and we index these representations to prune the search space, so as to minimize the number of comparisons required to locate the matching services. We then

present an algorithm that, given a desired ranking function, fetches the top- k matches progressively, thereby further reducing the search engine's response time. We extend this method to a suitable, structured P2P overlay network, showing that the search process can be done efficiently in a decentralized, dynamic environment. We demonstrate the efficiency and the scalability of our approach through experimental evaluation.

- We describe a Peer Data Management System (PDMS) enriched with a domain ontology that is used to semantically compare peer schemas, as well as queries propagated in the network. More specifically, we propose the use of the measures *recall* and *precision* for quantifying the notion of semantic similarity between peer schemas and (rewritten) queries. Based on that, we build a combined similarity measure that takes into consideration the semantics of the peers' schemas, the mappings between the peers, and the queries issued by the peers. Moreover, we extend the proposed similarity measure to allow for the comparison of elements across different ontologies.

1.1.3 Design of ETL processes

Successful planning and decision making in large enterprises requires the ability of efficiently processing and analyzing the organization's informational assets, such as data regarding products, sales, customers, and so on. Such data are typically distributed in several heterogeneous sources, ranging from legacy systems and spreadsheets to relational databases, XML documents and Web pages, and are stored under different structures and formats. Thus, data warehouses are employed to integrate the operational data and to provide an appropriate infrastructure that allows querying, reporting, mining and other advanced analysis techniques to be carried out easier and faster. A data warehouse comprises a front stage and a back stage. The former is targeted to end users, and it allows them to access the data and use them in decision support tools. The latter is targeted to administrators, which are responsible for designing and implementing the processes that populate the data warehouse with data from the operational sources. These specialized processes are referred to as Extract-Transform-Load (ETL) processes. ETL processes are responsible for the extraction of data from distributed and heterogeneous operational data sources, their appropriate cleansing and transformation, and finally, their loading into the target warehouse, so that they can be queried and processed in a uniform way. In more detail, this typically involves tasks such as: identification and extraction of relevant information from the data sources; cleaning of the data, which may be incomplete or inconsistent; transformation between different representations (e.g., different naming schemes and/or structures); transformation between different value formats (e.g., different units of measurement); aggregation of data values.

In general, the design of ETL processes, especially when multiple and heterogeneous sources are involved, constitutes a very costly –both in time and resources– and complex part of the data warehouse design. The problem becomes even more prominent with the explosion of the information available in Web repositories, further accelerated by the Web 2.0 trends and technologies, and combined with the ever increasing information needs of the users. All these necessitate that modern applications often draw from multiple, heterogeneous data sources to provide added value services to the end users. Such environments raise new challenges for the

problem of data integration, since naming conventions or custom-defined metadata, which may be sufficient for integration within a single organization, are of little use when integrating inter-organization information sources or Web data sources.

The key challenge underlying all such situations is how to reconcile, both structurally and semantically, the data between the source and target specifications. *Structural heterogeneity* refers to the fact that data in different sources may be structured under different schemas. For instance, information that is stored in one attribute/relation in one schema may be stored in more than one attributes/relations in another schema. On the other hand, *semantic heterogeneity* considers the intended meaning of the schema elements. In order to achieve semantic interoperability in heterogeneous information systems, the meaning of the information that is interchanged has to be understood across the systems.

However, previous work towards the conceptual design of ETL processes has treated this task mainly as a manual activity [166, 98, 162]. The same holds for the plethora of the commercial solutions currently existing in the market, such as IBM's Data Warehouse Manager [71], Informatica's PowerCenter [72], Microsoft's Data Transformation Services [104], and Oracle's Warehouse Builder [113]. All these approaches, at the conceptual level, focus on the graphical design and representation of the ETL process, whereas the identification of the required mappings and transformations needs to be done manually.

The lack of precise metadata hinders the automation of this task. The required information regarding the semantics of the data sources, as well as the constraints and requirements of the data warehouse application, tends to be missing. Usually, such information is incomplete, or even inconsistent, often being hard-coded within the schemata of the sources or provided in natural language format (e.g., after oral communication with the involved parties, including both business managers and administrators/designers of the enterprise data warehouse) [69]. Consequently, the first stage of designing an ETL process involves gathering the available knowledge and requirements regarding the involved datastores. Given that ETL processes are often quite complex, and that significant operational problems can occur with improperly designed ETL systems, following a formal approach at this stage can allow a high degree of automation of the ETL design. Such an automation can reduce the effort required for the specification of the ETL process, as well as the errors introduced by the manual process. Thus, in the context of a data warehouse application, and in particular of the ETL process design phase, an ontology can play a key role in establishing a common conceptual agreement and in guiding the extraction and transformation of the data from the sources to the target.

Motivated by this observation, we envision a methodology for the task of ETL design that comprises two main phases. First, we consider an ontology that captures the knowledge and the requirements regarding the domain at hand, and it is used to semantically annotate the datastores. The ontology may already exist, since in many real world applications the domain of the ETL environment is the same; e.g., enterprise or medical data. In such case, the ontology can be re-used or adapted appropriately. If such ontology does not exist, then during the first phase of the design, a new ontology should be created. Clearly, the details of this phase largely depend on the particular needs and characteristics of each project. For example, there may exist different ways and sources to gather requirements, different methodologies to create an ontology, annotations may be specified manually or semi-automatically,

and so on. In this work, we focus on the second phase of the design: having the ontology available, we investigate how the ontology and the annotations can be used to drive, in a semi-automatic manner, the specification of the ETL process.

Notice that the burden of using an ontology is reduced mainly to annotating the source and target schemata with it. Several approaches toward the facilitation of the automatic schema matching have already been proposed [129, 140]. Nevertheless, we argue that even if the designer has to do the whole task manually, still, it will be easier to map individual attributes (one each time) to a domain ontology rather than try to fill in the puzzle having all the pieces around at the same time. Additionally, the existence of an ontology that carries the mapping of the source and target tables can be used in other applications as well. We mention two prominent examples: (a) the use of such an ontology to produce textual reports (see Section 4.2.5); and (b) such an ontology can be used as a convenient means to warehousing Web data, as an individual may easily plug-in his/her data source into the ontology, and then the rest of the ETL process could be derived automatically, using our approach.

Summarizing, the final part of the work presented in this thesis addresses the issue of using ontologies to facilitate the conceptual design of ETL processes. More specifically, we consider the semantic annotation of available data sources and the derivation of appropriate transformations to structurally and semantically reconcile data between them. Regarding semantic heterogeneity, two main causes are considered: *naming conflicts*, which occur when naming schemes of information differ significantly (a frequent phenomenon is the presence of homonyms and synonyms); and *scaling conflicts*, which occur when different reference systems are used to measure a value; e.g., different currencies or different date formats. We propose two alternative approaches for addressing this problem, as outlined below:

- In the first approach we rely on the use of an OWL-DL reasoner to identify required ETL operations. At the beginning, a graph-based representation, called *datastore graph*, is employed as a common conceptual model for the datastore schemas, so that both structured and semi-structured sources can be handled in a unified way. Then, an application ontology is constructed, which is also represented by means of a graph, termed *ontology graph*, and which is used to semantically annotate the schemas, so that mappings between them can be later inferred. Based on the provided annotations, the reasoner infers semantic correspondences and conflicts among the involved datastores and proposes a set of conceptual operations for transforming data from the sources to the target. In addition, we show how this process can also support the generation of reports in a format resembling natural language, which can be used to facilitate the validation and the maintenance of the design by all the involved parties.
- In the second approach we treat the conceptual design of an ETL process as a graph transformation process. Exploiting the graph-based nature of the datastore schemata and of the ETL processes, we provide an appropriate formulation of the problem drawing from the well-established graph transformation theory. This allows us to develop a customizable and extensible set of graph transformation rules, which determine the choice and the order of operations comprising the ETL scenario, in conjunction with the semantic information conveyed by the associated ontology.

1.2 Thesis Outline

The rest of the thesis is structured as follows. Chapter 2 presents our techniques for the discovery and selection of Semantic Web services. Chapter 3 deals with the use of ontologies for searching for services and data in P2P environments. Then, our ontology-based approach towards the conceptual design of ETL processes is presented in Chapter 4. In more detail:

- *Chapter 2.* This chapter presents our approach for Web service matchmaking and ranking. We start with some preliminaries regarding Semantic Web service descriptions, and a discussion of related work. Then, we present our matching function for Semantic Web services, employing a semantic similarity measure to evaluate the degree of match between requested and offered service parameters. Next, we deal with the use of skyline queries for Semantic Web services, which allows us to compute the best matches between a service request and a service advertisement in a way that accommodates diverse user preferences. Finally, we propose methods for ranking offered Web services under multiple matching criteria.
- *Chapter 3.* This chapter deals with service discovery and data sharing in P2P overlay networks. First, we present our technique for Semantic Web service discovery in a structured P2P network, based on an efficient encoding and indexing of service descriptions. Then, we describe an approach for the use of ontologies on top of Peer Data Management Systems, to allow peers to search for information that is semantically similar, but not necessarily identical, to their requests.
- *Chapter 4.* This chapter concentrates on the use of ontologies for the conceptual design of ETL processes. Using a domain ontology to semantically annotate the sources and the target, we show how correspondences and conflicts between them can be automatically inferred in order to construct the conceptual design of the ETL scenario. Then we provide an alternative approach that proceeds with the design of ETL processes through graph transformations, based on a defined set of graph transformation rules.
- *Chapter 5.* This chapter presents our conclusions, as well as possible directions for future work.

Chapter 2

Web Service Matchmaking and Ranking

Users often need to find a service on the Web that performs a specific functionality, e.g., to purchase a product online or to reserve a ticket. In other cases, they want to find a service that provides access to some underlying data repository, where the service is used by the owner to provide platform independence or to apply access control policies. Thus, mechanisms are required to support the discovery of services that match the user needs. Current industry standards allow for the discovery of services based on syntactic match of the interfaces, while recent efforts have concentrated on semantically enriching service descriptions to increase the precision of the discovery process. To that end, the matchmaking between service descriptions is based on the use of a reasoner to infer the semantic relationship (e.g., equivalence or subsumption) between the concepts annotating the service parameters. Still, several issues remain open, especially regarding the ranking of the match results, the aggregation of the degrees of match of individual parameters to obtain the overall degree of match, and the use of multiple criteria to assess the similarity between service parameters.

This chapter deals with these problems in Web service matchmaking and ranking. First, Section 2.1 provides some preliminary information about the description of Web services, focusing on the role of ontologies to semantically annotate these descriptions, and Section 2.2 discusses related work. Then, in Section 2.3, we present our matching function, which uses the notions of recall and precision to measure the semantic similarity between the requested and offered service parameters. An example is provided first, and then the similarity measure is introduced, followed by a discussion of its properties. In Section 2.4 we use the notion of skyline queries to compute the best matches between a service request and a service advertisement. An appropriate formulation of the problem is provided, and a suitable skyline algorithm is adapted for the selection of the best matches. Issues regarding the requester's and the provider's perspectives are also discussed, and an experimental evaluation of the approach is given. Then, Section 2.5 proposes methods for ranking available Web services under multiple matching criteria. Three ranking approaches are discussed, together with corresponding algorithms for computing the best matches in each case, and an extensive experimental evaluation is provided. Finally, Section 2.6 concludes the chapter.

Our results in this chapter have been published in [153, 149, 148].

2.1 Preliminaries

Naturally, every approach for Web service discovery depends on the kind of information available in the corresponding service descriptions, e.g., whether the provided service descriptions are semantically enhanced or not. Therefore, in this section we provide some preliminary information about existing languages used to describe the functionality and capabilities of Web services. Moreover, since our focus is mainly on Semantic Web services, where the basic idea is that the service parameters are annotated using concepts from an associated ontology, we start with a brief overview of ontology languages used in the Web.

2.1.1 Ontologies on the Web

Currently, the existing Web content is primarily intended for human browsing. However, the Semantic Web [23] is emerging as a vision to enhance the current Web with machine-processable metadata, specifying the intended meaning of the provided information in a formal and explicit manner. Thus, software agents will be able to leverage these metadata in order to “understand”, process, and reason about the described resources. This will consequently increase the degree of automation, the efficiency, and the effectiveness of searching, sharing, and combining information. Ontologies constitute the cornerstone in this effort. An ontology is typically defined as a “formal and explicit specification of a shared conceptualization” [27]. They formally define the concepts of interest in a knowledge domain, their properties, and the relationships among them.

The Web Ontology Language has been proposed by W3C as a recommendation for a language for specifying ontologies on the Web [103]. OWL (in particular OWL-DL, see below) is based on Description Logics [15], a decidable fragment of first-order logic, constituting an important and commonly used knowledge representation formalism. It is also built on top of the Resource Description Framework (RDF) [100] and RDF Schema [29], which are also specifications of the W3C. Figure 2.1 depicts graphically the Semantic Web technology stack (created by Tim-Berners Lee). Below we focus briefly on RDF, RDF Schema and OWL, outlining their basic concepts. A detailed and comprehensive introduction to these topics can be found in [10].

RDF is a simple, domain independent, data model, expressed in XML syntax. It describes Web resources, and relationships between them, using statements which are triples of the form subject-predicate-object. The subject denotes the resource being described; the predicate denotes an attribute of this resource or its relationship to another resource; the object can be either a resource or a literal (e.g., a string). Each resource is uniquely identified by a URI (Universal Resource Identifier). Properties describe relations between resources, and they are also identified by URIs. Consider, for example, the following fact

Dimitris Skoutas is a PhD student at the Knowledge and Database Systems Lab and his email address is dskoutas@dblab.ece.ntua.gr.

Assuming that the resources Dimitris Skoutas and KDBSL are identified by the URIs <http://dblab.ece.ntua.gr/~dskoutas> and <http://dblab.ece.ntua.gr>, respectively, this fact can be described using three RDF statements, as shown below in XML syntax:

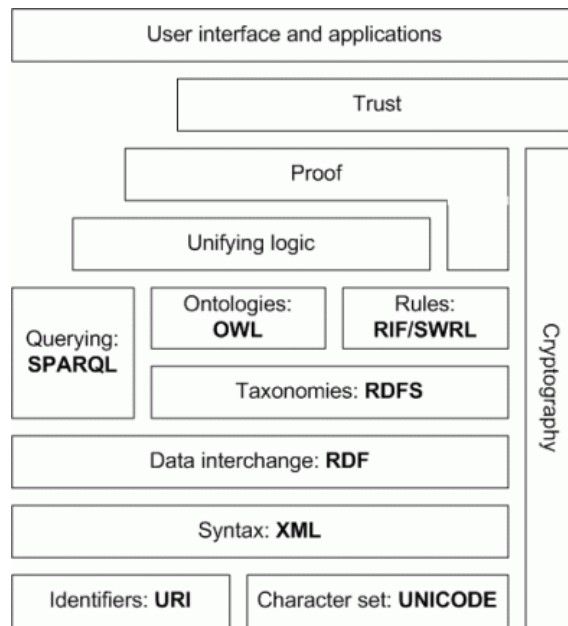


Figure 2.1: *The Semantic Web technology stack*

```
<rdf:Description rdf:about="http://dblab.ece.ntua.gr/~dskoutas">
  <occupation rdf:resource="PhDStudent"/>
  <worksAt rdf:resource="http://dblab.ece.ntua.gr"/>
  <email>dskoutas@dblab.ece.ntua.gr</email>
</rdf:Description>
```

RDF Schema is a vocabulary description language for describing classes and properties of RDF resources. Essentially, it allows to specify the terminology used in RDF statements. For example, it provide the means to specify classes of objects or to specify the domain and range of properties. Also, it provides semantics for generalization hierarchies of such properties and classes. A class being a subclass of another means that all the individuals of the first also belong to the second. Notice that multiple inheritance is allowed, i.e., a class may have more than one superclasses. For example, RDF Schema allows us to express the fact that

A PhD student is a postgraduate student

by means of the statement

```
<rdfs:Class rdf:about="PhDStudent">
  <rdfs:subClassOf rdf:resource="PostgraduateStudent"/>
</rdfs:Class>
```

Finally, OWL is a more expressive vocabulary description language for classes and properties. In particular, it comprises three sublanguages that strike a trade-off between expressiveness and complexity. OWL-Full contains all the constructors of the language and allows their combination in arbitrary ways; this allows for maximum expressiveness, making however the language undecidable. OWL-DL (based on Description Logics [15]) dictates some restrictions on how the constructors of the language can be used, thus permitting efficient reasoning support. Finally, OWL-Lite allows only a subset of the language constructors, leading to a language that is

simpler, more comprehensive, and easier to implement, but of course with restricted expressiveness. The constructors available in OWL allow, for example, to state that two classes are equivalent or disjoint, that a property is inverse of or symmetric with another, to add restrictions on properties or to form boolean combinations of classes (i.e., union and intersections).

In our work, the use of OWL focuses on defining

- a set of *classes*, representing the entities of interest in the domain of discourse, and
- a set of *properties*, representing attributes of these entities or relationships between them. Notice that two types of properties are provided:
 - *object properties*, which relate instances of one class to instances of another class, and
 - *datatype properties*, which relate instances of one class to values of a specified datatype.

Classes and properties are then organized in an appropriate hierarchy. Furthermore, restrictions on the values and the minimum or maximum cardinality of a property with respect to a specific class are defined. Notice also that the use of custom data types can be provided by proposed extensions of OWL, such as OWL 1.1 [123] or OWL-Eu [116].

For example, using OWL one could express the fact that

A lab should have at least 1 member

by means of the statements

```
<owl:Class rdf:about="Lab">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="hasMember"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
    </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

2.1.2 Web Service Description

The Web Services Description Language (WSDL) [38] is the current industry standard used to describe Web services. A WSDL document is written in XML and describes a Web service by means of the following main elements:

<**types**> describes the kinds of messages that the service receives and sends

<**interface**> describes what abstract functionality the service provides

<**binding**> describes how the service can be accessed

<**service**> describes where the service can be accessed

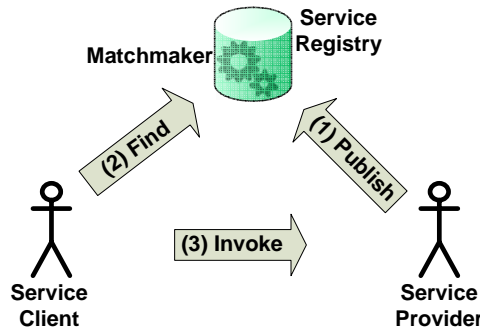


Figure 2.2: Overview of Web service discovery

As a simple example, consider a currency converter Web service, called “converter”, that converts euros to U.S. dollars. A (somewhat simplified) WSDL document describing the service would look as shown below:

```

<description>
  <types>
    <xs:schema>
      <xs:element name="convert" type="xs:double"/>
      <xs:element name="convertResponse" type="xs:double"/>
    </xs:schema>
  </types>

  <interface name = "convertInterface">
    <operation name="opConvert">
      <input messageLabel="In" element="convert"/>
      <output messageLabel="Out" element="convertResponse"/>
    </operation>
  </interface>

  <binding name="converterSOAPBinding"
    interface="convertInterface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/
      bindings/HTTP/">
    <operation ref="opConvert"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/
        soap-response"/>
  </binding>

  <service name="converterService" interface="convertInterface">
    <endpoint name="converterEndpoint"
      binding="converterSOAPBinding"
      address = "http://dskoutas.example.com/converter"/>
  </service>
</description>

```

WSDL descriptions of services can be published in a UDDI directory [22], so that they can then be searched by interested parties (see Figure 2.2). UDDI, which stands for Universal Description, Discovery and Integration, registers different types

of information regarding available services, such as a description of the service functionality, information about the organization that provides the service, and possibly various metadata of the service, such as its classification in a specific taxonomy.

Service descriptions based on WSDL and UDDI focus on the syntactic aspects of the service, and therefore limit the discovery process to essentially keyword-based search. To allow for more advanced discovery and selection mechanisms, increasing the precision of the discovery process, the descriptions of services need to be enhanced with semantics. Three main proposals have emerged recently for this purpose: SAWSDL [75], OWL-S [31], and WSMO [60]. Below, we outline the basic concepts of each approach.

SAWSDL, which was based on WSDL-S [6], defines mechanisms for adding semantic annotations to WSDL components. These annotations refer to concepts from a semantic model (i.e., an ontology). Notice, however, that SAWSDL does not specify a language for representing the semantic model itself. Each concept is identified by a URI. More specifically, to allow WSDL components to reference concepts from an ontology, SAWSDL defines three extensibility attributes to WSDL 2.0, namely `modelReference`, `liftingSchema Mapping`, and `loweringSchema Mapping`. A `modelReference` can be used within a WSDL or XML Schema element to reference the URI of a concept. This is useful for matchmaking purposes, i.e., to determine whether there is semantic agreement between compared elements of service descriptions. Consider as an example the aforementioned currency converter service and a sample ontology defining the class `Currency` and its subclasses `EUR` and `USD`:

```
<rdf:RDF xml:base="http://dblab.ece.ntua.gr/currencyOnt">
  <owl:Class rdf:ID="EUR">
    <rdfs:subClassOf rdf:resource="Currency"/>
  </owl:Class>
  <owl:Class rdf:ID="USD">
    <rdfs:subClassOf rdf:resource="Currency"/>
  </owl:Class>
</rdf:RDF>
```

Then, the previously shown WSDL description of the service can be annotated with these concepts as shown below:

```
<description>
  <types>
    <xs:schema>
      <xs:element name="convert" type="xs:double"
        sawsdl:modelReference="http://dblab.ece.ntua.gr/
          currencyOnt#EUR"/>
      <xs:element name="convertResponse" type="xs:double"
        sawsdl:modelReference="http://dblab.ece.ntua.gr/
          currencyOnt#USD"/>
    </xs:schema>
  </types>
  . . .
</description>
```


However, there can still be mismatches between the XML representation of the element and its corresponding representation in the semantic model, e.g., “full name” versus “first name” and “last name”. In this case, `liftingSchemaMapping` and `loweringSchemaMapping` specify mappings from XML to the semantic model and from the semantic model to XML, respectively. These mappings are then used during service invocation.

OWL-S specifies an upper ontology for services, based on the Web Ontology Language (OWL) [103]. A service description contains both functional parameters, namely inputs, outputs, preconditions and effects, as well as non-functional parameters, such as information about the service provider or QoS aspects. In particular, the description of a service in OWL-S comprises:

- a *service profile*, which describes what the service does;
- a *service model*, which describes how the service can be used;
- a *service grounding*, which describes how the service can be accessed.

Finally, WSMO provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services to facilitate the automation of service discovery and composition. It comprises four main elements, namely:

- *ontologies*, which provide the available terminology;
- *Web service descriptions*, which describe the functional and behavioral aspects of a Web service;
- *goals*, which represent user desires;
- *mediators*, which handle interoperability problems.

2.2 Related Work

In the following, we discuss related work in the area of Web service discovery, outlining the limitations of existing approaches. We also present works in the fields of skyline computation and data fusion, which are related to our approach for combining the individual parameter scores and for supporting multiple similarity measures.

2.2.1 Web Service Discovery

2.2.1.1 Semantic Matchmaking

Early works on Web service discovery have concentrated mostly on keyword-based search, with UDDI [22] receiving most attention and becoming the de-facto industry standard. Even though UDDI provides syntactic interoperability and a classification scheme to describe the service functionality, its search capabilities are limited due to the lack of explicit semantics. To overcome this deficiency, proposals for exploiting ontologies to semantically enhance service descriptions have emerged (WSDL-S [6],

OWL-S [31], WSMO [60]). As a result, several works have been proposed describing how to efficiently integrate Semantic Web service descriptions into the UDDI registry [7, 80, 117, 118, 155].

In the presence of semantically rich service descriptions, the problem of Web service matchmaking is treated as a logic inference task. In particular, in [118] four degrees of match are identified, based on the existence of subsumption relationship between concepts contained in the request and the advertisement. More specifically, the match is called *exact*, if the request is equivalent to or direct subclass of the advertisement; *plug-in*, if the request is subsumed by the advertisement; *subsume*, if the request subsumes the advertisement; and *fail*, if there is no subsumption relation. In [93] the last case is further distinguished in *intersection*, if the intersection of the request and the advertisement is satisfiable, and *disjoint*, if the two concepts are disjoint. Also, the match is considered *exact*, only when the two concepts are equivalent. In the same direction, the work presented in [21] addresses the matching of requested and offered parameters as matching of bipartite graphs. The main difference with our work is that these approaches only rank the matches in a discrete scale, according to the aforementioned types of match, without specifying a way for ranking services within the same type of match.

In [35] a semantic matching algorithm is presented, based on Tversky’s feature-based similarity model [163]. The algorithm assesses the similarity between requested and offered inputs or outputs by means of the proportion of shared properties between the corresponding concepts, instead of using the concept hierarchy. In contrast to the previous approaches, the result of the matching algorithm is a continuous value in the range [0..1]. However, even though the proposed matching function is asymmetric, asymmetry is achieved by assigning a score equal to 1 to one of the two alternative cases, which does not allow to differentiate services within this case, and is also the same score used for an exact match. Instead, our approach addresses this deficiency by employing two separate measures, namely recall and precision (see Section 2.3). Another difference is that the algorithm does not consider additional details about the properties of the concepts, such as the existence of property hierarchy or restrictions. Moreover, this approach, as well as the previous ones, do not address the matching of preconditions and effects.

In [68] the authors propose a method for matching OWL-S annotated services, by defining a similarity measure for OWL objects, based on the ratio of common RDF triples in their descriptions. To measure the information content of a triple, they use the notion of “inferencibility” of a tripple t , which is defined as the number of new RDF triples that can be generated by applying a set of inference rules to t . Therefore, this approach depends on the considered OWL constructs and set of inference rules. Furthermore, the proposed similarity function is symmetric.

Finally, several works exist addressing the issue of semantic similarity between concepts in a taxonomy. In [133] the notion of *information content* is used to define the similarity between two concepts. The information content of a concept is defined as the negative logarithm of the probability of encountering an instance of that concept. The similarity between two concepts is then assessed by the information content of their most specific common ancestor. In [97] a domain-independent, information-theoretic definition of similarity is provided, and its applicability in different domains is demonstrated. The similarity between two concepts is measured by the ratio of the amount of information needed to state their commonality and the

information needed to fully describe them. These approaches are not directly applicable in our case for three reasons: (a) they rely on the existence of a probabilistic model for the domain; (b) they consider only the concept hierarchy; and (c) they are symmetric. Another semantic similarity function is presented in [136], which allows similarities to be asymmetric. However, asymmetry is “hard-coded” in the similarity function. Additionally, the similarity function is based on the number of descendants of the compared concepts, and therefore the similarity decreases with the hierarchy depth.

2.2.1.2 Hybrid Matchmaking

The works presented previously compare service parameters using a single matching criterion. In [43], the need for employing many types of matching is identified, and the integration of multiple external matching services to a UDDI registry is proposed. Then, selecting the external matching service is based on specified policies (e.g., the first available, or the most successful). If more than one matching services are invoked, the policy specifies whether the union or the intersection of the results should be returned. The work in [46] focuses on similarity search for Web service operations, combining multiple sources of evidence. A clustering algorithm groups names of parameters into semantically meaningful concepts, used to determine the similarity between I/O parameters. Different types of similarity are combined using a linear function, with weights being assigned manually, based on analysis of the results from different trials. Learning the weights from user feedback is mentioned as for future work.

OWLS-MX [84] is a hybrid matchmaker for OWL-S services, which matches I/O parameters, utilizing both logic-based reasoning and IR techniques. Similarly, WSMO-MX [79] is a matchmaker for WSMO-oriented service descriptions.

The above works focus on matching pairs of parameters from the requested and offered services, while the overall match is typically calculated as a weighted average, assuming the existence of an appropriate weighting scheme. Furthermore, none of these approaches considers more than one matching criteria simultaneously. However, from the diversity of these approaches, it is evident that there is no single matching criterion that constitutes the silver bullet for the problem. On the other hand, our approach addresses this issue and provides a generic and efficient way to accommodate and leverage multiple matching criteria and service parameters, without loss of information from aggregating the individual results and without requiring a-priori knowledge concerning the user’s preferences.

2.2.2 Skyline Computation

Our proposed methods for service discovery resemble concepts of multi-objective optimization, which has been studied in the literature, initially referred to as *maximum vector* problem [88, 128], and more recently, as *skyline computation* [28]. Given a set of points in a d -dimensional space, the skyline is defined as the subset containing those points that are not dominated by any other point. Thus, the best answers for such a query exist in the skyline.

Skyline queries have received a lot of attention over the recent years, and several algorithms have been proposed. Block Nested Loop (BNL) [28] is a straightforward, generic skyline algorithm. It iterates over the data set, comparing each point with

every other point, and reports the points that are not dominated by any other point. Sort First Skyline (SFS) [39] improves the efficiency of BNL, by pre-sorting the input according to a monotone scoring function F , reducing the number of dominance checks required. The Sort and Limit Skyline algorithm (SaLSa) [18] proposes an additional modification, so that the computation may terminate before scanning the whole data set.

Even though our work exploits the basic techniques underlying these methods, these algorithms are not directly applicable to our problem, as they do not deal with ranking issues (the objects comprising the skyline are incomparable to each other) or with the requirement for multiple matching criteria. Furthermore, the size of the skyline is not known a-priori, and, depending on the data dimensionality and distribution may often be either too large or too small. In addition, our work borrows some ideas from the probabilistic skyline model for uncertain data introduced in [125], which however also does not provide any ranking of the data.

Other works exploit appropriate indexes, such as B⁺-tree or R-tree, to speed-up the skyline computation process [158, 87, 120, 90]. These approaches are not applicable to our problem, where no indexes are available.

The importance of combining top- k queries with skyline queries has been pointed out in [174]. However, there are some important differences to our work. First, this approach also relies on the use of an index, in particular an aggregate R-tree. Second, it considers only one of our proposed ranking criteria (see Section 2.5.1). Third, it does not address the requirement for handling multiple matching criteria.

2.2.3 Data Fusion

Since our work aims to support the ranking of service descriptions based on multiple matching criteria, in the following we review related work from the area of data fusion. Given a set of ranked lists of items returned from multiple methods – e.g., from different search engines, different databases, and so on – in response to a given query, *data fusion* (also known as results merging, metasearch or rank aggregation) is the construction of a single ranked list combining the individual rankings. FA, TA and NRA are typical algorithms for finding the top- k objects, given a set of rankings and a monotone aggregation function [50]. For example, FA scans the sorted lists in parallel until at least k items have been seen in all the lists. Then, for each item seen so far, it gets its local scores (doing random access if the item has not been seen in some of the lists), and it computes its overall score. Finally, the items with the k highest overall scores are returned. Hence, these algorithms require the use of a specific aggregation function. Instead, in our case we assume that the user preferences are not known, and therefore our approach relies on the notion of dominance to determine the overall ranking, as described in Section 2.5.

In Information retrieval, data fusion techniques can be classified [14] based on whether they require knowledge of the relevance scores and whether training data is used. The simplest method based solely on the documents’ ranks is the Borda-fuse model introduced in [14]. In its non-training flavor, it assigns as score to each document the summation of its rank (position) in each list. The documents in the fused list are ranked by increasing order of their score, solving ties arbitrarily. Training data can be used to assess the performance of each source and, hence, learn its importance. In this case, the sources are assigned weights relative to their

importance and a document's score is the weighted summation of its ranks.

The Condorcet-fuse method [107] is another rank-based fusion approach. It is based on a majoritarian voting algorithm, which specifies that a document should be ranked higher in the fused list than another document if the former is ranked higher than the latter more times than the latter is ranked higher than the former. Condorcet-fuse proceeds iteratively: it identifies the winner(s), i.e., the highest ranked document(s), removes it/them from the lists and then repeats the process until there are no more documents to rank.

For the case where the relevance scores are given/known, several fusion techniques, including CombSUM, CombANZ and CombMNZ, were discussed in [52]. In CombSUM, the final (fused) relevance score of a document is given by the summation of the relevance scores assigned by each source; if a document does not appear in a list, its relevance score is considered 0 for that list. In CombANZ (CombMNZ), the final score of a document is calculated as the score of CombSUM divided (multiplied) by the number of lists in which the document appears. In [89], the author concludes that CombMNZ provides the best retrieval efficiency.

When training data is available, it is shown in [168] that a linear (weighted) combination of scores works well when the various rank engines return similar sets of relevant documents and dissimilar sets of non-relevant documents. For example, a weighted variant of CombSUM is successfully used in [141] for the fusion of multilingual ranked lists. The optimal size of the training data that balances effectiveness and efficiency is investigated in [37].

Probabilistic fusion techniques, which rank documents based on their probability of relevance to the given query, have also appeared. The relevance probability is calculated in the training phase, and depends on which rank engine returned the document among its results and the document's position in the result set. In [96], such a technique was shown to outperform CombMNZ.

An outranking approach was recently presented in [51]. According to this, a document is ranked better than another if the majority of input rankings is in concordance with this fact and at the same time only a few input rankings refute it.

Seen in the context of data fusion, our work addresses the novel problem where in each ranking a *vector of scores*, instead of a single score, is used to measure the relevance for each data item.

2.3 A Similarity Measure for Semantic Web Services

2.3.1 Example

Assume a simple scenario where the user is interested in watching a movie and is searching for a Web service that, given as input some details about the movie, returns information about available cinemas and showtimes. To semantically describe the request and the service capabilities, a sample ontology is considered, as shown in Figure 2.3. The figure displays the class hierarchy. Also, the properties of each class are shown inside brackets, together with any restrictions applied on them. Note that for the subclasses of a class, only the additional properties and/or restrictions are displayed (i.e., not those inherited by the superclass). For instance, the class

	INPUT	OUTPUT ₁	OUTPUT ₂
Req	PopMovie	Multiplex	Showtimes
Adv₁	PopMovieEuro	LuxMultiplex	Evening
Adv₂	Movie	Cinema	Showtimes
Adv₃	Movie	Multiplex	Showtimes
Adv₄	Movie	Multiplex	Evening

Table 2.1: *Service request and advertisements*

PopMovie inherits the properties *hasTitle*, *hasDirector* and *hasActor* from the class *Movie*, and also has the additional property *hasLeadActor*. The range of the latter is restricted to the class *FamousActor*. We assume, additionally, that the property *hasLeadActor* is a subproperty of the property *hasActor*.

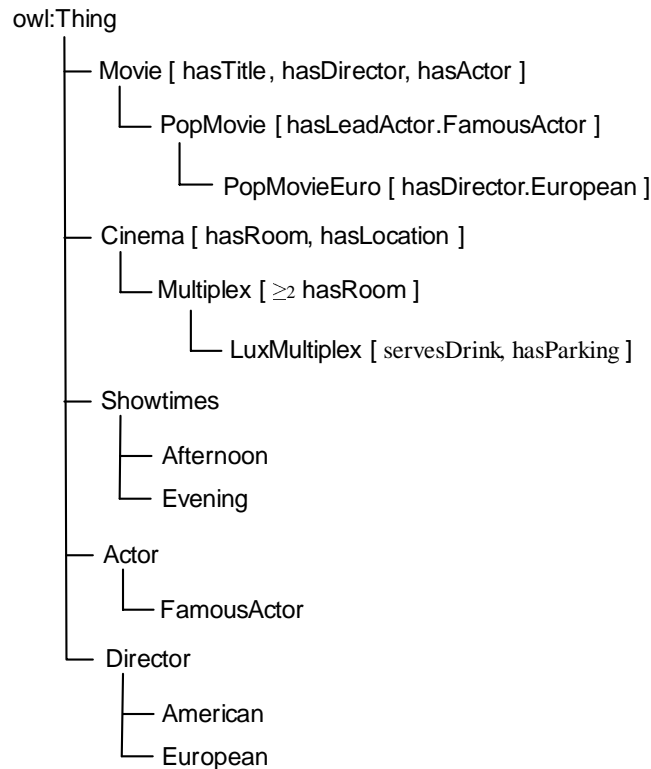


Figure 2.3: *A sample ontology snippet for the motivating example*

The service request and advertisements consist of one input parameter, corresponding to the type of movie, and two output parameters, corresponding to the type of cinema and showtimes. Each parameter is annotated by specifying its corresponding class in the sample domain ontology, as shown in Table 2.1. Based on the specified annotations, one can notice, for example, that the advertisement *Adv₁* satisfies only partially the user request, because it deals only with movies from European directors, luxury multiplex cinemas and evening showtimes. On the contrary, the advertisement *Adv₂* is more generic, as it refers to any type of movie, cinema and showtimes.

The question that arises is how to select the best candidate service for the given request. In the rest of this section we develop a similarity measure that allows to compare such a service request with available service descriptions, in terms of the

	Relevant	Not Relevant
Retrieved	TP	FP
Not Retrieved	FN	

Table 2.2: Domain of an answer to a search

semantic information conveyed by the accompanying domain ontology.

2.3.2 The Similarity Measure

To assess the degree of match between a service request and a service advertisement, we propose a similarity measure based on the concepts of *recall* and *precision* from the field of Information Retrieval. The similarity measure should take into consideration several types of semantic information available in the provided domain ontology, in a modular way.

In what follows, we consider a Semantic Web service as a tuple $SWS(S_{IN}, S_{OUT}, S_{PR}, S_{EF})$, where S_{IN} and S_{OUT} are two sets of classes in the domain ontology corresponding, respectively, to the service input and output parameters, while S_{PR} and S_{EF} are two sets of logical formulae denoting the service preconditions and effects.

2.3.2.1 Using recall and precision for matching services

Recall and precision are two widely used measures for evaluating the performance of Information Retrieval systems [16]. *Recall* expresses the proportion of relevant material actually retrieved in answer to a search request. *Precision* expresses the proportion of retrieved material that is actually relevant. More specifically, recall and precision are defined by means of the following quantities (see also Figure 2.2):

- True Positive (TP): items that are relevant and were retrieved
- False Negative (FN): items that are relevant but were not retrieved
- False Positive (FP): items that are not relevant but were retrieved

$$recall = \frac{TP}{TP + FN} \quad precision = \frac{TP}{TP + FP} \quad (2.1)$$

A single measure combining recall and precision is the weighted harmonic mean, also known as the *F-measure*. The general formula for non-negative real a is:

$$F_a = \frac{(1 + a) * precision * recall}{a * precision + recall} \quad (2.2)$$

Choosing $a > 1$, weights recall more than precision. In the literature, typical values for a are 0.5, 1, and 2.

We revisit the definitions of recall and precision measures for expressing the degree of match between a service request, denoted by the class C_R , and a service advertisement, denoted by the class C_A . C_R and C_A refer to service inputs or outputs. In Section 2.3.2.3, we extend the use of recall and precision to cover also preconditions and effects.

For the task of matching a service parameter C_A to a request parameter C_R , the relevant items are the instances of C_R , while the retrieved items are the instances of C_A . Thus, recall and precision have the following meaning:

Recall is the proportion of instances of C_R that are also instances of C_A .

Precision is the proportion of instances of C_A that are also instances of C_R .

Formally:

$$\begin{aligned} recall(C_R, C_A) &= \frac{|\{x \mid x \in (C_R \sqcap C_A)\}|}{|\{x \mid x \in C_R\}|} \\ precision(C_R, C_A) &= \frac{|\{x \mid x \in (C_R \sqcap C_A)\}|}{|\{x \mid x \in C_A\}|} \end{aligned} \tag{2.3}$$

The above definitions for recall and precision have the following properties:

- When the request is equivalent to the advertisement, i.e., $C_R \equiv C_A \equiv C_R \sqcap C_A$, then $recall = 1$ and $precision = 1$, meaning that the service capabilities exactly match the user needs.
- When the request is more specific than the advertisement, i.e., $C_R \sqsubseteq C_A$, then $C_R \sqcap C_A \equiv C_R$, thus $recall = 1$ and $precision < 1$, meaning that the service capabilities are a superset of the user needs.
- When the request is less specific than the advertisement, i.e., $C_R \sqsupseteq C_A$, then $C_R \sqcap C_A \equiv C_A$, thus $recall < 1$ and $precision = 1$, meaning that the service capabilities are a subset of the user needs.
- When the request and the advertisement overlap, i.e., $\neg(C_R \sqcap C_A \sqsubseteq \perp)$, then $recall < 1$ and $precision < 1$, meaning that some of the service capabilities match some of the user needs.
- Finally, when the request and the advertisement are disjoint, i.e., $C_R \sqcap C_A \sqsubseteq \perp$, then $recall = 0$ and $precision = 0$, meaning that the service capabilities do not match the user needs.

Observe that our approach allows the degree of match to be specified in a continuous scale, while maintaining a direct correspondence to the types of match established in related work, namely *exact*, *plug-in*, *subsume*, *intersection*, and *disjoint* [93, 118].

2.3.2.2 Calculating recall and precision

Since, in principle, the instances of a class are not known, the exact values of recall and precision can not be calculated by means of the Equations 2.3. Nevertheless, an estimation of recall and precision can be made, by comparing the two classes based on their semantic descriptions and the application ontology.

An ontology consists, at the very least, of a hierarchy of classes. Classes may be described by properties, which may also be hierarchically structured. Furthermore,

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	0.67	0.67	0.5	1	1	1
Adv₂	1	1	1	0.5	0.5	1
Adv₃	1	1	1	0.5	1	1
Adv₄	1	1	0.5	0.5	1	1

Table 2.3: Recall and precision based on the class hierarchy

it is possible to assign value or cardinality restrictions to the properties of a class. In the following, we compute the values of recall and precision taking into consideration the above types of semantic information encoded in the ontology. Notice that the derived formulae are actually applied when the type of match is either plug-in, subsume or intersection. In the case that the request and the advertisement are inferred to be equivalent (disjoint), then both recall and precision are equal to one (zero) and no further calculation is required.

Notation. $P(C)$ denotes the set of properties of a class C . $A(C)$ and $A(P)$ denote, respectively, the set of superclasses of class C and the set of superproperties of property P . $p_i(C)$ refers to the i -th property of class C .

Recall and precision based on the class hierarchy. A common approach for measuring the similarity between two classes in a taxonomy is by means of the ratio of their common ancestors [97, 133]. Given that the number of instances of a class is inversely related to the depth of this class in the hierarchy, i.e., to the number of its superclasses, recall and precision are estimated as follows:

$$\begin{aligned}
recall_I(C_R, C_A) &= \frac{|A(C_R) \cap A(C_A)|}{|A(C_A)|} \\
precision_I(C_R, C_A) &= \frac{|A(C_R) \cap A(C_A)|}{|A(C_R)|}
\end{aligned} \tag{2.4}$$

Observe that the values of recall and precision obtained from Equations 2.4 adhere to the properties discussed in subsection 2.3.2.1. For example, if $C_R \sqsubseteq C_A$, then $A(C_R) \supseteq A(C_A)$, and therefore $recall = 1$ and $precision < 1$.

As an example, consider the request *Movie_Req* and the advertisement *Movie_Adv* from the motivating example. The superclasses of *Movie_Req* are *Movie_Req*, *Advertisement* and *Movie*, while the superclass of *Movie_Adv* is *Movie*. Notice that we include in the set of superclasses the class itself, but not the root of the class hierarchy (e.g., the class *owl:Thing* in an OWL ontology). That is, top level classes are considered to be disjoint. Thus, applying Equations 2.4 we get:

$$\begin{aligned}
recall_I(Movie_Req, Movie_Adv) &= 1/1 = 1 \\
precision_I(Movie_Req, Movie_Adv) &= 1/3 = 0.33
\end{aligned}$$

Table 2.3 displays the values of recall and precision for each input and output parameter of each service advertisement considered in the motivating example, with respect to the corresponding parameters of the user request.

Recall and precision based on class properties. When classes are described by properties, this information can contribute in assessing their degree of similarity.

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	1	0.5	N/A	1	1	N/A
Adv₂	1	1	N/A	0.75	1	N/A
Adv₃	1	1	N/A	0.75	1	N/A
Adv₄	1	1	N/A	0.75	1	N/A

Table 2.4: Recall and precision based on common properties

Similarly to the previous case, we can use the ratio of common properties between two classes, as a measure of their similarity. In particular, we can estimate recall and precision by:

$$\begin{aligned}
 recall_{II}(C_R, C_A) &= \frac{|P(C_R) \cap P(C_A)|}{|P(C_A)|} \\
 precision_{II}(C_R, C_A) &= \frac{|P(C_R) \cap P(C_A)|}{|P(C_R)|}
 \end{aligned} \tag{2.5}$$

For instance, the properties of *Movie_Adv* are *hasDirector* and *hasActor*, inherited from class *Movie*. *Movie_Req* also inherits these two properties, but also has the additional property *hasLeadActor*. Thus, applying Equations 2.5 we get:

$$\begin{aligned}
 recall_{II}(Movie_Req, Movie_Adv) &= 2/2 = 1 \\
 precision_{II}(Movie_Req, Movie_Adv) &= 2/3 \simeq 0.67
 \end{aligned}$$

Table 2.4 illustrates the results from Equations 2.5 for the sample services of the motivating example. Notice how the values change compared to Table 2.3. For instance, the recall between *PopMovie* and *PopMovieEuro* is now equal to one, because the two classes share the same properties. Similarly, the precision between *PopMovie* and *Movie* is now higher, because their common properties are again considered. Equations 2.5 are not applicable for the class *Showtimes* and its subclasses, because no properties are assigned to them.

Property hierarchy. Next, the existence of property hierarchy is considered. In order to calculate the recall and precision between two classes based on their set of properties, we first calculate the recall and precision between their individual properties. The calculation of recall and precision for two properties, with respect to the specified property hierarchy, is done similarly to the respective calculation for classes, namely by the ratio of their common superproperties:

$$\begin{aligned}
 recall(P_1, P_2) &= \frac{|A(P_1) \cap A(P_2)|}{|A(P_2)|} \\
 precision(P_1, P_2) &= \frac{|A(P_1) \cap A(P_2)|}{|A(P_1)|}
 \end{aligned} \tag{2.6}$$

As an example, we calculate recall and precision for properties *hasLeadActor* and *hasActor*. The superproperties of the first are *hasLeadActor* and *hasActor*, while the second has one superproperty, namely itself. Therefore, from Equations 2.6 follows:

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	1	0.5	N/A	1	1	N/A
Adv₂	1	1	N/A	0.875	1	N/A
Adv₃	1	1	N/A	0.875	1	N/A
Adv₄	1	1	N/A	0.875	1	N/A

Table 2.5: Recall and precision considering property hierarchy

$$\begin{aligned} \text{recall}(\text{hasLeadActor}, \text{hasActor}) &= 1/1 = 1 \\ \text{precision}(\text{hasLeadActor}, \text{hasActor}) &= 1/2 = 0.5 \end{aligned}$$

The recall and precision between two properties may be used to calculate the recall and precision between two classes. For each property of one class, the property of the other class that has the highest recall or precision, accordingly, is found (if any) and then, the average is estimated by Equations 2.7. Notice that if no subproperties exist, then Equations 2.7 are equivalent to Equations 2.5.

$$\begin{aligned} \text{rec}_{II}(C_R, C_A) &= \frac{\sum_{i=1}^{|P(C_A)|} \max_{j=1}^{|P(C_R)|} (\text{rec}(p_j(C_R), p_i(C_A)))}{|P(C_A)|} \\ \text{prc}_{II}(C_R, C_A) &= \frac{\sum_{i=1}^{|P(C_R)|} \max_{j=1}^{|P(C_A)|} (\text{prc}(p_i(C_R), p_j(C_A)))}{|P(C_R)|} \end{aligned} \quad (2.7)$$

For the reference example, considering that *hasLeadActor* is a subproperty of *hasActor*, the precision between *PopMovie* and *Movie* is:

$$\text{precision}(\text{PopMovie}, \text{Movie}) = (1 + 1 + 1 + 0.5)/4 = 0.875$$

Table 2.5 displays the updated results for recall and precision, after the property hierarchy has been considered.

Value restrictions. Often one class extends another not (only) by defining additional properties, but (also) by imposing restrictions on the values of already existing properties. To consider this, we extend Equations 2.6 to account also for the recall and precision between the ranges of the properties. As it is typically the case for aggregating ratios, the geometric mean of the similarity of the properties and the similarity of their ranges is used:

$$\begin{aligned} \text{rec}(P_1, P_2) &= \sqrt{\frac{|A(P_1) \cap A(P_2)|}{|A(P_2)|} * \text{rec}(r(P_1), r(P_2))} \\ \text{prc}(P_1, P_2) &= \sqrt{\frac{|A(P_1) \cap A(P_2)|}{|A(P_1)|} * \text{prc}(r(P_1), r(P_2))} \end{aligned} \quad (2.8)$$

where $r(P)$ denotes the (possibly restricted) range of property P . It is often the case that two different, unrelated properties have the same (or similar) ranges. However,

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	0.9375	0.5	N/A	1	1	N/A
Adv₂	1	1	N/A	0.875	1	N/A
Adv₃	1	1	N/A	0.875	1	N/A
Adv₄	1	1	N/A	0.875	1	N/A

Table 2.6: Recall and precision including value restrictions

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	0.927	0.5	N/A	1	1	N/A
Adv₂	1	1	N/A	0.875	0.85	N/A
Adv₃	1	1	N/A	0.875	1	N/A
Adv₄	1	1	N/A	0.875	1	N/A

Table 2.7: Recall and precision based on properties, including hierarchy and restrictions

in such cases the similarity between the ranges of the properties should not be taken into account. Indeed, if the compared properties have no common superproperties, then the geometric mean is zero.

The recall and precision between two classes is now derived from Equations 2.7, using Equations 2.8 instead of 2.6 to calculate the recall and precision of their properties.

For example, considering value restrictions, the recall between *PopMovieEuro* and *PopMovie* is:

$$recall(PopMovie, PopMovieEuro) = (1 + \sqrt{1 * 0.5} + 1 + 1)/4 = 0.927$$

The values of recall and precision for the services of the reference example, after considering value restrictions, are displayed in Table 2.6

Cardinality restrictions. Apart from restricting the value of a property, it is also possible to restrict its minimum and/or maximum cardinality. With respect to minimum cardinality restrictions, the similarity could be assessed by subtracting the minimum cardinalities of the two properties, using zero as the default value. However, this is not applicable for maximum cardinality restrictions, since the upper bound in this case is infinite.

To get around this, when a cardinality restriction on a property P is encountered in the calculation, we create a (temporary) subproperty of P and use it to replace this restriction. If both of the compared classes define a cardinality restriction on the same property, the substitute properties are hierarchically structured according to their respective lower and/or upper bounds. That is, assuming two cardinality restrictions on property P , denoted by $P_{min_1}^{max_1}$ and $P_{min_2}^{max_2}$, if, for instance, $min_1 > min_2$ and $max_1 < max_2$, then the substitute properties will be $P_1 \sqsubset P_2 \sqsubset P$. The calculation of recall and precision is then done as previously, using Equations 2.7 and 2.8.

For example, considering cardinality restrictions, the precision between *Multiplex* and *Cinema* is:

$$precision(Multiplex, Cinema) = (\sqrt{0.5 * 1} + 1)/2 = 0.85$$

Table 2.7 displays the values of recall and precision for the reference example, based on the properties of the classes, and considering the property hierarchy, as well as value and cardinality restrictions.

2.3.2.3 Matching preconditions and effects

Up to now we have dealt with the matching of input and output parameters. However, a service description may also consist of a set of *preconditions* and *effects*, which describe the *state change* produced by the execution of the service. The preconditions of a service is a set of logical formulae, all of which must hold in order for the service to execute successfully. Effects is another set of formulae, which hold after the successful execution of the service. In the following we present how the use of recall and precision is extended for matching between request and advertisement preconditions and effects. We use ϕ_{Rpr} , ϕ_{Ref} , ϕ_{Apr} and ϕ_{Aef} to denote, respectively, the sets of preconditions and effects specified by the user and the service. Regarding service effects, a service advertisement satisfies the user needs, if all the effects requested by the user are true under the effects provided by the service, i.e.,

$$\forall \phi_i \in \phi_{Ref} : \phi_{Aef} \models \phi_i$$

We denote by ϕ'_{Ref} the subset of ϕ_{Ref} containing the effects satisfied by the service. On the other hand, the service may have additional, possibly undesirable, effects. Thus, we need to distinguish between the effects of the service that are actually required to satisfy the request and other “side-effects”. We denote by ϕ'_{Aef} the subset of ϕ_{Aef} containing the effects ϕ_i for which the following holds:

$$\exists \phi_j \in \phi_{Ref} : \phi_{Aef} \setminus \{\phi_i\} \not\models \phi_j$$

Therefore, we extend the definition of recall and precision for matching service effects as follows:

Recall_{ef} is the proportion of effects requested by the user that are satisfied by the service.

Precision_{ef} is the proportion of service effects required to satisfy the effects requested by the user.

Formally:

$$\begin{aligned} recall_{ef}(\phi_{Ref}, \phi_{Aef}) &= |\phi'_{Ref}| / |\phi_{Ref}| \\ precision_{ef}(\phi_{Ref}, \phi_{Aef}) &= |\phi'_{Aef}| / |\phi_{Aef}| \end{aligned} \tag{2.9}$$

Regarding service preconditions, the only difference is that in order to use the service, the user must provide sufficient conditions to satisfy the service preconditions. Therefore, the preconditions required by the service are considered as relevant, while those provided by the user are considered as retrieved. Thus, using corresponding notation, we have:

Recall_{pr} is the proportion of service preconditions that are satisfied by the user.

Precision_{pr} is the proportion of conditions provided by the user that are needed to satisfy the service preconditions.

Formally:

$$\begin{aligned} recall_{pr}(\phi_{Rpr}, \phi_{Apr}) &= |\phi'_{Apr}| / |\phi_{Apr}| \\ precision_{pr}(\phi_{Rpr}, \phi_{Apr}) &= |\phi'_{Rpr}| / |\phi_{Rpr}| \end{aligned} \tag{2.10}$$

	PRECONDITIONS	EFFECTS
Req	ϕ_1, ϕ_2	ϕ_4
Adv₁	ϕ_1	ϕ_3, ϕ_4
Adv₂	ϕ_1, ϕ_2	ϕ_4
Adv₃	ϕ_1	ϕ_4
Adv₄	ϕ_1, ϕ_2	ϕ_3, ϕ_4

Table 2.8: *Preconditions and effects for the motivating example*

	RECALL		PRECISION	
	PREC.	EFF.	PREC.	EFF.
Adv₁	1	1	0.5	0.5
Adv₂	1	1	1	1
Adv₃	1	1	0.5	1
Adv₄	1	1	1	0.5

Table 2.9: *Recall and precision values for the preconditions and effects*

To demonstrate the matching process for service preconditions and effects, we extend the motivating example, by assuming that the candidate services allow also for ticket reservations. Suppose that the request and service preconditions and effects are as shown in Table 2.8, where the logical formulae ϕ_1 , ϕ_2 , ϕ_3 and ϕ_4 express, respectively, the following conditions: “the user possesses a valid credit card”, “the reservation is done 3 days in advance”, “the credit card is charged” and “the tickets are reserved”.

Table 2.9 displays the values of recall and precision for matching the preconditions and effects of the services of the motivating example.

2.3.2.4 Combining partial results

The partial results described in the previous two sections are useful for providing a more in-depth analysis of the match between the service request and the service advertisement, allowing for higher control in the selection process and providing useful insight in refining the search criteria. However, an aggregated result is also required, so that a single list of ranked services may be returned to the user. A single measure indicating the degree of match between the user request and the service advertisement is derived by combining the partial results, using appropriate weights to determine the relative emphasis given on each parameter.

The first step is to combine the values of recall and precision obtained by the class hierarchy and the class properties (if any), using the geometric mean:

	RECALL			PRECISION		
	IN ₁	OUT ₁	OUT ₂	IN ₁	OUT ₁	OUT ₂
Adv₁	0.79	0.58	0.5	1	1	1
Adv₂	1	1	1	0.66	0.65	1
Adv₃	1	1	1	0.66	1	1
Adv₄	1	1	0.5	0.66	1	1

Table 2.10: *Recall and precision combining class hierarchy and properties*

	RECALL		PRECISION	
	IN	OUT	IN	OUT
Adv₁	0.79	0.54	1	1
Adv₂	1	1	0.66	0.83
Adv₃	1	1	0.66	1
Adv₄	1	0.75	0.66	1

Table 2.11: Recall and precision for multiple input/output parameters

$$\begin{aligned}
recall &= \sqrt{recall_I * recall_{II}} \\
precision &= \sqrt{precision_I * precision_{II}}
\end{aligned} \tag{2.11}$$

The results regarding the reference example are illustrated in Table 2.10.

Equations 2.11, as well as 2.4 and 2.7, refer to a single input or output parameter. However, a Web service typically has multiple inputs and outputs. In this case, the best match is identified for each parameter, and then the average match over all parameters is returned. That is, if S_{R_I} and S_{A_I} are the sets of request and service inputs, respectively, then:

$$\begin{aligned}
rec_{in}(S_{R_I}, S_{A_I}) &= \frac{\sum_{C_R \in S_{R_I}} \max_{C_A \in S_{A_I}} (rec(C_R, C_A))}{|S_{R_I}|} \\
prc_{in}(S_{R_I}, S_{A_I}) &= \frac{\sum_{C_R \in S_{R_I}} \max_{C_A \in S_{A_I}} (prc(C_R, C_A))}{|S_{R_I}|}
\end{aligned} \tag{2.12}$$

Matching multiple outputs is done similarly. Notice that a weighted average can be used instead, so that the user may give different emphasis on the various parameters.

Table 2.11 displays the results for the reference example, aggregating the values of recall and precision for the two output parameters.

Finally, the results from matching inputs, outputs, preconditions and effects are combined, using appropriate weights to determine the relative emphasis of each factor.

$$\begin{aligned}
rec &= \frac{w_{in} \cdot rc_{in} + w_{out} \cdot rc_{out} + w_{pr} \cdot rc_{pr} + w_{ef} \cdot rc_{ef}}{w_{in} + w_{out} + w_{pr} + w_{ef}} \\
prc &= \frac{w_{in} \cdot pr_{in} + w_{out} \cdot pr_{out} + w_{pr} \cdot pr_{pr} + w_{ef} \cdot pr_{ef}}{w_{in} + w_{out} + w_{pr} + w_{ef}}
\end{aligned} \tag{2.13}$$

The weights in Equations 2.12) and 2.13 reflect the user preferences. That is, a more advanced search interface may be presented to the user, allowing the association to each search parameter (or type of parameters) a degree of importance, indicated in a pre-defined, discrete scale (e.g. “low”, “medium”, “high”). Also the

	RECALL	PRECISION	F-MEASURE
Adv₁	0.81	0.75	0.79
Adv₂	1	0.89	0.96
Adv₃	1	0.86	0.95
Adv₄	0.92	0.78	0.87

Table 2.12: *Recall and precision combining input, output, preconditions and effects*

user may omit some parameter type(s), in which case the corresponding weight is set to zero. For instance, if the user does not specify any preconditions, then $w_{pr} = 0$.

To obtain a single score for each service advertisement, the values of recall and precision are combined using the F-measure (see Equation 2.2).

The final results for the motivating example are displayed in Table 2.12. For this example, we have assumed that all input and output parameters are considered of equal importance. Outputs and effects are weighted twice as much as inputs and preconditions, i.e. $w_{out} = w_{ef} = 2 w_{in} = 2 w_{pr}$. The reason for this is that mismatches regarding inputs and preconditions may be resolved by the user, by providing additional information, probably by using the results of other services. For the F-measure, the value $\alpha = 2$ was selected, so that recall is weighted twice as much as precision.

As shown in Table 2.12, the advertisements *Adv₂* and *Adv₃* advertisements *Adv₂* and *Adv₃* provide the best matches, with the first having a slightly higher rank. Indeed, as can be observed from Tables 2.1 and 2.8, the capabilities of these services are a superset of the requested capabilities. *Adv₂* has a lower precision regarding output parameters (see Table 2.11), however, this is compensated by a higher precision regarding preconditions (see Table 2.9).

2.3.3 Properties of the Similarity Measure

In the following we discuss several important properties of the presented similarity measure for Semantic Web services.

Similar to previous works in this area [118, 93], the proposed discovery mechanism uses the available semantic information provided by the domain ontology and performs logic inference to estimate the degree of match between the service capabilities and the user request. However, the degree of match is not expressed in a discrete scale, e.g., comprising four distinct types of match, but as a continuous value in the range [0..1]. This is important for handling cases where a large number of candidate services provide the same type of match. The use of recall and precision, as measures for the degree of match, allows to provide a ranking of the available services, while, at the same time, maintaining a straight-forward correspondence to these established types of match.

Another significant advantage of using the recall and precision measures is that they provide an intuitive and modular way to allow for asymmetry, an important requirement for a matching function [35]. Moreover, it is shown how the use of these measures is extended to handle in a uniform way the matching of service preconditions and effects. Finally, as argued in [118], a requirement for the matching process is to encourage advertisers to be honest with their descriptions. Through the use of recall and precision this is accomplished, as the service provider is obliged to strike a balance between these two factors in order to achieve a high rank.

The assessment of semantic similarity between request and advertisement parameters exploits the semantic information encoded both in the class hierarchy and the properties of the classes, including hierarchy of properties and value or cardinality restrictions. Therefore the proposed method is suitable for both applications relying on a taxonomy, as well as applications employing more expressive ontologies, such as OWL ontologies. A property of the matching function is that the assessed similarity is higher for concepts residing deeper in the concept hierarchy. For instance, consider two classes C_1 and C_2 , such that $C_1 \sqsubseteq C_2$, $|A(C_2)| = |A(C_2) \sqcap A(C_1)| = n$ and $|A(C_1)| = n + k$. Then the precision between C_1 and C_2 based on the class hierarchy is: $precision(C_1, C_2) = n/(n + k) = 1/(1 + k/n)$, which is an increasing function of n . Similar holds for matching based on properties, as well as matching preconditions and effects. The intuition for having this property is that reaching a high depth in the hierarchy means that the search process has reached a high level of granularity and thus differences encountered between the concepts at this level are more likely to be easier to compromise.

Moreover, the proposed ranking mechanism is flexible and customizable, allowing the consideration of user preferences. This refers to two aspects. First, by means of an advanced search interface, the user may determine the relative importance of each search parameter. Second, apart from presenting a single rank for each candidate service, more detailed results may also be provided (e.g., separate values for recall and precision or the degree of match for specific parameters), to facilitate the user in identifying the most suitable service or refining the search criteria.

Finally, notice that we have assumed that the service request and advertisements are annotated using the same ontology. However, committing to a common ontology is not always feasible. Even though the problem of matching Web services without a common ontology has been considered in previous works (e.g., [35]), we believe that it is an orthogonal issue. More specifically, applying ontology mapping techniques (see [76] for a comprehensive survey), it is possible to identify appropriate mappings between the involved ontologies and define the terms of the one ontology by means of the terms of the other. Then, once these definitions are available, the matching mechanism can be applied.

2.4 Selection of Services with Skyline Queries

2.4.1 Skyline Services

In the previous section we presented a similarity measure for Semantic Web services. The last step in calculating the similarity between a requested and an offered service was to aggregate the pairwise similarities between the parameters of the two descriptions. For this purpose, appropriate weights for the various parameters were assumed. This is also the case in other related work in the literature, where the final degree of match is typically determined either as the worst match over all parameters or as a (weighted) average of the partial results. Clearly, this approach relies on a priori knowledge of the user's preferences; otherwise, ad hoc values have to be assigned to the involved weights, leading potentially to biased results. This may have a significant impact on the perceived quality of the discovery process: in the case of a human user, a low precision on the top returned matches compromises the credibility of the discovery engine; even worse, in a fully automated scenario,

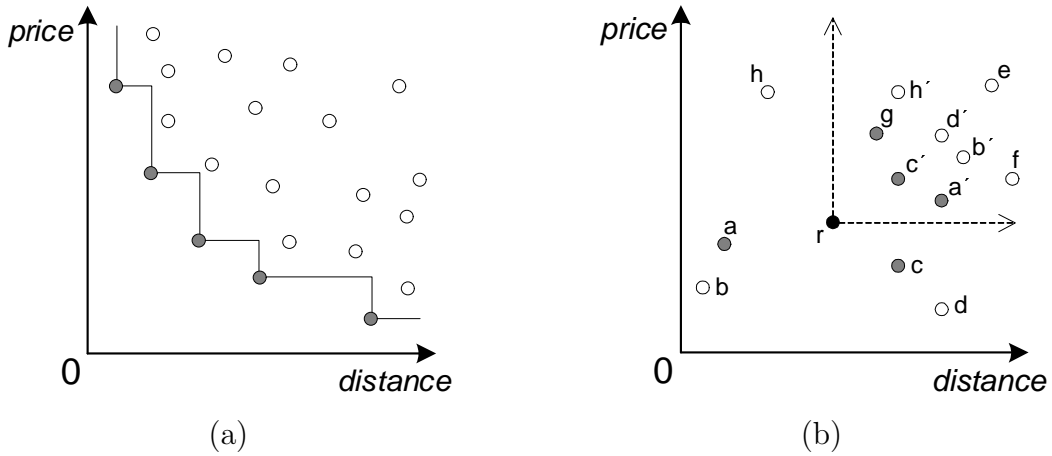


Figure 2.4: Illustration of (a) static and (b) dynamic skylines

the software agent is expected to make a selection among the top returned services, thus, choosing an inappropriate service breaks the whole workflow.

In this section we address this shortcoming by employing a service discovery approach based on the notion of *skyline* [28]. This allows us to define and compute the *potentially interesting* services for *any* type of user.

A typical example used to illustrate the concept of skyline is searching for cheap hotels close to the beach. A sample set of hotels is depicted in Figure 2.4(a), characterized by two dimensions, *distance* and *price*. The drawn line indicates the skyline. A hotel belongs in the skyline if there is no other hotel that is better than it in both dimensions, i.e., *both* cheaper and closer to the beach. The distinguishing property of the skyline is that for any preference function f that is monotone on all attributes, if an object maximizes f , then this object is part of the skyline. Also, for every object in the skyline, there exists a monotone preference function that is maximized by this object. Intuitively, this means that (a) regardless of how a user weighs his/her preferences, his/her top preferred object will be one of the skyline objects, and (b) there is no skyline object which is nobody's top preference. Further, the *dynamic skyline* is a variation, where the original objects are compared with respect to a given reference object r [120]. The reference object defines a new space, depicted as the inner coordinate system in Figure 2.4(b), and existing objects need to be transformed in this space. In this example, points a, c, b, d, h are projected to a', c', b', d', h' respectively (i.e., $p'_x = |p_x - r_x|, p'_y = |p_y - r_y|$). The dynamic skyline for the reference hotel r contains a', c' and g (i.e., a, c), but not b', d', e, f and h' (i.e., not b, d, h).

The analogy to our case is as follows: the space dimensions correspond to the service parameters being matched; the objects in the space correspond to the offered services; the reference object corresponds to the user's request.

In the following, we first give the formal definitions of skyline and dynamic skyline, and then we formalize the problem of selecting the best candidate services for fulfilling a user's request as a dynamic skyline computation problem. We also present an illustrative example to clarify these notions.

2.4.1.1 Background

Consider a set of points P in a d -dimensional space, with p^i denoting the value of point $p \in P$ in the i -th dimension.

Definition 2.4.1. (Dominance) A point $p \in P$ dominates another point $q \in P$, denoted as $p \prec q$, iff p is as good or better than q in all dimensions and better in at least one dimension, i.e., $\forall i \in [1, d] : p^i \leq q^i$ and $\exists i \in [1, d] : p^i < q^i$.

Definition 2.4.2. (Skyline) The skyline of P , denoted by SL_P , comprises the set of points in P that are not dominated by any other point, i.e., $SL_P = \{p \in P \mid \nexists q \in P : q \prec p\}$.

Definition 2.4.3. (Dynamic Dominance) Given a reference point $r \in P$, a point $p \in P$ dominates another point $q \in P$ w.r.t. r , denoted as $p \prec^r q$, iff $\forall i \in [1, d] : |r^i - p^i| \leq |r^i - q^i|$ and $\exists i \in [1, d] : |r^i - p^i| < |r^i - q^i|$.

Definition 2.4.4. (Dynamic Skyline) Given a reference point $r \in P$, the dynamic skyline of P w.r.t. r , denoted by SL_P^r , comprises the set of points in P that are not dynamically dominated by any other point w.r.t. r , i.e., $SL_P^r = \{p \in P \mid \nexists q \in P : q \prec^r p\}$.

2.4.1.2 Problem formulation

The functional part of a Semantic Web service can be described by a tuple $SWS = (I, O, P, E)$, where I, O, P, E are sets of inputs, outputs, preconditions, and effects, with each parameter semantically annotated by means of an associated ontology \mathcal{O} . We assume that the languages OWL and OWL-S are used to represent, respectively, the domain ontology and the requested and offered services. Matching a service request R with a service offer S is based on matching the individual parameters in the two descriptions. For this purpose, a semantic matching function f_m is used. For input and output parameters the degree of match is typically determined by checking for equivalence or subsumption relationship between the corresponding classes in the ontology \mathcal{O} [118, 93]. Notice that if a similarity measure like the one proposed in Section 2.3 is used, which allows for results in a continuous interval of values (i.e., $[0, 1]$), then it is straightforward to obtain different degrees of match through appropriate quantization. In fact, in this case it is possible to choose the number of distinct degrees of match according to the desired level of granularity for the matching function. In the rest of this section, for simplicity, we assume the following degrees of match, in decreasing order: $DM = \{exact, direct_subclass, subclass, direct_superclass, superclass, sibling, fail\}$. Notice that the distinction between direct subclass (superclass) and subclass (superclass) refers to whether the considered subsumption relationship is explicitly stated in the ontology or inferred by the reasoner (e.g., by transitivity.) Different ordering or variations of these degrees may also be meaningful in different applications and contexts [93]. Our approach is generic and does not depend on this particular assumption. Preconditions and effects are represented by logical formulae and are matched by checking for logical implication between them. The results of the match in this case is *exact* or *fail*, depending on whether such an implication holds or not.

The inputs and preconditions of the request should match those of the service, while the service outputs and effects should match those of the request. Thus, applying the function f_m to pairs of corresponding parameters from the requested and

<p style="text-align: center; margin: 0;">Algorithm Match(R, S)</p> <p>Input: request R, offer S Output: the match vector MV</p> <pre style="margin: 0;"> 1 begin 2 $I_R \leftarrow$ inputs of R , $O_R \leftarrow$ outputs of R 3 $I_S \leftarrow$ inputs of S , $O_S \leftarrow$ outputs of S 4 $MV \leftarrow$ add MatchIn (I_R, I_S) 5 $MV \leftarrow$ add matchOut (O_R, O_S) 6 return MV 7 end </pre>	<p style="text-align: center; margin: 0;">MatchIn(I_R, I_S)</p> <p>Input: requested (I_R) and offered (I_S) inputs Output: the input match vector IMV</p> <pre style="margin: 0;"> 1 begin 2 $hasMatch \leftarrow$ new array() 3 for $I \in I_R$ do 4 $tmpMatches \leftarrow$ new array() 5 for $J \in I_S$ do 6 $m \leftarrow$ DegreeOfMatch (I, J) 7 $tmpMatches \leftarrow$ add m 8 if $m \neq$ "fail" then 9 $hasMatch \leftarrow$ add J 10 $IMV \leftarrow$ add 11 $\max\{tmpMatches\}$ 12 if $hasMatch$ containsAll I_S 13 then 14 $IMV \leftarrow$ add "exact" 15 else $IMV \leftarrow$ add "fail" 16 return IMV 17 end </pre>
<p style="text-align: center; margin: 0;">MatchOut(O_R, O_S)</p> <p>Input: requested (O_R) and offered (O_S) outputs Output: the output match vector OMV</p> <pre style="margin: 0;"> 1 begin 2 for $I \in O_R$ do 3 $tmpMatches \leftarrow$ new array() 4 for $J \in O_S$ do 5 $m \leftarrow$ DegreeOfMatch (I, J) 6 $tmpMatches \leftarrow$ add m 7 $OMV \leftarrow$ add $\max\{tmpMatches\}$ 8 return OMV 9 end </pre>	

Table 2.13: Matching service I/O s

offered service, results in a match vector $MV \in DM^k$, $k = |MV| = |S_I| + |S_P| + |R_O| + |R_E|$. However, the number, as well as the order, of the parameters may vary among the set of available services, rendering the match vectors not comparable. To deal with this problem, i.e., to fix the number and the order of the dimensions, we use as reference dimensions the ones specified by the user's request. Still, two issues need to be resolved in this case. First, the same request input/precondition may provide a match for more than one service inputs/preconditions. Then, the best degree of match is considered for the corresponding position in MV . Second, it is possible that not all service inputs/preconditions are matched. To capture this, we introduce two additional fields in MV , corresponding respectively to inputs and preconditions, with the values *exact* or *fail*, indicating accordingly whether there exists a parameter that has not been matched (alternatively, the number of parameters that failed to match can be used). Thus, the size of MV becomes $|MV| = |R_I| + |R_O| + |R_P| + |R_E| + 2$ (i.e., fixed for a given R). The matching algorithm, $Match(R, S)$ is presented in detail in Table 2.13. The function $DegreeOfMatch(I, J)$ uses a reasoner to determine the degree of match between the ontology concepts I, J . For brevity, we only consider inputs and outputs; preconditions and effects can be matched accordingly.

We can now define the notions of (dynamic) dominance and (dynamic) skyline for Semantic Web services selection.

Definition 2.4.5. (Service Dominance) Given a set of Semantic Web services

\mathcal{S} and a request R , a service $S_1 \in \mathcal{S}$ dominates another service $S_2 \in \mathcal{S}$ with respect to R , denoted as $S_1 \prec^R S_2$, iff $\forall i \in [1, |MV_{R,S_1}|] : MV_{R,S_1}^i \leq MV_{R,S_2}^i$ and $\exists i \in [1, |MV_{R,S_1}|] : MV_{R,S_1}^i < MV_{R,S_2}^i$.

Definition 2.4.6. (Skyline Services) Given a set of Semantic Web services \mathcal{S} and a request R , the skyline services of \mathcal{S} with respect to R , denoted by $SL_{\mathcal{S}}^R$, are those not dominated by another service with respect to R : $SL_{\mathcal{S}}^R = \{S \in \mathcal{S} \mid \nexists S' \in \mathcal{S} : S' \prec^R S\}$.

2.4.1.3 Illustrative Example

Assume a sample service request and six available services, as shown in Table 2.14(a). For simplicity, we consider only input and output parameters, which are classes from the hierarchy depicted in Figure 2.14(b). The derived match vectors are presented in Table 2.14(c), with IN_X indicating whether all service inputs are matched or not. For instance, in the case of S_4 , the provided input C_6 provides a *direct_superclass* match with C_{10} and a *superclass* match with C_{14} . Thus, $MV_{R,S_4}^{IN_1} = \textit{direct_superclass}$ and $MV_{R,S_4}^{IN_X} = \textit{exact}$. For the service S_2 , $MV_{R,S_2}^{IN_1} = \textit{fail}$ and $MV_{R,S_2}^{IN_X} = \textit{fail}$, since the request input C_6 does not provide a match for the service input C_9 . The rest of the results can be verified similarly.

Given the match vectors shown in Table 2.14(c), the problem is to identify the best matches. It can be seen that even for such a small number of services this is no trivial task. For this purpose, we consider as best matches those services that belong in the skyline for the given request. Based on the definitions in Section 2.4.1, we can conclude that (a) the services S_2 , S_4 and S_5 are dominated by both S_1 and S_3 , (b) S_6 is dominated by S_3 , and (c) S_1 and S_3 are not dominated by any service. Therefore, S_1 and S_3 constitute the skyline, i.e., the best matches, for the request R .

One might argue that S_1 constitutes an “overall” better match than S_3 , given that *direct_subclass* indicates a closer match than *subclass*. However, this would only be true for users with an equal preference on both output parameters or a higher preference on OUT_1 . Instead, a user concerned about parameter OUT_2 would probably be more interested in the service S_3 . Selecting the skyline services guarantees the retrieval of the best matches regardless of user preferences.

2.4.2 Selection of the Best Candidates

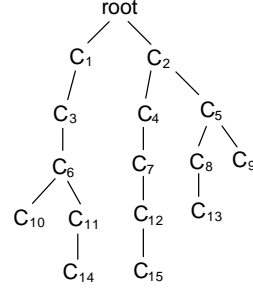
2.4.2.1 Main algorithm

We leverage work existing in the database literature, and in particular the Bitmap algorithm, introduced in [158]. Since the semantic match between requested and offered services is typically expressed by a small set of discrete degrees of match, as discussed in the previous section, the choice of the Bitmap algorithm is natural, as it is especially designed for discrete, low cardinality domains. Specifically, it employs a bitmap representation to encode the data points, and uses bit-wise operations to determine the skyline. The efficiency of the algorithm relies on the high speed of bit-wise operations. Note that, even though more efficient skyline algorithms have been proposed (e.g., [120]), they rely on the assumption that the data set is indexed.

The skyline service selection algorithm works as follows. First, the match vectors are translated to an appropriate bitmap representation. In fact, to avoid any extra

	INPUTS	OUTPUTS
R	C_6	C_7, C_8
S₁	C_3	C_5, C_7
S₂	C_9	C_2, C_9
S₃	C_3	C_2, C_8
S₄	C_{10}, C_{14}	C_{15}
S₅	C_1	C_6
S₆	-	C_6, C_8, C_9

(a) Service request and offers



(b) A sample class hierarchy

	IN ₁	IN _X	OUT ₁	OUT ₂
S₁	<i>dir_subcls</i>	<i>exact</i>	<i>exact</i>	<i>dir_subcls</i>
S₂	<i>fail</i>	<i>fail</i>	<i>subcls</i>	<i>sibling</i>
S₃	<i>dir_subcls</i>	<i>exact</i>	<i>subcls</i>	<i>exact</i>
S₄	<i>dir_supercls</i>	<i>exact</i>	<i>supercls</i>	<i>fail</i>
S₅	<i>subcls</i>	<i>exact</i>	<i>fail</i>	<i>fail</i>
S₆	<i>fail</i>	<i>exact</i>	<i>fail</i>	<i>exact</i>

(c) The resulting match vectors

Table 2.14: *Illustrative example*

	IN ₁	IN _X	OUT ₁	OUT ₂
S₁	0111111	1111111	1111111	0111111
S₂	0000001	0000001	0011111	0000011
S₃	0111111	1111111	0011111	1111111
S₄	0001111	1111111	0000111	0000001
S₅	0011111	1111111	0000001	0000001
S₆	0000001	1111111	0000001	1111111

Table 2.15: *Bitmap representation*

$$A^1 = 101110 \quad B^1 = 101010$$

$$A^2 = 101111 \quad B^2 = 000000$$

$$A^3 = 111100 \quad B^3 = 111000$$

$$A^4 = 111111 \quad B^4 = 111001$$

$$A = 101100 \quad B = 111011$$

$$A \& B = 101000$$

Table 2.16: *Dominance check for S₄*

overhead, this step can be integrated with the matching phase, i.e., the result of the matcher can be directly encoded in this representation. Then, each match vector is checked for dominance against all other match vectors. The latter step is efficiently performed by fast bit-wise AND/OR operations on the bitmap representations obtained in the former step.

Obtaining the bitmap representation. We assume the dominance relationship described in Section 2.4.1.2, and we assign the values $\{1, 2, \dots, 7\}$ to the 7 possible

<p style="text-align: center; margin: 0;">Algorithm Skyline(R, \mathcal{S})</p> <p>Input: request R, offers \mathcal{S} Output: skyline services $SL_{\mathcal{S}}^R$</p> <pre style="margin: 0;"> 1 begin 2 for $S \in \mathcal{S}$ do 3 $MV \leftarrow MV \cup \text{Match}(R, S)$ 4 $bm \leftarrow \text{BuildBitmap}(MV)$ 5 for $S_j \in \mathcal{S}$ do 6 $\mathcal{D}_{S_j} \leftarrow \text{DominatedBy}(j, bm)$ 7 if \mathcal{D}_{S_j} <i>is empty</i> then 8 $SL_{\mathcal{S}}^R \leftarrow SL_{\mathcal{S}}^R \cup S_j$ 9 return $SL_{\mathcal{S}}^R$ 10 end </pre>	<p style="text-align: center; margin: 0;">DominatedBy(j, bm)</p> <p>Input: service index j, bitmaps bm Output: services \mathcal{D}_{S_j} that dominate S_j</p> <pre style="margin: 0;"> 1 begin 2 $\{A, B\} \leftarrow \text{ABvectors}(j, bm)$ 3 for $S_k \in \mathcal{S}$ do 4 if $(A \& B)[k]$ <i>is set</i> then 5 $\mathcal{D}_{S_j} \leftarrow \mathcal{D}_{S_j} \cup S_k$ 6 return \mathcal{D}_{S_j} 7 end </pre>
<p style="text-align: center; margin: 0;">ABvectors(j, bm)</p> <p>Input: service index j, bitmaps bm Output: vectors A, B for service S_j</p> <pre style="margin: 0;"> 1 begin 2 $A \leftarrow \mathbf{1}$ // mask with all 1s 3 $B \leftarrow \mathbf{0}$ // mask with all 0s 4 for $i \in [1, MV_{S_j}]$ do 5 $q \leftarrow MV_{S_j}^i$ 6 $A^i \leftarrow \text{BitSlice}(q, i)$ 7 $B^i \leftarrow \text{BitSlice}(q-1, i)$ 8 $A \leftarrow A \& A^i$ 9 $B \leftarrow B B^i$ 10 return $\{A, B\}$ 11 end </pre>	<p style="text-align: center; margin: 0;">Dominates(j, bm)</p> <p>Input: service index j, bitmaps bm Output: services \mathcal{D}_{S_j} that S_j dominates</p> <pre style="margin: 0;"> 1 begin 2 $\{A, B\} \leftarrow \text{ABvectors}(j, bm)$ 3 for $S_k \in \mathcal{S}$ do 4 if $(A B)[k]$ <i>is not set</i> then 5 $\mathcal{D}_{S_j} \leftarrow \mathcal{D}_{S_j} \cup S_k$ 6 return \mathcal{D}_{S_j} 7 end </pre>

Table 2.17: Determining the skyline services

degrees of match, with 1 corresponding to *exact* and 7 to *fail*¹. We represent these values in a bitmap of size 7, as follows: if $q \in \{1, 2, \dots, 7\}$ is the degree of match, then its bitmap representation has value 0 for the bits 1 to $q - 1$, and 1 for the bits q to 7. For example, an *exact* degree of match, i.e., value 1, is represented as 1111111, whereas a *sibling* degree of match, i.e., value 6, is represent as 0000011. Returning to our running example, the corresponding bitmap representations are depicted in Table 2.15 (the function of the bold and italicized bits will be discussed in the following).

Checking for dominance. Determining whether a service belongs to the skyline involves extracting vertical *bitslices* and performing bitwise AND/OR operations. This process is best illustrated through our running example. Assume we wish to discover whether service S_4 with match vector $MV_{S_4} = (4, 1, 5, 7)$ is part of the skyline. For each field $i \in [1, |MV_{S_4}|]$ of MV_{S_4} , two vertical bitslices, A^i and B^i , are extracted. In particular, letting $q = MV_{S_4}^i$, we obtain the bitslice A^i (resp., B^i) by juxtaposing the q -th (resp., the preceding $(q - 1)$ -th) bit of the i -th field

¹The adaptation to different degrees of match and dominance relationships is straightforward.

for all services. Note that when $q - 1 < 1$, B^i is explicitly set to all zeros. Since $MV_{S_4}^1 = 4$ the bitslice $A^1 = 101110$ is obtained by juxtaposing the 4th bits of the first field for all services. Similarly, $B^1 = 101010$ is obtained by juxtaposing the 3rd bits. Table 2.15 shows the A^i bitslices in bold typeface; the B^i bitslices are shown italicized ($B^2 = 000000$ is omitted).

Assume a service request R and an offer S . Observe that the bitslice A^i of S encodes which services (i.e., those whose bit is set) are equally as good or better matches than S w.r.t. the i -th field of the match vector. On the other hand, the bitslice B^i of S encodes the services that are strictly better matches for the i -th field. Let $A = A^1 \& A^2 \& \dots \& A^{|MV_S|}$, where $\&$ represents the bitwise AND operation. Then, A indicates the services that are equally as good or better in *all* fields of the match vector. Similarly, let $B = B^1 | B^2 | \dots | B^{|MV_S|}$, where $|$ represents the bitwise OR operation. Then, B indicates the services that are strictly better in *at least one* field of the match vector. According to Definition 2.4.5, if a service has its bit set both in A and B , then it dominates S , and, hence, the latter is not in the skyline. On the other hand, if $A \& B$ has no bit set, then S is not dominated by any other service, and thus belongs to the skyline. Table 2.16 illustrates the dominance check for S_4 , which is dominated by S_1 and S_3 . The algorithm is presented in detail in Table 2.17.

Next, we extend the algorithm to provide key aspects of functionality desirable by service requesters and providers.

2.4.2.2 Requester's perspective

The algorithm presented previously identifies the skyline services for a given request. However, this has some limitations; for example, the returned services are not ranked, and they may be too many or too few. For this purpose, in the following we identify and address three additional elements of functionality that may be required by a service requester, referred to as *ranking*, *redefinition*, and *relaxation*.

Ranking. The selected skyline services are determined regardless of specific user preferences; hence, are not ranked. However, in many cases, e.g., when the number of returned results is large, ranking is required. To this end, we present a ranking function that is user preference agnostic and is well aligned with the dominance notion. Intuitively, services that dominate a large number of other services are potentially more interesting and should be examined first.

Definition 2.4.7. (Dominance Set and Score) *Given a set of Semantic Web services \mathcal{S} , a request R , and a service $S \in \mathcal{S}$, the dominance set of S comprises those services dominated by S , i.e., $\mathcal{D}_S = \{S_i \in \mathcal{S} | S \prec^R S_i\}$. The dominance score of S is the cardinality of \mathcal{D}_S , i.e., $ds_S = |\mathcal{D}_S|$.*

The skyline services are ranked based on their dominance score. To calculate this score we utilize the A and B bitmaps for service S . Observe that $\neg A$, where \neg denotes negation, indicates the services that are strictly worse than S in *at least one* field of the match vector. Similarly, $\neg B$ indicates those services being worse or equal to S in *all* fields of the match vector. It is easy to show that if a service has its bit set both in $\neg A$ and $\neg B$, then it is dominated by S . Hence, calculating the dominance score of S resolves to counting the bits set in $(\neg A) \& (\neg B)$.

Redefinition. Suppose that the user would like to redefine his/her request in terms of removing or adding request parameters, either because he/she is not satisfied by the matchmaking, or due to exploratory behaviour. The proposed methodology handles such a scenario efficiently, requiring minimum invocation of the matcher and the fewest changes to the bitmap representation of the match vectors. We distinguish 4 cases and examine the necessary changes to the services selection process.

Adding input parameter. We need to run the match algorithm for the new parameter. Note also that the IN_X field of the match vector might be affected by the matching, and thus, it needs to be re-computed (if the previous value was *fail*). Then, we need to build the bitmap representation for the field corresponding to the new parameter, to update the representation for the IN_X field, if changed, and to execute the bitmap algorithm.

Deleting input parameter. Only the IN_X field of the match vector may be affected by the matching (if it was previously set to *exact*). Therefore, we need to rebuild its bitmap representation. Since the deleted parameter might be needed in a future request, we do not delete the representation corresponding to its match vector field; rather, we modify the bitmap algorithm to skip that field in the calculation of the A, B bitmaps.

Adding output parameter. We need to run the match algorithm for the added output parameter. Then, the bitmap representation for the new parameter must be built and the bitmap algorithm must be executed.

Deleting output parameter. In this case, the match algorithm need not run. We choose to preserve the bitmap representation for the corresponding field and modify the bitmap algorithm to skip that field in the A, B calculation.

Relaxation. Consider the case that the user would like to relax the dominance requirement and retrieve additional relevant services besides those included in the skyline. Such a functionality would prove useful when there are a few very dominant services that *hide* some other potentially interesting offers. For this purpose, we provide the user with the option to examine the next most dominant services, i.e., the next *skylayer*.

Definition 2.4.8. (l -Skylayer Services) Given a set of Semantic Web services \mathcal{S} and a request R , the l -skylayer services of \mathcal{S} w.r.t. R , denoted by $SL_{\mathcal{S}}^R(l)$, is defined recursively as follows: $SL_{\mathcal{S}}^R[1, l] = \bigcup_{0 < k \leq l} SL_{\mathcal{S}}^R(k)$, where $SL_{\mathcal{S}}^R(1)$ is the skyline services $SL_{\mathcal{S}}^R$ and $SL_{\mathcal{S}}^R(l) = SL_{\mathcal{S} \setminus SL_{\mathcal{S}}^R[1, l]}^R$.

Finding the l -skylayer services can be performed by some tweaking of the bitmap algorithm, without invoking the matcher. Assume that the $(l-1)$ -skylayer has been found. We maintain a bitmap mask C that indicates which services belong to one of the previous skylayers, i.e., in $SL_{\mathcal{S}}^R[1, l]$. In the calculation of the A bitmaps for the l -skylayer we need to mask it (i.e., perform bitwise AND operation) with the negation of C , so as to suppress services previously found. Finally, the bitmap mask C is updated by setting the bits of the l -skylayer services.

Algorithm $\text{ModDim}(R, S_j, \mathcal{S})$	
Input:	request R , service S_j , competing services \mathcal{S}
Output:	the dimension d to modify
1	begin
2	$SL_S^R \leftarrow \text{Skyline}(R, \mathcal{S})$
3	$\mathcal{D}_{S_j} \leftarrow \text{DominatedBy}(j, bm)$
4	$SLD \leftarrow SL_S^R \cap \mathcal{D}_{S_j}$
5	$d \leftarrow i \in [1, MV_{S_j}^i]$ s.t. $\max_{S \in SLD} MV_{S_j}^i - MV_S^i $ is minimized
6	return d
7	end

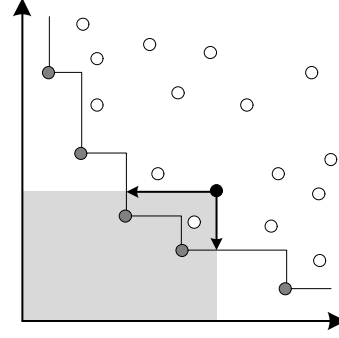


Table 2.18: *Modifying service parameter*

2.4.2.3 Provider’s perspective

Existing works on service discovery focus on locating one or more services that are appropriate for fulfilling the client’s request. In the remaining of this section, we turn our attention towards the provider’s view of the service selection process. From this perspective, a provider would be interested in analyzing the position of his/her services in the market and their potential to attract clients. We consider two scenarios that might be of interest for a service provider.

Service competitiveness. In this scenario, the provider is interested in evaluating how competitive his/her provided service S is with respect to a request R and a set of other available services \mathcal{S} . This can be accomplished by means of two measures: (a) the number of services dominated by S with respect to R ; (b) the number of services dominating S with respect to R . The first is the dominance score of S (see Definition 2.4.7) and is calculated by the function *Dominates*, shown in Table 2.17. The second is provided similarly, through the function *DominatedBy*, also shown in Table 2.17.

Service adaptation. In this scenario, the provider would like to appropriately modify the offered service S in order to target specific user requests, i.e., so that the service would be in the skyline for a considered request R . To keep the required modifications to a minimum, we consider the case where only one parameter is subject to change, and our goal is to determine the parameter for which the required change is minimized. For this purpose, we calculate the services that dominate S and are part of the skyline for the request R . Then, we compare the values in all the dimensions of the selected match vectors, to find the maximum differences in each dimension. The dimension having the minimum among these differences is selected. The intuition lies in the fact that a service is included in the skyline, when it becomes better than all its competitors in at least one dimension. This process is formally described by the algorithm in Table 2.18. As an example, consider the service shown in black in the same figure. The shaded area contains the services that dominate it, including two in the skyline. The arrows represent the maximum differences for each dimension; clearly, the dimension of the shortest arrow corresponds to the minimal change required.

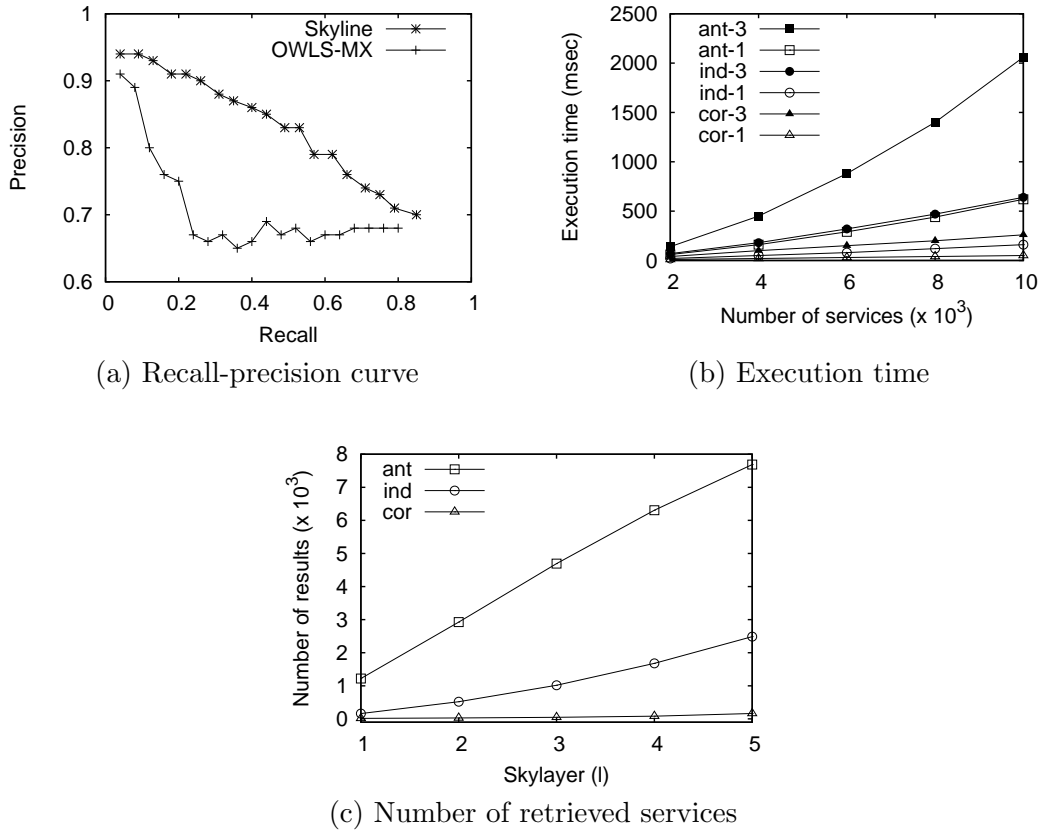


Figure 2.5: *Experimental evaluation on real and synthetic data*

2.4.3 Experimental Evaluation

In the following we present an experimental evaluation of the effectiveness and the efficiency of our skyline-based approach, termed **Skyline**, for selecting the best Semantic Web services with respect to a desirable service description, using both real and synthetic data.

Retrieval Effectiveness. To simulate a real-world scenario, we use the OWL-S service retrieval test collection OWLS-TC v2 [115]. This collection contains services retrieved mainly from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S. More specifically, it comprises: (a) a set of ontologies, derived from 7 different domains (education, medical care, food, travel, communication, economy and weapons), used to semantically annotate the service parameters, (b) a set of 576 OWL-S services, (c) a set of 28 sample requests, and (d) the relevance set for each request (manually identified).

To better gauge the performance of our approach we consider the matchmaking algorithm of [84], termed **OWLS-MX**. **OWLS-MX** is a hybrid matchmaker, which, apart from logic-based match, supports different IR similarity metrics for content-based retrieval (e.g., cosine similarity or extended Jacquard similarity). Since the focus of this paper is on semantic matching, we only consider the logic-based filters while running **OWLS-MX**. It is important to note, however, that our approach is generic in that it can straightforwardly consider other types of filters, simply by terms of increasing the size of the match vectors to accommodate for the additional parameters; hence, the process of selecting the best matches does not change.

Method	MAP	P@1	P@2	P@3	P@5	P@10
Skyline	0.83	0.94	0.93	0.91	0.87	0.76
OWLS-MX	0.71	0.91	0.79	0.75	0.67	0.67

Table 2.19: *Evaluation of retrieval accuracy*

For each request, we calculate the match vectors and apply **Skyline** to select the best candidates. Since the number of services in the skyline may vary, successive skylayers are computed until an adequate number of services has been retrieved (see the *relaxation* case in Section 2.4.2.2). Thus, to each obtained service S that belongs to the l -skylayer and has dominance score ds_S , we assign the tuple $\langle l, ds_S \rangle$. As discussed in Section 2.4.2, we consider better matches the services that belong to the lowest l skylayer, and among those that belong to the same skylayer we consider better matches the ones with higher dominance score ds_S , i.e., we rank the obtained services by l , solving ties using ds_S .

Similarly, for each request the logic-based filters of **OWLS-MX** are applied to all services. **OWLS-MX** assigns to each service a score based on the worst degree of match among all parameters. Finally, the services are ranked according to their score, i.e., the best match is a service that has *exact* match on all parameters.

We apply well-established IR metrics to measure the performance of the two methods, w.r.t. the corresponding relevance sets [16]. In particular, Figure 2.5(a) depicts the micro-averaged recall-precision curves for all the queries in the test collection. It is clear that **Skyline** outperforms **OWLS-MX** in terms of precision at all recall levels, as well as achieving a higher final recall. The results for the measures (a) Mean Average Precision (MAP), where average precision refers to the average of the precision after each relevant service retrieved, and (b) precision at N are detailed in Table 2.19.

These measures emphasize on returning relevant results earlier, which is important as users often tend to examine only the first few results retrieved. In particular, P@1 is especially important, as it determines the success in fully automated service discovery scenarios, where no human user is involved in the process, and thus the top-1 result is selected. Again, **Skyline** outperforms **OWLS-MX** in all cases.

Synthetic data. We measure the performance overhead associated with our approach for computing the skyline services. The algorithm was implemented in Java and the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux. The reported measurements refer only to the process of computing the skyline, and do not include the time to perform the logic-based match for each parameter. The later depends on factors which are outside the scope of this paper, e.g., the size and type of ontologies used or the performance of the employed reasoner.

We construct match vectors of 6 parameters and assign to each degrees of match under three distributions: in **independent** (**ind**), degrees of match are assigned independently to each parameter; in **correlated** (**cor**), the values in the match vector are positively correlated, i.e., a good match in some service parameters increases the possibility of a good match in the others; in **anti-correlated** (**ant**) the values are negatively correlated, i.e., good matches (or bad matches) in *all* parameters are less likely to occur.

Figure 2.5(b) illustrates the running time, in milliseconds, for determining the

services that belong to the l -skylayer, for $l = 1$ (i.e., the skyline), and $l = 3$, for the three types of distributions, while varying the number of services from 2K up to 10K. Observe that the time required is higher (lower) for anti-correlated (correlated) data, as the number of skyline services in this case is also higher (lower). Still, it does not exceed roughly 0.5 seconds for all cases, except that of 3 skylayers of anti-correlated data, where it takes roughly 2 seconds. Notice, however, that since for the anti-correlated case the number of services contained in the first skylayer is already quite large, computing additional layers is normally not required.

Figure 2.5(c) illustrates, for each distribution, the number of services retrieved by the first l -skylayers, for $l = 1$ to 5, and for an initial set of 10K services. As shown, the correlation of the degrees of match directly affects the number of selected services for each layer. For instance, the skyline comprises 16, 162, and 1221 services for the correlated, independent, and anti-correlated case respectively. These results prove the necessity of the extensions proposed in Section 2.4.2.2 for *ranking* and *relaxation*.

2.5 Ranking of Services under Multiple Criteria

2.5.1 Motivation and Problem Definition

In the previous sections we have addressed the problem of service discovery and selection, assuming that a single matching function is used to compare user requests against offered services. However, the matching process may accommodate a variety of methods, ranging from keyword-based match to logic-based match. Even though hybrid service matchmakers have been proposed (see Section 2.2 for more details), the different matching criteria supported are typically provided to the user as alternatives; that is, the final match is computed using eventually one of the available criteria, instead of applying all of them and combining the partial results. On the contrary, in this section we propose a method for service matchmaking and ranking under multiple matching criteria. We begin with a motivating example, a discussion of the requirements, and the definition of three ranking criteria that are applicable to our setting. In particular, we introduce our notion of *top- k dominant Web services*, and we justify our formulation by discussing some related notions, namely *p -skyline* [125] and *K -skyband* [120], and showing that these concepts are inadequate in capturing the requirements of the problem at hand. Then, we give corresponding algorithms for service selection (Section 2.5.2), and an experimental evaluation of our method (Section 2.5.3).

Example. Consider a user searching for a Web service providing weather information for a specific location. For simplicity, we assume only one input and one output parameter. There are four available Web services, www.worldweather.org (A), www.weather.gov (B), www.weather.com (C), and www.webservicex.net (D). Furthermore, three different matching filters (e.g., different string similarity measures), have been applied, resulting in the degrees of match shown in Figure 2.6. Observe that under any criterion, service A constitutes a better match with respect to both parameters, than any other service. However, there is no clear winner among the other three services. For instance, consider services B and D. If the first matching criterion is the one that more closely reflects the actual relevance of B to the given request, then B is definitely a better match than D. On the other hand, if the second measure is chosen, then B has a lower match degree for the input parameter

Service	Parameter	fm_1	fm_2	fm_3
www.worldweather.org	Country	0.96	1.00	0.92
	WeatherForecast	0.92	0.96	1.00
www.weather.gov	State	0.80	0.60	0.64
	7DayForecast	0.80	0.88	0.72
www.weather.com	ZipCode	0.84	0.88	0.72
	LocalWeather	0.84	0.64	0.60
www.webservicex.net	City	0.76	0.68	0.56
	Weather	0.76	0.64	0.68

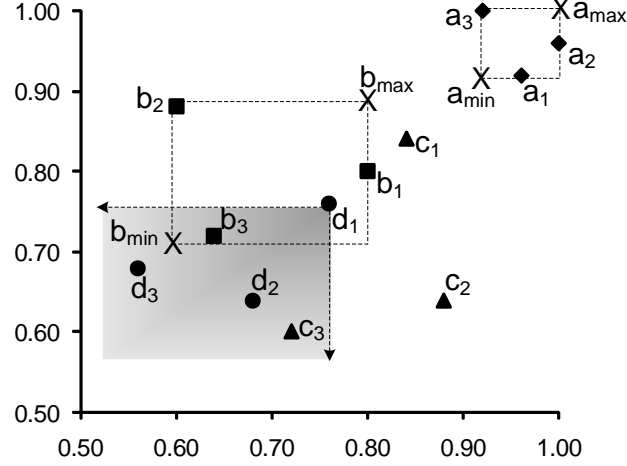


Figure 2.6: An illustrative example

but a higher degree for the output. Even for such a simple scenario, specifying an appropriate ranking for the candidate matches is not straightforward. \square

Based on the above, we can identify the following main requirements for Web services search:

- (\mathcal{R}_1) how to combine the degrees of match for the different parameters in the matched descriptions;
- (\mathcal{R}_2) how to combine the match results from the individual similarity measures; and
- (\mathcal{R}_3) how to rank the results.

To abstract away from a particular Web service representation, we model a Web service operation as a function that receives a number of inputs and returns a number of outputs. Other types of parameters, such as pre-conditions and effects or QoS parameters, can be handled similarly. Hence, in the following, the description of a Web service operation corresponds to a vector S containing its I/O parameters. A request R is viewed as the description of a desired service operation, and is therefore represented in the same way.

Given a Web service request, the search engine matches registered services against the desired description. For this purpose, it uses a similarity measure f_m to assess the similarity between the parameters in these descriptions. If more than one offered parameters match a requested parameter, the closest match is considered. Thus, the result of matching a pair $\langle R, S \rangle$ is specified by a vector $U_{R,S}$, such that

$$\forall i \in [0, |R|] \quad U_{R,S}[i] = \max_{j=0}^{|S|} f_m(R[i], S[j]) \quad (2.14)$$

Employing more than one matching criteria means that for each different similarity measure f_{m_i} , a match vector $U_{R,S}^{m_i}$ is produced. Hereafter, we refer to each such individual vector as *match instance*, denoted by lowercase letters (e.g., u, v), whereas to the set of such vectors for a specific pair $\langle R, S \rangle$ as *match object*, denoted by uppercase letters (e.g., U, V).

To address the aforementioned challenges \mathcal{R}_1 , \mathcal{R}_2 , and \mathcal{R}_3 , we propose an approach based on the notion of *dominance* (see Section 2.4.1 for more details). Assume a set of match instances \mathcal{I} in a d -dimensional space. Given two instances $u, v \in \mathcal{I}$, we say that u dominates v , denoted by $u \succ v$, iff u is better than or equal to v in all dimensions, and strictly better in at least one dimension, i.e.

$$u \succ v \Leftrightarrow \forall i \in [0, d) u[i] \geq v[i] \wedge \exists j \in [0, d) u[j] > v[j] \quad (2.15)$$

If u is neither dominated by nor dominates v , then u and v are incomparable.

Example (Cont'd). Consider the match instances depicted in the 2-dimensional space in Figure 2.6. We can observe, for example, that the instance d_1 dominates the instances b_3 and c_3 . Similarly, it is dominated by the instances b_1 and c_1 , as well as by all the instances of a , but it neither dominates nor is dominated by b_2 and c_2 . \square

Using the notion of dominance, allows us to deal with the requirement (\mathcal{R}_1), since comparing matched services takes into consideration the degrees of match in all parameters, instead of calculating and using a single, overall score.

To address the requirement of multiple matching criteria (\mathcal{R}_2), which results in a set of match instances per service, we use a model that is similar to the probabilistic skyline model proposed in [125]. The dominance relationship between two instances u and v is defined as previously. Then, the dominance relationship between two objects U and V is determined by comparing each instance $u \in U$ to each instance $v \in V$. This may result in a partial dominance of U by V , in other words a probability under which U is dominated by V . Note that, without loss of generality, all the instances of an object are considered of equal probability, i.e., all the different matching criteria employed are considered of equal importance; it is straightforward to extend the approach to the case that different weights are assigned to each matching criterion. Based on this notion, the work in [125] defines the concept of *p-skyline*, which comprises all the objects belonging to the skyline with probability at least p .

Although the assumption of multiple instances per object satisfies \mathcal{R}_2 , the requirement \mathcal{R}_3 is not fulfilled by the concept of *p-skyline*. The notion of skyline, and consequently that of *p-skyline*, is too restrictive: only objects not dominated by any other object are returned. However, for Web services retrieval, given that the similarity measures provide only an indication of the actual relevance of the considered service to the given request, interesting services may be missed.

A possible work-around would be to consider a *K-skyband* query, which is a relaxed variation of a skyline query, returning those objects that are dominated by at most K other objects. In particular, we could extend the *p-skyline* to the *p-K-skyband*, comprising those objects that are dominated by at most K other objects with probability at least p . Relaxing (restricting) the value of K , increases (reduces) the number of results to be returned. Still, such an approach faces two serious drawbacks. The first is how to determine the right values for the parameters p and

K . A typical user may specify the number of results that he/she would like to be returned (e.g., top 10), but he/she cannot be expected to understand the semantics or tune such parameters neither it is possible to determine automatically the values of p and K from the number of desired results. Second, the required computational cost is prohibitive. Indeed, in contrast to the p -skyline, where, for each object, only one case needs to be tested (i.e., the case that this object is not dominated by any other object), the p - K -skyband requires to consider, for each object, the cases that it is dominated by exactly 0, 1, 2, \dots , K other objects, i.e., a number of $\sum_{j=0}^K \frac{N!}{j!(N-j)!}$ cases, where N is the total number of matches.

In the following, we formulate three ranking criteria that meet the requirements \mathcal{R}_1 , \mathcal{R}_2 , and \mathcal{R}_3 , without facing the aforementioned limitations. The first two are based, respectively, on the following intuitions: (a) a match is good if it is *dominated* by as *few* other matches as possible, and (b) a match is good if it *dominates* as *many* other matches as possible; the third is a *combination* of both.

Dominated Score. Given an instance u , we define the dominated score of u , denoted by dds , as:

$$u.dds = \sum_{v \neq u} \frac{|\{v \in V \mid v \succ u\}|}{|V|} \quad (2.16)$$

Hence, the dominated score of u considers the instances that dominate u . Then, the dominated score of an object U is defined as the (possibly weighted) average of the dominated scores of its instances:

$$U.dds = \sum_{u \in U} \frac{u.dds}{|U|} \quad (2.17)$$

The dominated score of an object indicates the average number of objects that dominate it. Hence, a *lower* dominated score indicates a better match.

Dominating Score. Given an instance u , we define the dominating score of u , denoted by dgs , as:

$$u.dgs = \sum_{v \neq u} \frac{|\{v \in V \mid u \succ v\}|}{|V|} \quad (2.18)$$

Hence, the dominating score of u considers the instances that u dominates. Then, the dominating score of an object U is defined as the (possibly weighted) average of the dominating scores of its instances:

$$U.dgs = \sum_{u \in U} \frac{u.dgs}{|U|} \quad (2.19)$$

The dominating score of an object indicates the average number of objects that it dominates. Hence, a *higher* dominating score indicates a better match.

Dominance Score. Given an instance u , we define the dominance score of u , denoted by ds , as:

$$u.ds = u.dgs - \lambda \cdot u.dds \quad (2.20)$$

The dominance score of u promotes u for each instance it dominates, while penalizing it for each instance that dominates it. The parameter λ is a scaling factor explained in the following. Consider an instance u corresponding to a good

match. Then, it is expected that u will dominate a large number of other instances, while there will be only few instances dominating u . In other words, the dgs and dds scores of u will differ, typically, by orders of magnitude. Hence, the factor λ scales dds so that it becomes sufficient to affect the ranking obtained by dgs . Consequently, the value of λ depends on the size of the data set and the distribution of the data. An effective heuristic for selecting the value for λ is $\Delta dgs/\Delta dds$, where Δdgs and Δdds are the differences in the scores of the first and second result obtained by each respective criterion (see Section 2.5.3.1 for more details).

In addition, the dominance score of an object U is defined as the (possibly weighted) average of the dominance scores of its instances:

$$U.ds = \sum_{u \in U} \frac{u.ds}{|U|} \quad (2.21)$$

Example (Cont'd). Consider the case of the object C with instances c_1 , c_2 and c_3 , as shown in Figure 2.6. The instance c_1 is dominated by the instances a_1 , a_2 and a_3 , whereas it dominates b_1 , b_3 , d_1 , d_2 and d_3 . Thus, its scores are: $dds = 1$, $dgs = 5/3$ and $ds = 2/3$ (for $\lambda = 1$). \square

We can now provide the formal definition for the top- k dominant Web services selection problem.

Problem Statement. *Given a Web service request R , a set of available Web services \mathcal{S} , and a set of similarity measures \mathcal{F}_m , return the top- k matches, according to the aforementioned ranking criteria.*

2.5.2 Algorithms

We first introduce some important observations pertaining to the problem at hand. The algorithms for selecting the top- k services according to the criteria dds , dgs and ds are then presented in Sections 2.5.2.1, 2.5.2.2 and 2.5.2.3, respectively.

A straightforward algorithm for calculating the dominated (resp., dominating) score is the following. For each instance u of object U iterate over the instances of all other objects and increase a counter associated with U , if u dominates (resp., is dominated by) the instance examined. Then, to produce the top- k list of services, simply sort them according to the score in the counter. However, the applicability of this approach is limited by its large computation cost, which does not depend on k . Observe that no matter the value of k , it exhaustively performs all possible dominance checks among instances.

On the other hand, our algorithms address this issue by establishing lower and upper bounds for the dominated/dominating scores. This essentially allows us to (dis-)qualify objects to or from the results set, without computing their exact score. Let U be the current k -th object. For another object V to qualify for the result set, the score of V , as determined by its bounds, should be at least as good (i.e., lower, for dds , or higher, for dgs) as that of U . In the following, we delve into some useful properties of the dominance relationship (see Equation 2.15), in order to prune the search space.

Observe that the dominance relationship is transitive, i.e., given three instances u , v and w , if $u \succ v$ and $v \succ w$, then $u \succ w$. An important consequence for obtaining upper and lower bounds is the following.

Property 1. *If $u \succ v$, then v is dominated by at least as many instances as u , i.e., $v.dds \geq u.dds$, and it dominates at most as many instances as u , i.e., $v.dgs \leq u.dgs$.*

Presorting the instances according to a monotone function, e.g., $F(u) = \sum_i u[i]$, can help reduce unnecessary checks.

Property 2. *Let $F(u)$ be a function that is monotone in all dimensions. If $u \succ v$, then $F(u) > F(v)$.*

To exploit this property, we place the instances in a list sorted in descending order of the sum of their values. Then, given an instance u , searching for instances by which u is dominated (resp., it dominates) can be limited to the part of the list before (resp., after) u . Furthermore, if $F(u)$ is also symmetric in its dimensions [18], e.g., $F(u) = \sum_i u[i]$, the following property holds, providing a termination condition.

Property 3. *Let $F(u)$ be a function that is monotone and symmetric in all dimensions. If $\min_i u[i] \geq F(v)$ for two instances u and v , then u dominates v as well as all instances with $F()$ value smaller than v 's.*

Given an object U , let u_{min} be a virtual instance of U whose value in each dimension is the minimum of the values of the actual instances of U in that dimension, i.e., $u_{min}[i] = \min_{j=0}^{|U|} u_j[i]$, $\forall i \in [0, d)$. Similarly, let $u_{max}[i] = \max_{j=0}^{|U|} u_j[i]$, $\forall i \in [0, d)$. Viewed in a $2-d$ space, these virtual instances, u_{min} and u_{max} , form, respectively, the lower-left and the upper-right corners of a virtual minimum bounding box containing the actual instances of U (see Figure 2.6). The following property holds.

Property 4. *For each instance $u \in U$, it holds that $u_{max} \succeq u$, and $u \succeq u_{min}$.*

Combined with the transitivity of the dominance relationship, this allows us to avoid an exhaustive pairwise comparison of all the instances of two objects, by first comparing their corresponding minimum and maximum virtual instances. More specifically, given two objects U and V , (a) if u_{min} dominates v_{max} , then all the instances of U dominate all the instances of V , i.e., $u_{min} \succ v_{max} \Rightarrow u \succ v \forall u \in U, v \in V$; (b) if u_{min} dominates an instance of V , then all the instances of U dominate this instance of V , i.e., $u_{min} \succ v \Rightarrow u \succ v \forall u \in U$; (c) if an instance of U dominates v_{max} , then this instance of U dominates all the instances of V , i.e., $u \succ v_{max} \Rightarrow u \succ v \forall v \in V$.

2.5.2.1 Ranking by dominated score

The first algorithm, hereafter referred to as *TKDD*, computes top- k Web services according to the dominated score criterion, *dds*. The goal is to quickly find, for each object, other objects dominating it, avoiding an exhaustive comparison of each instance to all other instances.

The algorithm maintains three list, \mathcal{I}_{min} , \mathcal{I}_{max} , and \mathcal{I} , containing, respectively, the minimum bounding instances, the maximum bounding instances, and the actual instances of the objects. The instances inside these lists are sorted by $F(u) = \sum_i u[i]$

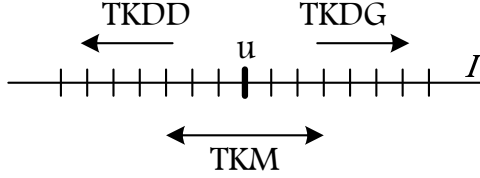


Figure 2.7: Search space for $TKDD$, $TKDG$, and TKM

and are examined in descending order. The results are maintained in a list \mathcal{R} sorted in ascending order of dds . The algorithm uses two variables, $ddsMax$ and $minValue$, which correspond to an upper bound for dds , and to the minimum value of the current k -th object, respectively.

Given that, for an object U , we are interested in objects that *dominate* it, we search only for instances that are prior to those of U in \mathcal{I} (see Figure 2.7). Since, the top matches are expected to appear in the beginning of \mathcal{I} , this significantly reduces the search space. The basic idea is to use the bounding boxes of the objects to avoid as many dominance checks between individual instances as possible. After k results have been acquired, we use the score of the k -th object as a maximum threshold. Objects whose score exceeds the threshold are pruned. In addition, if at some point, it is guaranteed that the score of all the remaining objects exceeds the threshold, the search terminates.

More specifically, the algorithm, shown in Table 2.20, proceeds in the following six steps.

Step 1. *Initializations (lines 2–7).* The result set \mathcal{R} and the variables $ddsMax$ and $minValue$ are initialized. The lists \mathcal{I}_{min} , \mathcal{I}_{max} , and \mathcal{I} are initialized, and sorted by $F(u)$. Then the algorithm iterates over the objects, according to their maximum bounding instance.

Step 2. *Termination condition (line 10).* If the $F()$ value of the current u_{max} does not exceed the minimum value of the current k -th object, the result set \mathcal{R} is returned and the algorithm terminates (see Property 3).

Step 3. *Dominance check object-to-object (lines 12–17).* For the current object U , the algorithm first searches for objects that fully dominate it. For example, in the case of the data set of Figure 2.6, with a single dominance check between b_{max} and a_{min} , we can conclude that all the instances b_1 , b_2 and b_3 are dominated by a_1 , a_2 and a_3 . According to property 2, only objects with $F(v_{min}) > F(u_{max})$ need to be checked. If a v_{min} is found to dominate u_{max} , then the score of U is increased by 1, and the sum of the new score and the score of V (see Property 1) is compared to the current threshold, $ddsMax$. If it exceeds the threshold, the object is pruned and the iteration continues with the next object. In this case, the score of the object is propagated to its instances for later use. Otherwise, the score of the object is reset, to avoid duplicates, and the search continues in the next step.

Step 4. *Dominance check object-to-instance (lines 19–24).* This step searches for individual instances v that dominate U . For example, in Figure 2.6, a dominance check between d_{max} (which coincides with d_1) and c_1 shows that all the instances d_1 , d_2 , and d_3 are dominated by c_1 . As before, only instances with $F(v) > F(u_{max})$ are considered. If an instance v is found to dominate u_{max} , then the score of U is

Algorithm <i>TKDD</i>	
	Input: A set of objects \mathcal{U} , each comprising M instances; The number k of results to return.
	Output: The top- k objects w.r.t. dds in a sorted set \mathcal{R} .
1	begin
2	Initialize $\mathcal{R} = \emptyset$; $ddsMax = \infty$; $minValue = -1$;
3	for $U \in \mathcal{U}$ do
4	$(u_{min}, u_{max}) \leftarrow$ calculate min and max bounding instances ;
5	$\mathcal{I}_{min} \leftarrow$ insert u_{min} ordered by $F(u_{min})$ desc. ;
6	$\mathcal{I}_{max} \leftarrow$ insert u_{max} ordered by $F(u_{max})$ desc. ;
7	for $u \in U$ do $\mathcal{I} \leftarrow$ insert u ordered by $F(u)$ desc. ;
8	for $u_{max} \in \mathcal{I}_{max}$ do
9	if $ \mathcal{R} = k$ then
10	if $F(u_{max}) \leq minValue$ then return \mathcal{R} ;
11	$U.dds = 0$;
12	for $v_{min} \in \mathcal{I}_{min}^{F(u_{max})}$ do
13	if $v_{min} \succ u_{max}$ then
14	$U.dds = U.dds + 1$;
15	if $(U.dds + V.dds) \geq ddsMax$ then
16	for $u \in U$ do $u.dds = U.dds$;
17	skip U ;
18	$U.dds = 0$;
19	for $v \in \mathcal{I}^{F(u_{max})}$ do
20	if $v \succ u_{max}$ then
21	$U.dds = v.dds + 1/M$;
22	if $(U.dds + v.dds) \geq ddsMax$ then
23	for $u \in U$ do $u.dds = U.dds$;
24	skip U ;
25	$U.dds = 0$;
26	for $u \in U$ do
27	for $v_{min} \in \mathcal{I}_{min}^F(u)$ do
28	if $v_{min} \succ u$ then
29	$u.dds = u.dds + 1/M$;
30	if $(U.dds + u.dds + V.dds) \geq ddsMax$ then
31	$U.dds = U.dds + u.dds + V.dds$;
32	skip U ;
33	$U.dds = U.dds + u.dds + V.dds$;
34	$U.dds = 0$;
35	for $u \in U$ do $u.dds = 0$;
36	for $u \in U$ do
37	for $v \in \mathcal{I}^F(u)$ do
38	if $v \succ u$ then
39	$u.dds = u.dds + 1/M^2$;
40	if $(U.dds + u.dds + v.dds) \geq ddsMax$ then
41	$U.dds = U.dds + u.dds + v.dds$;
42	skip U ;
43	$U.dds = U.dds + u.dds + v.dds$;
44	if $ \mathcal{R} = k$ then remove the last result from \mathcal{R} ;
45	$\mathcal{R} \leftarrow$ insert U ordered by dds asc.
46	if $ \mathcal{R} = k$ then
47	$U_k \leftarrow$ the k -th object in \mathcal{R} ;
48	$ddsMax = U_k.dds$;
49	$minValue = \min_{i=1}^M (U_{k_{min}}[i])$;
50	return \mathcal{R} ;
51	end

Table 2.20: Algorithm *TKDD*

Algorithm <i>TKDG</i>	
Input:	A list \mathcal{I} containing all the instances u , in descending order of $F(u)$; The number k of results to return.
Output:	The top- k objects w.r.t. dds in a sorted set \mathcal{R} .
1 begin	
2	Initialize $\mathcal{R} = \emptyset, \mathcal{L} = \emptyset$;
3	$\mathcal{U} \leftarrow$ the set of objects in descending order of $F(u_{max})$;
4	for every object $U \in \mathcal{U}$ do
5	if ($\frac{ \mathcal{I} - pos(u_{max})}{M} < R_{k-1}.dgs^-$) then return \mathcal{R} ;
6	if ($ \mathcal{R} = 0$) then add U in \mathcal{R} ;
7	if ($\exists V \in \mathcal{L} \cup \mathcal{R}_{k-1}$ s.t. V fully dominates U) then skip U ;
8	set $U.dgs^- = 0, U.dgs^+ = \sum_{u \in \mathcal{U}} \frac{ \mathcal{I} - pos(u)}{M^2}, U_i = pos(u_{max})$;
9	for $j = \mathcal{R} - 1$ to 0 do
10	while (not ($U.dgs^+ < R_j.dgs^-$ or $U.dgs^- > R_j.dgs^+$)) do
11	\lfloor refineBounds (U, R_j);
12	if ($U.dgs^+ < R_j.dgs^-$) then
13	if ($j = k - 1$) then add U in \mathcal{L} , and continue with the next object;
14	else move R_{k-1} to \mathcal{L} , add U in \mathcal{R} after R_j , and continue with the next object;
15	\lfloor move R_{k-1} to \mathcal{L} , and add U at the beginning of \mathcal{R} ;
16	return \mathcal{R} ;
17 end	

Table 2.21: Algorithm *TKDG*

increased by $1/M$, where M is the number of instances per object, and the sum of the new score and that of v is compared to the current threshold, $ddsMax$.

Step 5. *Dominance check instance-to-object (lines 26–33).* If the object U has not been pruned in the previous two steps, its individual instances are considered. Each instance u is compared to instances v_{min} , with $F(v_{min}) > F(u)$. If it is dominated, the score of u is again increased by $1/M$, and the threshold is checked. In Figure 2.6, this is the case with d_3 and b_{min} .

Step 6. *Dominance check instance-to-instance (lines 35–42).* If all previous steps failed to prune the object, a comparison between individual instances takes place where each successful dominance check contributes to the object’s score by $1/M^2$.

Step 7. *Result set update (lines 44–49).* If U has not been pruned in any of the previous steps, it is inserted in the result set \mathcal{R} . If k results exist, the last is removed. After inserting the new object, if the size of \mathcal{R} is k , the thresholds $ddsMax$ and $minValue$ are set accordingly.

2.5.2.2 Ranking by dominating score

The *TKDG* algorithm, shown in Table 2.21, computes the top- k dominant Web services with respect to the dominating score, i.e., it retrieves the k match objects that dominate the larger number of other objects. This is a more challenging task compared to that of *TKDD*, for the following reason. Let $pos(u)$ denote the position of the currently considered instance u in the sorted, decreasing by F , list \mathcal{I} of instances. To calculate $u.dds$, *TKDD* performs in the worst case $pos(u)$ dominance checks, i.e., with those before u in the list. On the other hand to calculate $u.dgs$, *TKDG* must perform in the worst case $|\mathcal{I}| - pos(u)$ checks, i.e., those after u (see Figure 2.7). Since the most dominating and less dominated objects are located

close to the beginning of \mathcal{I} , execution will terminate when $pos(u)$ is small relative to $|\mathcal{I}|$. As a result, the search space for \mathcal{TKDG} is significantly larger than \mathcal{TKDD} 's. Furthermore, \mathcal{TKDD} allows for efficient pruning as it searches among objects and/or instances that have already been examined in a previous iteration, and therefore (the bounds of) their scores are known.

The \mathcal{TKDG} algorithm maintains three structures: (1) the \mathcal{I} list; (2) a list \mathcal{R} of at most k objects (current results), ordered by dominating score descending; (3) a list \mathcal{L} containing objects that have been disqualified from \mathcal{R} , used to prune other objects. The lists \mathcal{R} and \mathcal{L} are initially empty.

Similar to \mathcal{TKDD} , the algorithm iterates over the objects, in descending order of their maximum bounding instance (*lines 3–4*). Let U be the currently examined object. U can dominate at most $|\mathcal{I}| - pos(u_{max})$ instances. If this amount, divided by the number of instances per object, is lower than T , where T is the lower bound for the score of the k -th object in \mathcal{R} , the whole process terminates, and the result set \mathcal{R} is returned (*line 5*). On the other hand, if the result set is empty, then U is added as the first result (*line 6*).

Next, if U is dominated by the k -th object in \mathcal{R} or by any object in \mathcal{L} , it is pruned (*line 7*). Otherwise, we need to check whether U qualifies for \mathcal{R} . For an examined object U it is straightforward to calculate its dominating score, by examining all the instances in \mathcal{I} , starting from the position of its best instance. However, we avoid unnecessary computations by following a lazy approach, which examines instances in \mathcal{I} until a position that is sufficient to qualify (disqualify) U for (from) the current result set \mathcal{R} . For this purpose, we maintain for each examined object U a lower and an upper bound for its dominating score, $U.dgs^-$ and $U.dgs^+$ respectively, as well as the last examined position in \mathcal{I} , denoted by U_i . We initialize the lower and upper bounds for the dominating score of U to

$$U.dgs^- = 0 \text{ and } U.dgs^+ = \sum_{u \in U} \frac{|\mathcal{I}| - pos(u)}{M^2},$$

respectively. Also, the last examined position for U is initialized to $U_i = pos(u_{max})$ (*line 8*).

Let V be the k -th result in \mathcal{R} . We start by comparing U with V . Three cases may occur: (1) if $U.dgs^+ < V.dgs^-$, then U does not qualify for \mathcal{R} , and it is inserted in \mathcal{L} ; (2) if $U.dgs^- > V.dgs^+$, U is inserted in \mathcal{R} before V , and it is recursively compared to the preceding elements of V in \mathcal{R} ; if V was the k -th object in \mathcal{R} , it is removed from \mathcal{R} and it is inserted in \mathcal{L} ; (3) otherwise, the lower and upper bounds of U and V need to be refined, until one of the conditions (1) or (2) is satisfied. This refinement is performed by searching in \mathcal{I} for instances dominated by an instance of U , starting from the position U_i . At each step of this search, the instance at this position, v , is compared to the instances of U preceding it. For each instance u of U that dominates (does not dominate) v , the lower (upper) bound of the dominating score of U is increased (decreased) by $1/M^2$. Also, the last examined position for U is incremented by 1. Notice that, as in \mathcal{TKDD} , if $F(v)$ does not exceed the minimum value of u , then u dominates v and all its subsequent instances, hence, the lower bound of the score of u is updated accordingly, without performing dominance checks with those instances (*lines 9–15*).

2.5.2.3 Ranking by dominance score

The previously presented algorithms take into consideration either one of the *dds* or *dgs* scores. In the following, we present an algorithm, referred to as \mathcal{TKM} , that computes the top- k matches with respect to the third criterion introduced in Section 2.5.1, which combines both measures. In particular, this algorithm is derived by the algorithm \mathcal{TKDG} , with an appropriate modification to account also for the dominated score. More specifically, this modification concerns the computation of the lower and upper bounds of the scores. First, the lower bound for the score of an object is now initialized as:

$$U.dgs^- = -\lambda \cdot \sum_{u \in U} \frac{pos(u)}{M^2} \quad (2.22)$$

instead of 0. Second, the bounds refinement process now needs to consider two searches, one for instances dominated by the current object, and one for instances that dominate the current object (see Figure 2.7). These searches proceed interchangeably, and the bounds are updated accordingly. Consequently, two separate cursors need to be maintained for each object, to keep track of the progress of each search in the list containing the instances.

2.5.3 Experimental Evaluation

In this section we present an extensive experimental study of our approach. In particular, we conduct two sets of experiments. First, we investigate the benefits resulting from the use of the proposed ranking criteria with respect to the recall and precision of the computed results. For this purpose, we rely on a publicly available, well-known benchmark for Web service discovery, comprising real-world service descriptions, sample requests, and relevance sets. In particular, the use of the latter, which are manually identified, allows to compare the results of our methods against human judgement. In the second set of experiments, we consider the computational cost of the proposed algorithms under different combinations of values for the parameters involved, using synthetic data sets.

Our approach has been implemented in Java and all the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux.

2.5.3.1 Retrieval Effectiveness

To evaluate the quality of the results returned by the three proposed criteria, we have used the publicly available service retrieval test collection OWLS-TC v2 [115]. This collection contains real-world Web service descriptions, retrieved mainly from public IBM UDDI registries. More specifically, it comprises: (a) 576 service descriptions, (b) 28 sample requests, and (c) a manually identified relevance set for each request.

Our prototype comprises two basic components: (a) a matchmaker, based on the OWLS-MX service matchmaker [84], and (b) a component implementing the algorithms presented in Section 2.5.2 for processing the degrees of match computed by the various matching criteria and determining the final ranking of the retrieved services.

OWLS-MX matches I/O parameters extracted from the service descriptions, exploiting either purely logic-based reasoning (M0) or combined with some content-

based, IR similarity measure. In particular, the following measures are considered: loss-of-information measure (M1), extended Jaccard similarity coefficient (M2), cosine similarity (M3), and Jensen-Shannon information divergence based similarity (M4). Given a request R and a similarity measure (M0–M4), the degrees of match among its parameters and those of a service S are calculated and then aggregated to produce the relevance score of S . Therefore, given a request, a ranked list of services is computed for each similarity criterion. Note that in OWLS-MX no attempt to combine rankings from different measures is made.

We have adapted the matching engine of OWLS-MX as follows. For a pair $\langle R, S \rangle$, instead of a single aggregated relevance score, we retrieve a score vector containing the degrees of match for each parameter. Furthermore, for any such pair, all similarity criteria (M0–M4) are applied, resulting in five score vectors. Hence, for a request having in total d I/O parameters, each matched service corresponds essentially to an object, and the score vectors correspond to the object’s d -dimensional instances. Then, the algorithms \mathcal{TKDD} , \mathcal{TKDG} , and \mathcal{TKM} , described in Section 2.5.2, are applied to determine the ranked list of services for each criterion.

To evaluate the quality of the results, we apply the following standard IR evaluation measures [16]:

- *Interpolated Recall-Precision Averages*: measures precision, i.e., percent of retrieved items that are relevant, at various recall levels, i.e., after a certain percentage of all the relevant items have been retrieved.
- *Mean Average Precision (MAP)*: average of precision values calculated after each relevant item is retrieved.
- *R-Precision (R-prec)*: measures precision after all relevant items have been retrieved.
- *bpref*: measures the number of times judged non-relevant items are retrieved before relevant ones.
- *Reciprocal Rank (R-rank)*: measures (the inverse of) the rank of the top relevant item.
- *Precision at N (P@N)*: measures the precision after N items have been retrieved.

The conducted evaluation comprises three stages.

First, we compare the three different ranking criteria considered in our approach. The resulting recall-precision graphs are depicted in Figure 2.8(a). Regarding \mathcal{TKM} , we study the effect of the parameter λ (see Section 2.5.1), considering 4 variations, denoted as $\mathcal{TKM}-\lambda$, for $\lambda=1, 5, 20, 50$. As shown in Figure 2.8(a), for a recall level up to 30%, the performance of all methods is practically the same. Differences start to become more noticeable after a recall level of around 60%, where the precision of \mathcal{TKDG} starts to degrade at a considerably higher rate compared to that of \mathcal{TKDD} . This means that several services, even though dominating a large number of other matches, were not identified as relevant in the provided relevance sets. On the other hand, as expected, the behavior of \mathcal{TKM} is dependent on the value of λ . Without considering any scaling factor, i.e., for $\lambda=1$, the effect of the dds

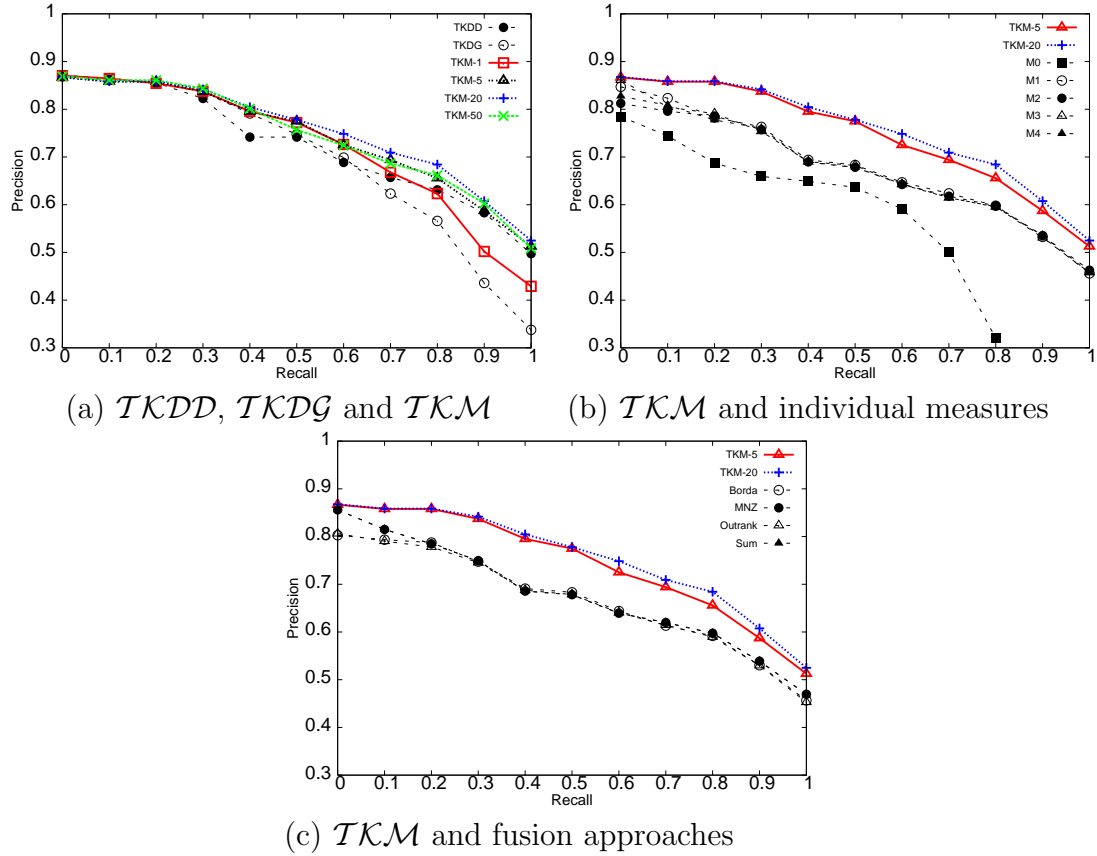


Figure 2.8: Recall-Precision graphs

criterion is low, and, hence, although TKM performs better than $TKDG$, it still follows its trend. However, significant gains are achieved by values of λ that strike a good balance between the two criteria, dds and dgs . The heuristic presented in Section 2.5.1, provides us with a starting value λ_H , which is equal to 5 for the data set into consideration. All our experiments with the real data show that $TKM-\lambda_H$, i.e., TKM having $\lambda=\lambda_H$, produces better results than the other two methods. This is illustrated by the graph $TKM-5$ in Figure 2.8(a). In addition, the experiments show that for values of λ lower than λ_H , TKM does not produce better results; i.e., the effect of dds is still not sufficient. On the other hand, we can get further improved results (by a factor of around 1% in our experimental data set), by tuning λ into a range of values belonging to the same order of magnitude as λ_H . (Obviously, the tuning of λ is required only once per data set.) In our experiments, we got the best performance of TKM for values of λ around 20, which, as demonstrated by the graph in Figure 2.8(a), produces slightly better precision than $TKM-5$. Further increasing the factor λ , i.e., the effect of the dds criterion, fails to provide better results, and, as expected, it eventually converges back to $TKDD$, as illustrated by the $TKM-50$ graph.

Next, we examine the resulting benefit of the dominance-based ranking compared to applying either of the individual similarity measures M0-M4. The recall-precision measures are illustrated in Figure 2.8(b). To avoid overloading the figure, only the $TKM-5$ and $TKM-20$ have been plotted. As shown, the dominance-based ranking clearly outperforms all the individual similarity measures.

As this is not very surprising, to better gauge the effectiveness of our method-

Table 2.22: IR metrics for all methods

Method	MAP	R-prec	bpref	R-rank	P@5	P@10	P@15	P@20
TKDD	<i>0.7050</i>	<i>0.6266</i>	<i>0.6711</i>	0.8333	<i>0.8071</i>	0.6893	0.6143	<i>0.5446</i>
TKDG	0.6750	0.6233	0.6334	0.8333	0.8143	<i>0.7143</i>	<i>0.6238</i>	0.5089
TKM-5	0.7249	0.6618	0.7098	<i>0.8393</i>	0.8000	0.7036	0.6738	0.5714
TKM-20	0.7375	0.6808	0.7243	<i>0.8393</i>	0.8000	0.7250	0.6857	0.5750
M0	0.5097	0.5128	0.5138	0.7217	0.6357	0.6071	0.5357	0.4464
M1	0.6609	0.5966	0.6313	0.8155	0.7571	0.6679	0.5738	0.5268
M2	0.6537	0.5903	0.6260	0.7708	0.7357	0.6536	0.5762	0.5232
M3	0.6595	0.5924	0.6254	0.8482	0.7357	0.6571	0.5762	0.5161
M4	0.6585	0.5822	0.6234	0.8127	0.7429	0.6571	0.5690	0.5250
Borda	0.6509	0.5778	0.6210	0.7577	0.7357	0.6464	0.5667	0.5179
MNZ	0.6588	0.5903	0.6274	0.8214	0.7357	0.6536	0.5738	0.5286
Outrank	0.6477	0.5811	0.6164	0.7575	0.7214	0.6500	0.5643	0.5179
Sum	0.6588	0.5903	0.6274	0.8214	0.7357	0.6536	0.5738	0.5286

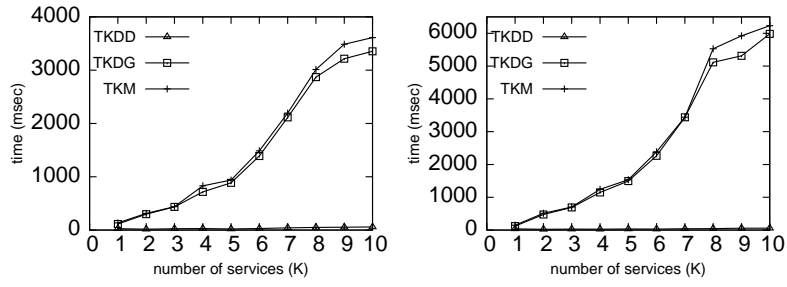
ology, we finally compare it to better informed approaches, as well. When multiple rankings exist, a common practice for boosting accuracy is to combine, or *fuse*, the individual results. Several methods, reviewed in Section 2.2.3, exist for this task. We compare our method to four popular fusion techniques: the score-based approaches CombSum and CombMNZ [52], the simple rank-based method of Borda-fuse [14], and the Outranking approach [51]. The first three techniques are parameter-free. On the other hand, the latter requires a family of outranking relations, where each relation is defined by four threshold values ($s_p, s_u, c_{min}, d_{max}$). We chose to employ a single outranking relation, setting the parameters to $(0, 0, M, M - 1)$, satisfying, thus, Pareto-optimality (M denotes the number of ranking lists, or criteria, which is 5 in our case). The obtained recall-precision graphs are shown in Figure 2.8(c). Again, our approach clearly outperforms the other methods. This gain becomes even more apparent, when noticing through Figures 2.8(b) and 2.8(c) that these fusion techniques, in contrast to our approach, fail to demonstrate a significant improvement over the individual similarity measures.

In addition to the recall-precision graphs discussed above, Table 2.22 details the results of all the compared methods for all the aforementioned IR metrics. For each metric, the highest value is shown in bold (we treat the values of both versions of TKM uniformly), whereas the second highest in italic. In summary, $TKDD$ and $TKDG$ produce an average gain of 8.33% and 6.44%, respectively, with respect to the other approaches. Additionally, $TKM-5$ and $TKM-20$ improve the quality of the results by a percentage (average values) of 11.44% and 12.56%, respectively.

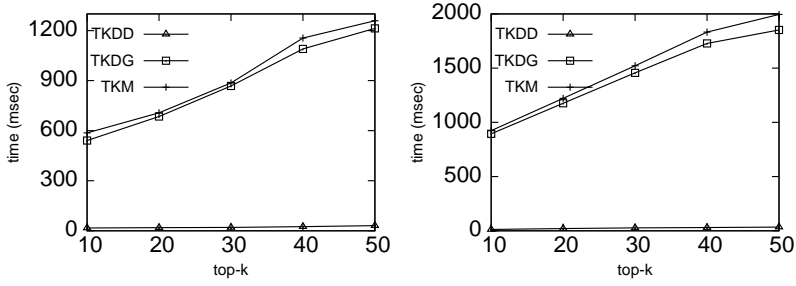
2.5.3.2 Computational Cost

In the following, we consider the computational cost of $TKDD$, $TKDG$, and TKM , for different values of the involved parameters. These parameters and their examined values are summarized in Table 2.23. Parameters N and k refer to the number of available services and the number of results to return, respectively. Parameter d corresponds to the number of parameters in the service request, i.e., the dimensionality of the match objects. Parameter M denotes the number of distinct matching criteria employed by the service matchmaker.

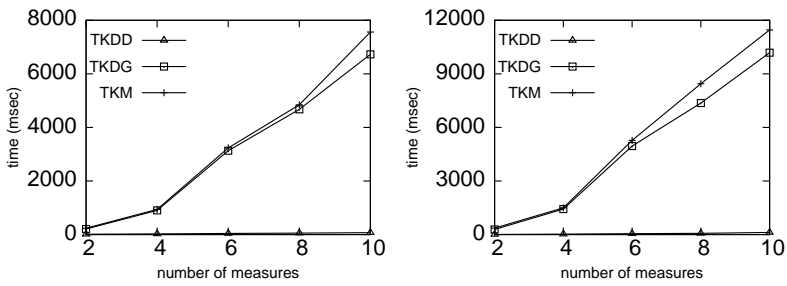
We first provide a theoretical analysis, and then report our experimental findings



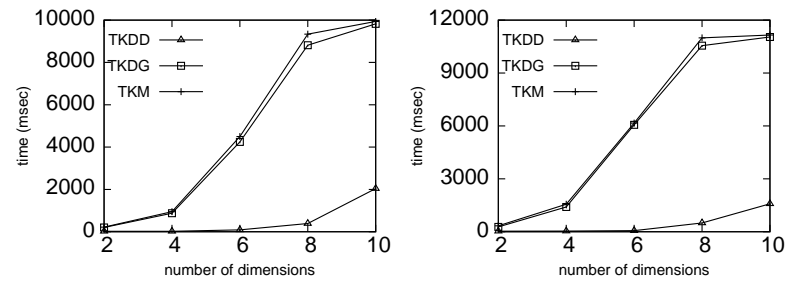
(a) Effect of N



(b) Effect of k



(c) Effect of M



(d) Effect of d

Figure 2.9: Effect of parameters under low (left graph of each pair) and high (right graph of each pair) variance var

on synthetically generated data sets.

Theoretical Analysis. To determine the dominated and dominating scores, our methods need to compare the instances of all services with each other, in the worst case. In total, there are $N \cdot M$ instances (i.e., M match instances per service), hence we perform $O(N^2M^2)$ dominance checks. For any pair of instances, a dominance check needs to examine the degrees of match for all d parameters. As a result, the complexity of our methods is $O(dN^2M^2)$. Clearly, this is a worst-case bound, as our algorithms need only find the top- k dominant services and employ various optimizations for reducing the number of dominance checks.

For the sake of comparison, we also discuss briefly the computational cost of the fusion techniques considered in Section 2.5.3.1. These take as input M lists, one for each criterion, containing the N advertised services ranked in decreasing order of their *overall* degree of match with the request. Therefore, an aggregation of the individual parameter-wise scores is required. CombSum, CombMNZ and Borda-fuse scan the lists, compute a *fused* score for each service and output the results sorted by this score. This procedure costs $O(NM + N \log N)$, where the first (second) summand corresponds to scanning (sorting). The Outranking method computes the fused score in a different manner: for each pair of services it counts agreements and disagreements as to which is better in the ranked lists. Therefore, its complexity is $O(N^2M)$. Note that all fusion techniques are independent of d due to the reduction of the individual parameter scores to a single overall score. In practice, the performance of $TKDG$ and TKM resembles that of the Outranking method, while $TKDD$ performs as well as the other fusion approaches. Therefore, for clarity of the presentation, in the rest of our analysis, we focus only on the three proposed methods.

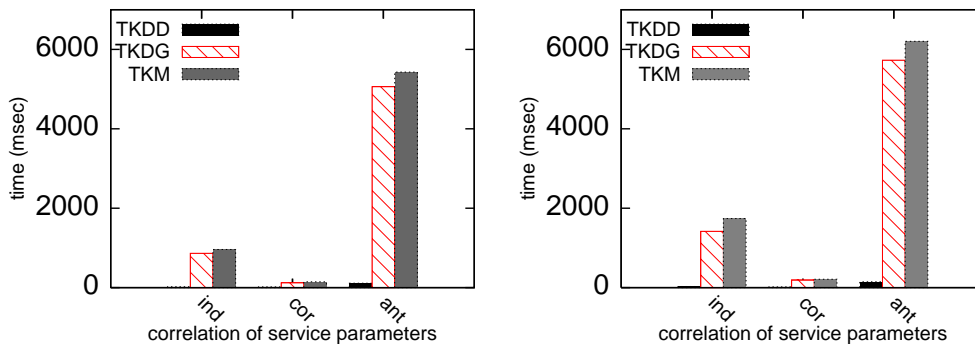
Experimental Analysis. We used a publicly available synthetic generator [130] to obtain different types of data sets, with varying distributions represented by the parameters *corr* and *var*, shown in Table 2.23. Given a similarity metric, parameter *corr* denotes the correlation among the degrees of match for the parameters of a service. We consider three distributions: in independent (*ind*), degrees of match are assigned independently to each parameter; in correlated (*cor*), the values in the match instance are positively correlated, i.e., a good match in some service parameters increases the possibility of a good match in the others; in anti-correlated (*ant*) the values are negatively correlated, i.e., good matches (or bad matches) in all parameters are less likely to occur. Parameter *var* controls the variance of results among similarity metrics. When *var* is low, matching scores from different criteria are similar. Hence, the instances of the same match object are close to each other in the d -dimensional space. On the other hand, when *var* is high, the matching scores from different criteria are dissimilar and, consequently, instances are far apart. We report our measures for variance around 10% (*low*) and 20% (*high*).

In all experimental setups, we investigate the effect of one parameter, while we set the remaining ones to their default values, shown bold in Table 2.23. As a default scenario, we consider a request with 4 parameters, asking for the top-30 matches of a set of 5K partially matching service descriptions, using 4 different similarity measures. For the factor λ in TKM , we used the value λ_H appropriately estimated for each corresponding data set. The results are presented in Figure 2.9.

In general, all experiments indicate that $TKDD$ is the most efficient method.

Table 2.23: Parameters and examined values

Parameter	Symbol	Values
Number of services	N	[1, 10]K, 5K
Number of results	k	10, 20, 30 , 40, 50
Number of dimensions	d	2, 4 , 6, 8, 10
Number of instances	M	2, 4 , 6, 8, 10
Parameter correlation	$corr$	ind , cor, ant
Instance variance	var	low, high

**Figure 2.10: Effect of $corr$ under low (left) and high (right) variance var**

As already discussed, $TKDD$ is interested in objects that dominate the top match objects; hence, it searches a relatively small portion of the data set. On the contrary, the search space for $TKDG$ is significantly larger, so its delay is expected. Similarly, TKM performance suffers mainly due to the impact of dgs score; therefore, it is reasonable that it follows the same trend as $TKDG$, with a slight additional overhead for accounting for dds score, as well. These observations are more apparent in Figure 2.9(a), where it can be seen that $TKDD$ is very slightly affected, as opposed to $TKDG$ and TKM , by the size of the data set. Another interesting observation refers to the effect of the dimensionality (Figure 2.9(d)), which at higher values becomes noticeable even for $TKDD$. This, in fact, is a known problem faced by the skyline computation approaches as well. As the dimensionality increases, it becomes increasingly more difficult to find instances dominating other instances; hence, many unnecessary dominance checks are performed. A possible work-around is to group together related service parameters so as to decrease the dimensionality of the match objects. For the same reasons, a similar effect is observed in Figure 2.10. For correlated data sets, where many successful dominance checks occur, the computational cost for all methods drops close to zero. On the contrary, for anti-correlated data sets, where very few dominance checks are successful, the computational cost is significantly larger.

Summarizing, the final choice of the appropriate ranking method depends on the application. All three proposed measures produce significantly more effective results than the previously known approaches. If an application favors more accurate results, then TKM seems as an excellent solution. If the time factor acts as the driving decision point, then $TKDD$ should be favored, since it provides high quality results (see Table 2.22) almost instantly (see Figures 2.9 and 2.10).

2.6 Summary

In this chapter, we have addressed the problem of (Semantic) Web service discovery and we have proposed methods for ranking offered services with respect to users' requests. First, a semantic similarity measure for ranking service descriptions was presented. Based on the notions of recall and precision, this measure determines the degree of match between the service request and advertisement. For this purpose, it assesses the semantic similarity of ontology classes, exploiting both the class hierarchy and the properties of the classes. It also takes into consideration service preconditions and effects. Next, we have formulated the problem of Semantic Web service selection as a skyline computation problem. We have shown how the best matches can be identified efficiently by a skyline computation algorithm, and we have discussed common tasks involved in the service selection process, referring both to the requesters' and the providers' perspectives. Finally, we have addressed the issue of top- k retrieval of Web services, with multiple parameters and under different matching criteria. We have presented three suitable criteria for ranking the match results, based on the notion of *dominance*, and we have provided corresponding algorithms for computing the best matches in each case. The proposed methods have been evaluated on both real and synthetic data sets.

Chapter 3

Service and Data Selection in Peer-to-Peer Networks

Although the client-server architecture has been the traditional and predominant paradigm in the Web, peer-to-peer networks have emerged in the previous years as an alternative to resolve critical issues such as scalability and fault-tolerance. The first and most common applications of peer-to-peer networks were related to massive file sharing (e.g., exchanging music or video files). Recently, research has focused on the use of structured overlays to efficiently search and manage the available resources, as well as on the sharing of structured data, where each peer uses a local schema to organize its contents.

In this chapter we deal with the problems of service discovery and data sharing in P2P environments. First, we present related work on these issues. Then, in Section 3.2 we describe an efficient encoding and indexing of service descriptions, and we show how, based on these, service descriptions can be stored and searched efficiently in an appropriate P2P overlay network. Section 3.3 deals with the use of ontologies on top of Peer Data Management Systems, and in particular with an environment where peers seek, and are satisfied with, information that is semantically similar, but not necessarily identical, to their requests. It investigates the notion of semantic similarity of peer schemas and of queries with their rewritten versions, and proposes a similarity measure to identify semantically relevant peers for propagating queries through the network. Finally, Section 3.4 concludes the chapter.

Our results in this chapter have been published in [147, 146].

3.1 Related Work

3.1.1 P2P Service Discovery

The majority of the works addressing the problem of Web service discovery has focused on centralized architectures (see Section 2.2.1 for a detailed presentation), which can not be easily adapted to the P2P case. For example, a typical approach in a centralized registry is to pre-compute and store, for each concept in the ontology, the list of services matching this concept (together with the type of match) [155]. Then, when a request is issued, the lists corresponding to the request parameters are looked up, and they are intersected to give the final results. Notice that, even though this is very efficient in terms of time, it imposes excessive storage requirements, and

fails to scale as the number of available services (i.e., the size of the stored lists) and the size of the ontologies (i.e., the number of lists to store) increase. This becomes even more evident, when considering service discovery in a distributed environment. Each time a new service is published, a very large number of peers needs to be updated, which makes this solution inappropriate. Instead, our approach stores only the service representations, which can be distributed in the P2P network, together with the corresponding index.

A P2P approach for Web service discovery is presented in [138]. However, the services are not semantically described; instead, the search is based on (possibly partial) keywords. Semantic Web service discovery in P2P networks has been studied in [119, 19]. In contrast to our work, these approaches deal with unstructured networks. In [94] Web service descriptions are indexed by keywords taken from domain ontologies, and are then stored on a DHT network. In [137] the peers are organized in a hypercube and the ontology is used to partition the network into concept clusters, so that queries are forwarded to the appropriate cluster. However, the subset of concepts to be used as structuring concepts should be known in advance. The approach in [170] distributes semantic service advertisements among available registries, by categorizing concepts into different groups based on their semantic similarity, and assigning groups to peers. In [167] services are distributed to registries depending on their type, e.g., a registry related to the travel domain will only maintain Web services specific to this domain. Instead, our work does not rely on some partitioning of the domain concepts. Furthermore, it supports ranking of the results, and it allows to retrieve the top- k matches progressively.

3.1.2 Semantics-based P2P Data Sharing

The importance of semantics in P2P overlays has been apparent from the early stages of research in this field. One of the first works to consider semantics is [44], which suggests the construction of semantic overlay networks (SONs). Another work in the steps of [44] is [156], which suggests the dynamic construction of the interest-based shortcuts in order for peers to route queries to nodes that are more likely to answer them. Towards this end, the works in [169] and [67] exploit implicit approaches for discovering semantic proximity based on the history of query answering and the least recently used nodes. Additionally, SQPeer is an extensive work on Peer Data Management Systems (PDMS) that share RDF data and localize the query patterns using views [85]. In the same spirit, our work focuses on overlays that share structured data, and it considers the problem of semantic similarity of schemas and queries.

In a different line of research, semantics have been considered in the specialized field of structured P2P overlays. GridVine deals with the distributed management of complex data and schemas of meta-data, specifically RDF [3]. The system allows schema inheritance, and it supports the creation and indexing of translation links that map pairs of schemas. Similarly, pSearch forms a structured semantic overlay [159]. Documents as well as queries are represented as semantic vectors. Both GridVine and pSearch rely for search efficiency on the structured form of the overlay, and, thus, their solution is not applicable to the semantic diversity problem in an unstructured P2P system, which is the focus of our work. Additionally, Bibster exploits ontologies in order to enable P2P sharing of bibliographic data [61]. Ontolo-

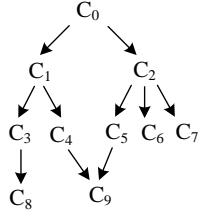
gies are used for importing data, formulating and routing queries, and processing answers. Peers advertise their expertise and learn through ontologies about peers with similar data and interests. However, Bibster does not incorporate the ontology information into any kind of semantic similarity, as our work does.

Our work is based on the definition of a semantic similarity measure to identify relevant peers and assess the quality of rewritten queries. Query similarity has been explored in several works in the recent past. The work in [5] deals with attribute similarity, but focuses on numeric data and on conclusions about similarity that can be deduced from the workload. Furthermore, in [53] queries are classified according to their structural similarity; yet, the authors focus on features that differentiate queries with respect to optimization plans. The works in [41] and [78] deal with semantic similarity which can be extracted from structural query features. Finally, [99] proposes the creation of a distributed index that is used to route queries effectively in a PDMS, given the semantic similarity of peer schemas. In contrast to these approaches, our work relies on the use of ontologies to semantically annotate the peer schemas, and then exploit these annotations to measure the similarity between peers as well as queries that are forwarded and rewritten in the network. In [2] a notion of syntactic similarity is used to measure the extent to which a query is preserved after transformation. To achieve semantic interoperability in a bottom-up, semi-automatic manner, two feedback mechanisms are presented: one at the schema level, namely analyzing query translations along cycles in the network, and another at the data level, namely analyzing query results obtained through composite translations. This approach can be viewed as complementary to ours, as it can be used to incrementally develop global agreements among the participating peers. Finally, a similarity measure for semantic concepts is proposed in [73]. The concepts are represented in disjunctive normal form (in Description Logics [15] notation), and their similarity is measured based on the overlap of these descriptions. However, this approach is not applicable in our case, since the proposed similarity measure is symmetric (for primitive concepts), and it decreases only when the reference concept is more specific than the examined one.

3.2 Service Discovery in Structured P2P Networks

3.2.1 Encoding Service Descriptions

In the following, we briefly review the process of Semantic Web service matchmaking (for more details see Chapter 2). We consider an ontology as a set of hierarchically organized concepts. Since multiple inheritance is allowed, the concepts form a rooted directed acyclic graph. The nodes of the graph correspond to concepts, with the root corresponding to the top concept, e.g., `owl:Thing` in an OWL ontology, whereas the edges represent subsumption relationships between the concepts, directed from the father to the child. To allow for semantic search of services on the Web, the description of a service is enhanced by annotating its parameters (typically inputs and outputs) with concepts from an associated ontology [31, 6, 60]. A service request is the description of a desired service, also annotated with ontology concepts. Figure 3.1a illustrates a sample ontology fragment, while a sample set of a service request and 3 service advertisements is shown in Figure 3.1b. The underlying assumption is that if a service provides as output (resp., accepts as input) a concept



(a)

	INPUTS	OUTPUTS
R	C_8	C_4, C_7
S₁	C_1	C_4, C_2
S₂	C_3	C_9, C_7
S₃	C_5	C_1

(b)

Concept	Intervals
C_0	[1,20]
C_1	[2,11]
C_2	[12,19],[8,9]
C_3	[3,6]
C_4	[7,10]
C_5	[13,14],[8,9]
C_6	[15,16]
C_7	[17,18]
C_8	[4,5]
C_9	[8,9]

(c)

Figure 3.1: (a) A sample ontology fragment, (b) A service request (R) and three service advertisements (S_1, S_2, S_3), (c) Intervals assigned to ontology concepts

C , then it is also expected to likely provide (resp., accept) the subconcepts of C . For instance, a service advertised as selling computers is expected to sell servers, desktops, laptops, PDAs, etc.; similarly, a service offering delivery in Europe is expected to provide delivery within all (or at least most) European countries.

Matchmaking of semantically annotated Web services is then based on subsumption reasoning between the semantic descriptions of the service request and the service advertisement. Along the lines of earlier works [118, 93], we specify the match between a service request R and a service advertisement S based on the semantic match between the corresponding parameters in their descriptions. More specifically, for a service parameter C_S and a request parameter C_R , we consider the match as *exact*, if C_S is equivalent to C_R ($C_S \equiv C_R$); *plug-in*, if C_S subsumes C_R ($C_S \sqsupset C_R$); *subsumes*, if C_S is subsumed by C_R ($C_S \sqsubset C_R$); *fail*, otherwise. Exact matches are preferable to plug-in matches, which in turn are preferable to subsumes matches. In the example of Figure 3.1, service S_1 provides one exact and two plug-in, service S_2 provides one plug-in, one subsumes, and one exact, whereas S_3 provides two fail and one plug-in matches.

Given that a large number of services may provide a partial match to the request, differentiating between the results within the same type of match is also required. Further following the aforementioned assumption regarding the semantics of a service description, we use as a criterion for assessing the degree of match between two concepts C_1 and C_2 the portion of their common subconcepts, or in other words, the extend to which the subtrees (more generally, subgraphs) rooted at C_1 and C_2

overlap. Intuitively, the higher the overlap, the more likely it is for the service to match the request. Thus, in the following, we consider the degree of match between two concepts C_1 and C_2 as

$$\text{degreeOfMatch}(C_1, C_2) = \frac{|\{C \mid C \sqsubseteq C_1 \wedge C \sqsubseteq C_2\}|}{\max(|\{C \mid C \sqsubseteq C_1\}|, |\{C \mid C \sqsubseteq C_2\}|)} \quad (3.1)$$

Returning to our example from Figure 3.1, notice that regarding the requested input, services S_1 and S_2 provide a plug-in match. However, using Equation (3.1), the degree of match for the service S_1 is $1/5$, whereas for the service S_2 is $1/2$. Notice that the proposed approach for service selection is not limited by this criterion. Different ranking criteria may be appropriate in different applications (for example, see Section 2.3 for a more elaborate similarity measure for ranking Semantic Web services). Our approach is generic and it can accommodate different ranking functions, as shown later in Section 3.2.2. Retrieving services in descending order of their degree of match to the given request constitutes an important feature for a service discovery engine. In the case that the requester is a human user, it can be typically expected that he/she will navigate only the first few results. In fact, experiments conducted in a recent survey [74] showed that the users viewed the top-1 search result in about 80% of the queries, whereas results ranked below 3 were viewed in less than 50% of the queries. Even though this study refers to Web search, it is reasonable to assume a roughly similar behavior for users searching for services. On the other hand, Semantic Web service discovery plays an important role in fully automated scenarios, where a software agent, such as a travel planning agent, acting on behalf of a human user, selects and composes services to achieve a specific task. Typically, the agent will select the top-1 match, ignoring the rest of the results. Hence, computing only the best possible match would be sufficient in this case. In fact, this often makes sense for human users as well; Google’s “I’m Feeling Lucky” feature is a characteristic example based on this assumption.

Invoking the reasoner to check for subsumption relationships between the ontology concepts annotating the service parameters constitutes a significant overhead, which has to be circumvented in order to allow for fast service selection at query time. For this purpose, we employ an appropriate service encoding based on labeling schemes [40]. The main idea works as follows. In the case of a tree hierarchy, each concept is labeled with an interval of the form $[begin, end]$. This is achieved by performing a depth-first traversal of the tree, and maintaining a counter, which is initially set to 1 and is incremented by 1 at each step. Each concept is visited twice, once before visiting any of its subconcepts and once after all its subconcepts have been visited. The interval assigned to the concept is constructed by setting its lower (resp., upper) bound to the value of the counter when the concept is visited for the first (resp., second) time. Observe that due to the way intervals are assigned, a concept C_1 is subsumed by another concept C_2 if and only if its interval is contained in that of C_2 , i.e., $I_{C_1} \subset I_{C_2}$. This scheme generalizes to the case of graphs, which is the typical case for ontologies on the Semantic Web, by first computing a spanning tree T and applying the aforementioned process. Then, for each non spanning tree edge, the interval of a node is propagated recursively upwards to its parents. Hence, more than one intervals may be assigned to each concept. As before, subsumption relationships are checked through interval containment: C_1 is subsumed by C_2 if and only if every interval of C_1 is contained in some interval of C_2 .

Type of match	Condition
exact	$\mathcal{I}_{C_R} = \mathcal{I}_{C_S}$
plug-in	$\mathcal{I}_{C_R} \subset \mathcal{I}_{C_S}$
subsumes	$\mathcal{I}_{C_R} \supset \mathcal{I}_{C_S}$

Table 3.1: Types of match using the intervals based encoding

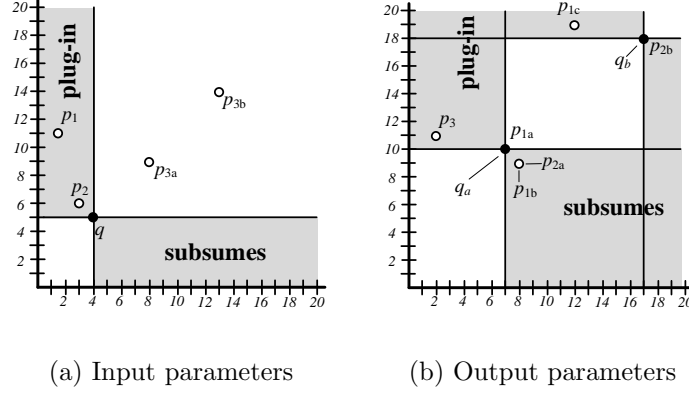


Figure 3.2: Interval based search

In our example, the intervals assigned to the ontology concepts are shown in Figure 3.1c, and have been computed considering the spanning tree formed by removing the edge (C_5, C_9) . Notice how the interval assigned to the concept C_9 is then propagated to the concepts C_5 , C_2 , and C_0 (in the latter, it is subsumed by the initially assigned interval).

Consequently, a service request or advertisement can be represented by the set of intervals associated to its input and output concepts. With this encoding, determining the type of match between two service parameters is reduced to checking for containment relationship between the corresponding intervals; a constant time operation. In particular, we can rewrite the conditions determining the type of match between a request parameter C_R and a service parameter C_S , as shown in Table 3.1, where \mathcal{I}_C denotes the set of intervals assigned to C .

Furthermore, the aforementioned ranking criterion can be expressed by means of the intervals based representation. For a concept C , the size of the subgraph rooted at C , G_C , is given by

$$|G_C| = \sum_{I \in \mathcal{I}_C} \left\lceil \frac{|I|}{2} \right\rceil \quad (3.2)$$

Hence, for two concepts C_1, C_2 , where $C_1 \sqsubseteq C_2$ or $C_1 \sqsupseteq C_2$, Equation 3.1 becomes:

$$\text{degreeOfMatch}(C_1, C_2) = \frac{\min\{|G_{C_1}|, |G_{C_2}|\}}{\max\{|G_{C_1}|, |G_{C_2}|\}} \quad (3.3)$$

3.2.2 Indexing Service Descriptions

The service representation presented in the previous section allows the evaluation of the type and degree of match between a pair of requested and offered services

in constant time. Still, the number of comparisons required is proportional to the number of available services. To further reduce the time required by the matcher, an index structure is employed for pruning the search space, keeping the number of comparisons required to a minimum. For this purpose, each interval is represented as a point in a 2-dimensional space, with the coordinates corresponding to the intervals' lower and upper bounds respectively, i.e., *begin* and *end*. Then, checking for containment between intervals is translated to a range query on this space. Figures 3.2(a) and 3.2(b) draw the input and output parameters, respectively, of the example in Figure 3.1. Points labeled as q_x , correspond to the parameters of the requested service, whereas p_{ix} correspond to parameters of the i -th offered service. For example, the output parameters of service S_2 is represented by points $p_{2a} = (8, 9)$ for class C_9 and $p_{2b} = (17, 18)$ for class C_7 . For a given interval, the intervals contained by it are those located in its lower-right region, whereas those containing it are located in its upper-left region.

In a centralized environment, where a single registry contains the information about all the advertised services and is responsible for performing the matchmaking and ranking process, an R-tree [59] can be used to expedite service selection. The R-tree is a typical multi-dimensional index structure. It partitions points in hierarchically nested, possibly overlapping, *minimum bounding rectangles* (MBR). Each node in the tree stores a variable number of entries, up to some predefined maximum. Leaf nodes contain data points, whereas internal nodes contain the MBRs of their children.

For convenience, we use two separate R-trees, T_{in} , T_{out} to index the services, where T_{in} (T_{out}) stores the intervals associated with the input (output) parameters. Thus, the corresponding R-tree is used when matching input or output parameters. Consider as an example the 3 services discussed in the previous section. Figure 3.3 shows the MBRs and the structure of the two R-trees. An MBR is denoted by N_i and its corresponding entry as e_i . Notice that points that are close in the space (e.g., p_1 , p_2 in Figure 3.3(a)) are grouped and stored in the same leaf node (N_2 in Figure 3.3(b)).

In the following we describe the algorithm (shown in Figure 3.2) for finding the services matching a request using our running example. The algorithm examines all request parameters in turn (Line 2). Assume that the first examined parameter par is the input corresponding to concept C_8 ; thus, T_{in} is examined (Line 3). The intervals, in this case $[4, 5]$, associated with the ontology concept is inserted in \mathcal{I} (Line 5). Subsequently, three queries are posed to T_{in} retrieving the exact matches under point $(4, 5)$ (Line 7), the plug-in matches inside the range extending from $(0, 5)$ up to $(4, \infty)$ (Line 8) and the subsumes matches inside $(4, 0)$ up to $(\infty, 5)$ (Line 9). A range query is processed traversing the R-tree starting from the root. At each node, only its children whose MBR overlaps with the requested range are visited. Similarly, for the case of a point query, only children whose MBR contains the requested point are visited. A small performance optimization is to perform the three queries in parallel minimizing, thus, node accesses. Subsequently, all matches to par are merged into m_{par} (Line 10). Once all parameters have been examined, the candidate services $\mathcal{S}_{\mathcal{R}}$ are constructed by intersecting the parameter matching results (Line 11). This retains only the services which match all request parameters. Since some services in $\mathcal{S}_{\mathcal{R}}$ can have additional input parameters that are not satisfied by the request, they are filtered out from the final result (Line 12).

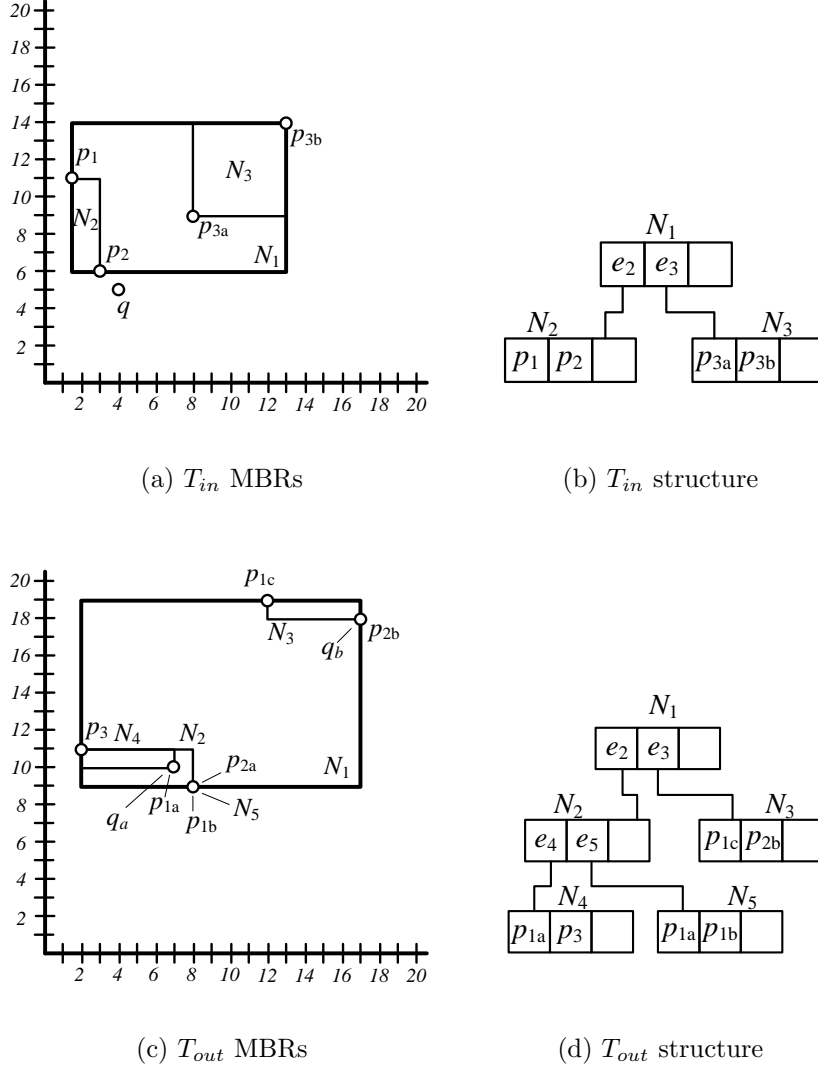


Figure 3.3: *R-trees example*

As discussed in Section 3.2.1, in many cases a ranked list of the top- k best matching services is preferred as the result of the matchmaking process. Table 3.3 illustrates the Progressive Search Algorithm to retrieve the top- k services given a request R using our framework. As before, we present the algorithm using our running example. Initially, all intervals associated with the request parameters are inserted into \mathcal{I} (Line 2). In particular, \mathcal{I} contains $[4, 5]$ (represented by point q in Figure 3.3(a)) for the input parameter, and $[7, 10]$, $[17, 18]$ (represented by points q_a , q_b , respectively, in Figure 3.3(c)) for the two output parameters. A heap H_I is associated with each interval $I = [i_s, i_e] \in \mathcal{I}$ (Lines 3–7); in our case there are 3 heaps for q , q_a and q_b . Initially, these heaps contain the root node of T_{in} or T_{out} , depending on the interval’s parameter type (Lines 5–6). Entries e_I in I ’s heap are R-tree nodes and are sorted increasingly by their *minimum distance* (MINDIST) to (i_s, i_e) . The MINDIST of a leaf node, i.e., a point, is its distance from (i_s, i_e) . The MINDIST of an internal node, i.e., an MBR, is the minimum distance of the MBR from (i_s, i_e) .

The Progressive Search Algorithm proceeds examining heap entries until k services have been retrieved (Lines 8–19). The heap whose head entry has the mini-

Search Algorithm	
	Input: request R , available services \mathcal{S} indexed in T_{in}, T_{out}
	Output: services $\mathcal{S}_{\mathcal{R}}$ matching R
1	begin
2	foreach $par \in IN_R \cup OUT_R$ do
3	if $par \in IN_R$ then $T \leftarrow T_{in}$
4	else $T \leftarrow T_{out}$
5	$\mathcal{I} \leftarrow$ the intervals associated with par
6	foreach $interval I = (i_s, i_e) \in \mathcal{I}$ do
7	$m_{par}^{ex} \leftarrow$ point $[i_s, i_e]$ query in T
8	$m_{par}^{pl} \leftarrow$ range $(0, i_e) \times (i_s, \infty)$ query in T
9	$m_{par}^{sb} \leftarrow$ range $(i_s, 0) \times (\infty, i_e)$ query in T
10	$m_{par} = m_{par}^{ex} \cup m_{par}^{pl} \cup m_{par}^{sb}$
11	$\mathcal{S}_{\mathcal{R}} = \bigcap_{par} m_{par}$
12	$\mathcal{S}_{\mathcal{R}} = \mathcal{S}_{\mathcal{R}} \setminus \{S : \exists IN_S \text{ not matched by any } IN_R\}$
13	return $\mathcal{S}_{\mathcal{R}}$
14	end

Table 3.2: Algorithm for index-based service matchmaking

Progressive Search Algorithm	
	Input: request R , available services \mathcal{S} indexed in T_{in}, T_{out}, k
	Output: services $\mathcal{S}_{\mathcal{R}}$ matching R , in descending order of degree of match
1	begin
2	$\mathcal{I} \leftarrow$ the intervals associated with all parameters in $IN_R \cup OUT_R$
3	foreach $I = [i_s, i_e] \in \mathcal{I}$ do
4	create a heap H_I
5	if I corresponds to some $par \in IN_R$ then insert in H_I root of T_{in}
6	else insert in H_I root of T_{out}
7	H_I entries are sorted increasingly by their MINDIST to (i_s, i_e)
8	while $k > 0$ do
9	find the heap H_I whose head entry has the minimum MINDIST
10	$e_I \leftarrow \text{pop}(H_I)$
11	if e_I is an internal node then insert in H_I all children of e_I
12	else
13	let S be the service corresponding to e_I
14	let par_I be the parameter corresponding to interval I
15	mark that S has a match for par_I
16	if S has matches for all parameters in $IN_R \cup OUT_R$ then
17	insert S in $\mathcal{S}_{\mathcal{R}}$; // S is a result
18	$k \leftarrow k - 1$
19	if $k = 0$ then return $\mathcal{S}_{\mathcal{R}}$
20	end

Table 3.3: Algorithm for progressively returning matches

mum MINDIST is selected (Line 9). In our example both heaps for q_a and q_b have MINDIST 0 as their head entry (T_{out} 's root) contains both q_a and q_b ; assume q_a 's heap is selected. The entry (node N_1 in *out*) is popped from the heap (Line 10) and since it is an internal node all its children are inserted in the heap (Line 11). Then, the heaps are examined again and q_a 's heap is selected, as node N_2 is in its head and has MINDIST 0. N_2 is popped and its children are inserted. Repeating the process once more, a leaf entry p_{1a} is popped, which corresponds to the first output parameter of service S_1 (Lines 13–14). We mark that S_1 has a match for a request parameter (Line 15). Then, S is checked if it has matches for all parameters, i.e., it is a result (Lines 16–19). The algorithm returns when k results have been found.

The output of the algorithm is the ranked service list S_2, S_1, S_3 . Notice that S_2 has a subsumes match but it is ranked higher than S_1 , having only exact and plug-in matches. Further, S_3 is included even though it has two fail matches. This is due to the fact that the MINDIST function described does not discriminate among points in different regions with respect to the point corresponding to a request parameter's interval. For example, in Figure 3.3(c) p_{2a} and p_{1b} are closer to q_a than p_3 and are regarded as better matches to parameter q_a , even though they are only subsumes matches (they lie in the lower right quadrant w.r.t. q_a). To obtain arbitrary rankings as described in Section 3.2.1, MINDIST can be trivially modified to be region aware. For example, it can evaluate heap entries that correspond to plug-in as closer compared to subsumes matches.

3.2.3 Managing Services in the P2P Overlay

As the availability and demand for Web services grows, the issue of managing Semantic Web services in a distributed environment becomes vital. Thus, we focus next on a scalable and fault-tolerant solution that is adaptable and efficient in a distributed environment. More specifically, we consider service discovery in a flat, structured P2P overlay network, since it provides self-maintenance and robustness, as well as efficiency in data management

Before discussing distributed service discovery, we have to choose a suitable framework. To support the adaptation of the algorithms presented in the previous section, such a framework must support both point and range queries, so as to allow for the retrieval of both exact and plug-in/subsumes matches, respectively. Furthermore, since our proposed service encoding and service search algorithm are based on the 2-dimensional space, it is necessary to select a P2P framework that is efficient and scalable for 2-dimensional data. More specifically, the selected P2P framework should preserve locality and directionality, if possible.

SpatialP2P [77] is a recently proposed structured P2P framework, targeted to spatial data. It handles areas, which are either cells of a grid-partitioned space or sets of cells that form a rectangular. The basic assumption of the framework is that each area has knowledge of its own coordinates and the coordinates of some other areas to which it is directly linked. The goal of SpatialP2P is to guarantee that any stored area can be searched and reached from any other, solely by exploiting local area knowledge.

Figure 3.4 shows an example of a SpatialP2P overlay with four peers. Each peer maintains links to others towards the four directions of the 2D space. The grid is hashed to the four peers, such that each cell is stored and managed by the closest

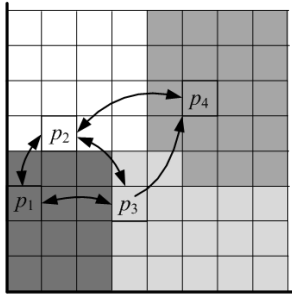


Figure 3.4: Illustrative SpatialP2P overlay

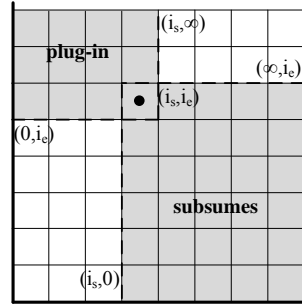


Figure 3.5: Search regions

peer. In the figure cells and their storing peers share the same color.

In SpatialP2P, search is routed according to locality and directionality. This means that search is propagated to the area that is closer to and towards the same direction with the sought area, choosing from the available areas that are linked to the one on which the search is currently iterated.

The management of services in the P2P overlay consists of two basic operations: insertion of services and search for services in the system. Search can be either exhaustive, i.e., seeking for any possible results, or top- k , i.e., seeking the k best-matching results. In the following we discuss the details of these operations.

Service insertion. In order to use the P2P framework for the distributed management of Semantic Web services, we assume that the ID space of the overlay (i.e. the space of values for node and data IDs) corresponds to the space of values defined by the encoding of the service descriptions.

When a new service is published, its description is encoded using the intervals based representation presented in Section 3.2.1, and then it is inserted in the network. Specifically, each encoded service parameter is hashed to and eventually stored by the peer whose ID is closer to its value in the 2-dimensional space. Each inserted service parameter is accompanied by some meta-data about the respective type, (input or output), as well as the service it belongs to.

The locality-preserving property of the SpatialP2P overlay guarantees that similar service parameters are stored by the same or neighboring peers. By similar, we mean services whose input and output parameters correspond to matching concepts. Moreover, the preservation of directionality means that following subsequent peers in a particular direction results, for example, in locating concepts subsuming or subsumed by the ones previously found. As described below, these properties are essential for minimizing the search time, and this applies to both exhaustive range and top- k queries.

Service search. Searching for services in the P2P overlay is performed by an adaptation of the search algorithm of Section 3.2.2 to the SpatialP2P API. For each requested service parameter, a point or a range query is performed, depending on the requirement of exact, plug-in or subsumes match with the available service parameters. An exact request for a service parameter corresponding to interval $I = [i_s, i_e]$ is performed by a point query asking the retrieval of the point (i_s, i_e) , if such data exists in the overlay. For plug-in and subsumes requests for a parameter associated to the interval $I = [i_s, i_e]$, a pair of range queries is initiated. Since the data space is bounded (recall the intervals construction from Section 3.2.1), these requests are represented by range queries for rectangular areas. Specifically, for

plug-in matches, a query requesting the range extending from $(0, i_e)$ up to (i_s, ∞) is issued, while for the subsumes request, the corresponding range is $(i_s, 0) \times (\infty, i_e)$ (see Figure 3.5). The results of these two queries are unified to provide the answer to the requested parameter. Parallel searches are conducted for each requested parameter, and the results are finally intersected to compute the final matches.

Finding the top- k matches. SpatialP2P supports top- k search by extending search for range queries to dynamically increase the respective range. In detail, a search for a service parameter represented by an interval $I = [i_s, i_e]$ is initiated as the minimum range query that includes (i_s, i_e) ; thus, the minimum range is extended only in the grid cell in which the point (i_s, i_e) resides. After the search is performed in this minimum range, if the number of retrieved results is lower than k , then the range is increased towards the desired direction of the 2D space by the minimum, i.e., by one grid cell. The process repeats iteratively, until k results have been retrieved (or the whole space has been searched).

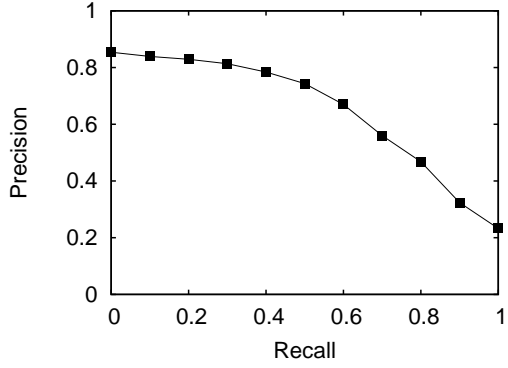
3.2.4 Experimental Evaluation

Experimental setup. We have evaluated our approach on two data sets. For the first data set, to simulate a real-world scenario, we used the OWL-S service retrieval test collection OWLS-TC v2 [115]. This collection contains services retrieved mainly from public IBM UDDI registries, and semi-automatically transformed from WSDL to OWL-S. More specifically, it comprises: (a) a set of ontologies, derived from 7 different domains (education, medical care, food, travel, communication, economy and weapons), comprising a total of 3500 concepts, used to semantically annotate the service parameters, (b) a set of 576 OWL-S services, (c) a set of 28 sample requests, and (d) the relevance set for each request (manually identified).

The second data set was synthetically generated, based on the first one, so as to maintain the properties of real-world service descriptions. In particular, we constructed a set of approximately 10K services, by creating variations of the 576 services of the original data set. For each original service, we selected randomly one or more input or output parameters, and created a new service description by replacing them with randomly chosen superconcepts or subconcepts from the corresponding domain ontology. A set of 100 requests was generated following the same process, based on the original 28 requests. All the experiments were conducted on a Pentium D 2.4GHz with 2GB of RAM, running Linux.

Ranking. In the first set of experiments we used the first data set to evaluate the effectiveness of the service selection approach. For each of the 28 queries we retrieved the ranked list of match results, and compared them against the provided relevance sets. We use well-established IR metrics [161] to evaluate the performance of the search and ranking process. In particular, Figure 3.6(a) depicts the micro-averaged recall-precision curves for all the 28 queries, i.e., the precision (averaged over all queries) for different recall levels. Observe that a 30% of the relevant services can be retrieved with precision higher than 80%, whereas for retrieving more than 70% of the relevant services the precision drops below 50%. Also, the following metrics are presented in Figure 3.6(b): (a) precision at k , i.e., the (average) precision after k results have been retrieved; (b) success at k , i.e., whether a relevant result has been found after k results have been retrieved.

As we can see, the precision drops below 70% after the top-10 matches have been

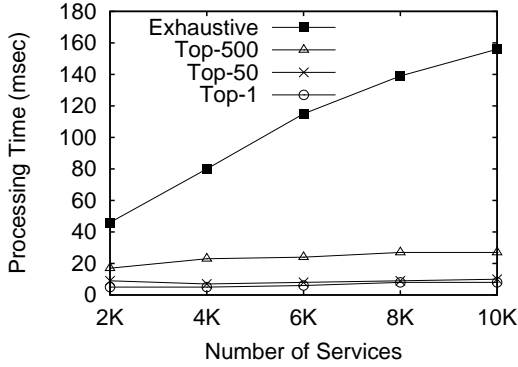


P@5	0.764
P@10	0.721
P@15	0.631
S@1	0.857
S@2	0.964
S@4	1

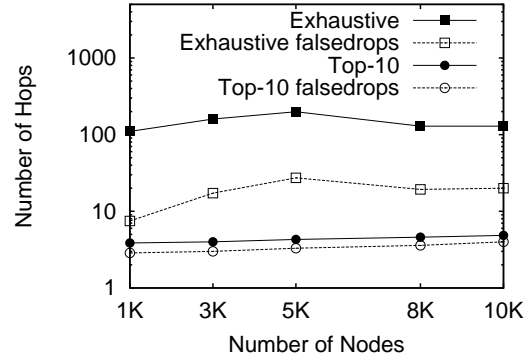
(a)

(b)

Figure 3.6: (a) Recall-precision curve, (b) Precision and Success at k ($P@k$, $S@k$)



(a) Centralized



(b) P2P

Figure 3.7: Search cost for: (a) centralized registry, (b) P2P registry

retrieved. Moreover, for the set of 28 queries, in 24 of them the top-1 match is a relevant one, in 27 queries there is a relevant result among the top-2 matches, and in all cases there is a relevant result among the top-4 matches. The above results opt for the emphasis on top- k queries and on fetching results progressively, as discussed in Section 3.2.1.

Experiments for centralized search. In this set of experiments we measured the time required by our search algorithm to discover and rank services in a centralized registry. In particular, we investigated the performance benefits, i.e., the reduction in response time, resulting from restricting the search to retrieving only the top- k matches. For this purpose, we used the synthetically generated data set described previously. We varied the number of services from 2K up to 10K, and we measured the processing time (averaged over 100 queries) for retrieving: (i) all matches, and (ii) the top- k matches for $k \in \{1, 50, 500\}$. The experimental results are illustrated in Figure 3.7(a). Notice the significant savings in the processing time when restricting the search to top- k matches, as well as the fact that the processing time in the latter case is significantly less sensitive to the number of available services.

Experiments for search in P2P environment. In the last set of experiments,

we evaluated our search method in a P2P environment, as described in Section 3.2.3. We varied the size of the P2P network, from 1K up to 10K peers, and we inserted a total of 10K services. We conducted two experiments. In the first experiment, we retrieved, for each request, all the identified matches, whereas in the second, we restricted the search range to obtain (approximately) the top-10 results for each request. For each of the experiments we report two measures: (i) total number of hops (i.e., number of peers processing the query), and (ii) number of falsedrops (i.e., number of peers on the search path not contributing to the result set). The results are shown in Figure 3.7b. Both measures are quite low and relatively stable w.r.t. the size of the network. As discussed in Section 3.2.3, this is due to the fact that SpatialP2P is particularly designed to preserve the locality and the directionality of the data space, thus queries are effectively routed towards peers containing relevant information. As in the centralized case, the search cost is significantly lower, when retrieving only the top- k matches.

3.3 Ontology-based Data Sharing in a PDMS

3.3.1 Problem Description

In this section we consider the problem of sharing structured data in a flat, unstructured Peer Data Management System (PDMS) using ontologies to identify peers containing relevant information. More specifically, we consider a PDMS accompanied by one or more domain ontologies, which are used to semantically annotate the content a peer makes available to the network. Note that the use of these ontologies does not contravene the requirements of the lack of global schema and peer autonomy: peer schemas do not have to adhere to any restrictions; they may just use terms from these ontologies to semantically describe their elements.

Each peer possesses a local database exposing a relational schema. Queries are issued according to this schema. Each peer shares data with its acquainted peers via a set of mappings, which are used for query rewriting between the respective schemas. We focus our study on select-project-join (SPJ) queries, and mappings of the well known forms global-as-view (GAV), local-as-view (LAV) and global-local-as-view (GLAV) [63] as they are adapted to the P2P paradigm [91, 65]. A query Q specifies the information to be retrieved, by means of a set of attributes to be returned (SELECT clause), and a set of conditions to be applied (WHERE clause).

In addition, we assume the existence of a domain ontology, providing a shared conceptualization of the domain of interest for the community of peers (observe Figure 3.8a.) The domain ontology may be provided by a third-party, such as a standardization organization. This is a realistic hypothesis for a wide range of applications, involving networks where peers are professionals, companies, or organizations (e.g., universities, libraries, hospitals), exchanging information about a specific domain. An alternative case is the collaborative construction of the ontology within the peer network itself. In large-scale peer-to-peer networks, where global consensus is difficult to achieve and maintain, there may exist several ontologies, allowing different views of the domain, and clusters of peers using either of these ontologies (observe Figure 3.8b.) In these cases the proposed approach is still applicable, provided that mappings between these ontologies are available. For simplicity, we first introduce the proposed similarity measure assuming the existence

OWL construct	Notation	Description
owl:Class	C	Classes
owl:ObjectProperty	P	Object properties
owl:DatatypeProperty	P	Datatype properties
owl:equivalentClass	$C_1 \equiv C_2$	Class equivalence
rdfs:subClassOf	$C_1 \sqsubseteq C_2$	Class subsumption
owl:equivalentProperty	$P_1 \equiv P_2$	Property equivalence
rdfs:subPropertyOf	$P_1 \sqsubseteq P_2$	Property subsumption
owl:Thing	\top	The class containing all the individuals
owl:Nothing	\perp	The class containing no individuals
owl:DataRange	d	Data types
rdfs:domain	$domain(P)$	The domain of a property
rdfs:range	$range(P)$	The range of a property
owl:allValuesFrom	$\forall P.C$	Value restrictions on object properties
owl:allValuesFrom	$\forall P.d$	Value restrictions on datatype properties
owl:minCardinality	$\geq_n P$	Min cardinality restriction
owl:maxCardinality	$\leq_n P$	Max cardinality restriction

Table 3.4: OWL constructs and notation used

of a shared ontology (Sections 3.3.2 and 3.3.3), and then we extend it to deal with the case of multiple ontologies (Section 3.3.4).

OWL is used for representing the ontology. In particular, Table 3.4 summarizes the OWL constructs and notation used in the following. Additionally, we use the notation $P(C)$ to denote the set of properties that are related to a class C , and the notation $R(P_C)$ to denote the set of restrictions on a property P with respect to a class C .

The domain ontology is used to semantically annotate the schemas of the peers participating in the network, describing the type of information that a peer makes available to other peers. The semantic annotation of a peer’s schema is achieved by declaring correspondences between terms in the peer schema and terms in the domain ontology. The high-level architecture of the system is depicted in Figure 3.8. Solid lines represent pairwise mappings between acquainted peers, while dashed lines represent correspondences between elements of the local schema and elements of the domain ontology.

Definition 3.3.1. *A semantic peer is a tuple $\mathcal{P} = (\mathcal{R}, \mathcal{O}, \mathcal{A})$, where \mathcal{R} is the peer’s database schema, \mathcal{O} the domain ontology used, and \mathcal{A} the peer’s semantic annotation. \mathcal{A} holds the set of annotations A for the relations in \mathcal{R} . Each A consists of a pair of the form (R, \mathcal{C}) and a set of pairs of the form $(R.t, C_i.P)$, $C_i \in \mathcal{C}$. That is, a relation R is semantically annotated by means of a set of classes \mathcal{C} . Each $C_i \in \mathcal{C}$ is an ontology class, possibly enhanced with additional restrictions to make explicit the semantics of the underlying relation, i.e., $C_i \equiv C'_i \sqcap_k R_k$, $C'_i \in \mathcal{O}$. Attributes $R.t$ are annotated by means of properties of the same set of classes, i.e., $(R.t, C_i.P)$, $C_i \in \mathcal{C}$.*

Our work aims at the exploitation of the information conveyed by the ontology and the annotations to provide a measure that represents how semantically close are two peers \mathcal{P}_1 and \mathcal{P}_2 , namely $Sem_Sim(\mathcal{P}_1, \mathcal{P}_2)$. Furthermore, when a query Q is forwarded by \mathcal{P}_1 to \mathcal{P}_2 and is rewritten to Q' based on the corresponding mappings, then it is usually degraded. This means that some part of Q cannot be rewritten on peer \mathcal{P}_2 [78]. Hence, our goal is to provide a measure of the degree of

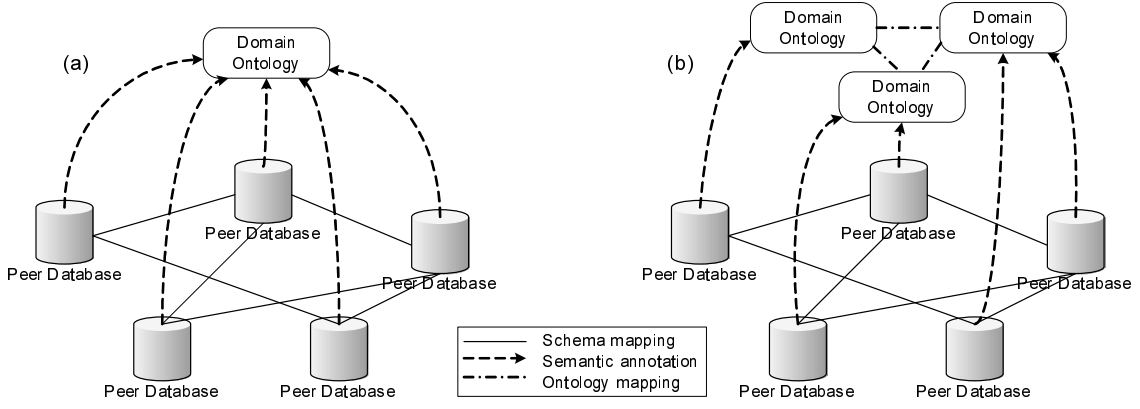


Figure 3.8: *Unstructured network of semantic peers with (a) single ontology (b) multiple ontologies*

match between the requested information Q and the retrieved information Q' , i.e., $Sem_Sim(\mathcal{P}_1, \mathcal{P}_2, Q, Q')$.

Having such qualitative measures for the semantic relationship between the peers is important, when, for instance, a peer chooses its acquaintances or evaluates the quality of the answers returned by another peer.

Motivating example. As a motivating example, consider a simple scenario where in the context of a social network system, two peers, \mathcal{P}_1 and \mathcal{P}_2 , want to exchange data about music bands. Suppose that \mathcal{P}_1 and \mathcal{P}_2 have the following schemas and mapping:

$$\begin{aligned}
 \mathcal{P}_1 &: bands(name, members, year) \\
 \mathcal{P}_2 &: bands(name, singer, year) \\
 M_{\mathcal{P}_1, \mathcal{P}_2} &: bands(name, members, year) : - bands(name, singer, year)
 \end{aligned}
 \tag{3.4}$$

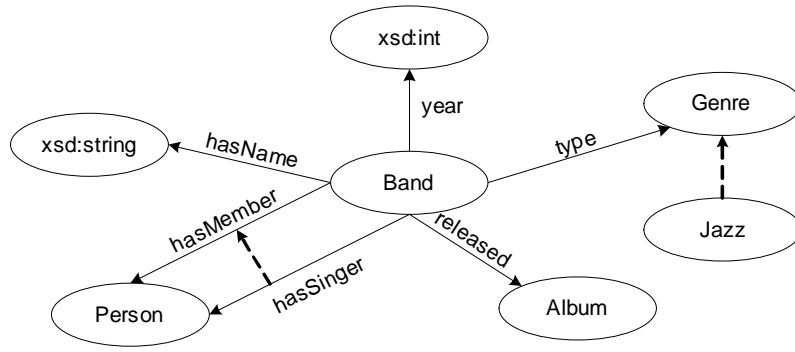
An ontology for the music domain is used to describe semantically the contents of the peers. A sample snippet of such ontology is illustrated in Figure 3.9. Nodes represent classes or datatypes, while edges represent properties. Dashed lines between two classes or properties represent subsumption relation.

Furthermore, we assume that \mathcal{P}_1 contains information about bands having at least one album, while \mathcal{P}_2 , being more specialized, stores information about bands that play Jazz music, were formed before the year 2000, and have released at least 3 albums. These facts can be made explicit by each peer, by annotating the relations and attributes in its local schema using terms from the domain ontology, as shown in Table 3.5, where $Band_{\mathcal{P}_1}$ and $Band_{\mathcal{P}_2}$ two new classes defined as follows:

$$\begin{aligned}
 Band_{\mathcal{P}_1} &: Band \sqcap \geq_1 released \\
 Band_{\mathcal{P}_2} &: Band \sqcap \forall type. Jazz \sqcap \geq_3 released \sqcap \forall year. (\leq 2000)
 \end{aligned}
 \tag{3.5}$$

For simplicity, we have assumed a single relation for each peer in this example. The case of multiple relations linked with foreign keys is handled similarly: the class definition would contain an additional object property, corresponding to the foreign key, and having as range the class annotating the linked relation.

Exchanging data between \mathcal{P}_1 and \mathcal{P}_2 is meaningful and useful for both peers. However, information available at \mathcal{P}_2 is only partially sufficient for the information



```

<owl:Class rdf:ID="Genre"/>
<owl:Class rdf:ID="Jazz">
  <rdfs:subClassOf rdf:resource="#Genre" />
</owl:Class>
<owl:ObjectProperty rdf:ID="hasMember">
  <rdfs:domain rdf:resource="#Band" />
  <rdfs:range rdf:resource="#Person" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSinger">
  <rdfs:subPropertyOf rdf:resource="#hasMember" />
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:about="#hasName">
  <rdfs:domain rdf:resource="#Band"/>
  <rdfs:range rdf:resource="&xsd:string"/>
</owl:DatatypeProperty>

```

Figure 3.9: A sample ontology and a snippet of the corresponding XML representation

Schema element	Ontology element	Schema element	Ontology element
bands.name	hasName	bands.name	hasName
bands.members	hasMember	bands.singer	hasSinger
bands.year	year	bands.year	year
bands	Band_P1	bands	Band_P2

(a) Peer \mathcal{P}_1

(b) Peer \mathcal{P}_2

Table 3.5: Semantic annotation of peer schemas

needs of \mathcal{P}_1 , while, inversely, \mathcal{P}_1 contains much information that is irrelevant for the interests of \mathcal{P}_2 . This obviously affects the quality of results that may be obtained by each peer. Notice also the *asymmetry* in the two directions. A qualitative measure is needed so that each peer may evaluate how suitable a particular acquaintance is, both in general, as well as with respect to a specific query.

3.3.2 Comparison of Peer Schemas

In the following, we propose a measure for the semantic similarity between the type of information provided by two peers, hereafter referred to as $Sem_Sim(\mathcal{P}_1, \mathcal{P}_2)$. This measure essentially builds on the one defined in Section 2.3.2.1 for the similar problem of Semantic Web service matchmaking; thus it is also based on the notions of

recall and precision, which express, respectively, the proportion of relevant material actually retrieved in answer to a search request, and the proportion of retrieved material that is actually relevant. Using the pair of values (*recall*, *precision*) to express the degree of semantic similarity between two peers, the proposed measure provides an intuitive way for (a) accounting for the asymmetry resulting from the specific direction considered in the comparison, and (b) expressing the extent to which the information provided by a peer is a subset or superset of the requested information. Although it is possible to express the similarity measure in terms of the F-measure, in the following we will refer to the proposed measure as a pair of values (*recall*, *precision*), i.e., keeping track of both the individual measures, as they can be of different value and use to each peer. In particular, each peer can decide whether to employ the F-measure and, if so, determine the value of parameter a appropriately (see Equation 2.2).

The semantics of the peer's schema is made explicit by its annotation, which comprises a set of classes. Thus, instead of estimating the semantic similarity $Sem_Sim(\mathcal{P}_1, \mathcal{P}_2)$ between two peers \mathcal{P}_1 and \mathcal{P}_2 , it suffices to estimate the similarity between the respective sets of classes in the ontology, i.e., $Sem_Sim(\mathcal{C}_{\mathcal{P}_1}, \mathcal{C}_{\mathcal{P}_2})$. For simplicity, we first focus our study on the similarity between two classes and then, we extend the results for comparing two sets of classes. Hence, to adapt the notions of recall and precision in our context, we consider as relevant items the instances of the class in the semantic annotation of the requesting peer and as retrieved items the instances of the class in the semantic annotation of the provider peer. Therefore:

$$\begin{aligned} recall(\mathcal{C}_{\mathcal{P}_1}, \mathcal{C}_{\mathcal{P}_2}) &= \frac{|\{x \mid x \in (\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2})\}|}{|\{x \mid x \in \mathcal{C}_{\mathcal{P}_1}\}|} \\ precision(\mathcal{C}_{\mathcal{P}_1}, \mathcal{C}_{\mathcal{P}_2}) &= \frac{|\{x \mid x \in (\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2})\}|}{|\{x \mid x \in \mathcal{C}_{\mathcal{P}_2}\}|} \end{aligned} \tag{3.6}$$

Notice that the above definitions for recall and precision have the following properties:

- When the two classes are equivalent, i.e., $\mathcal{C}_{\mathcal{P}_1} \equiv \mathcal{C}_{\mathcal{P}_2} \equiv \mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2}$, then $recall = 1$ and $precision = 1$, meaning that the contents of the two peers refer to the same type of entities.
- When $\mathcal{C}_{\mathcal{P}_1} \sqsubseteq \mathcal{C}_{\mathcal{P}_2}$, then $\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2} \equiv \mathcal{C}_{\mathcal{P}_1}$, thus $recall = 1$ and $precision < 1$, meaning that \mathcal{P}_2 contains information that is not of interest for \mathcal{P}_1 .
- When $\mathcal{C}_{\mathcal{P}_1} \sqsupseteq \mathcal{C}_{\mathcal{P}_2}$, then $\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2} \equiv \mathcal{C}_{\mathcal{P}_2}$, thus $recall < 1$ and $precision = 1$, meaning that \mathcal{P}_2 can only partial cover the information needs of \mathcal{P}_1 .
- When $\neg(\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2} \sqsubseteq \perp)$, then $recall < 1$ and $precision < 1$, meaning that part of the information of each peer is of interest to the other.
- Finally, when $\mathcal{C}_{\mathcal{P}_1} \sqcap \mathcal{C}_{\mathcal{P}_2} \sqsubseteq \perp$, then $recall = 0$ and $precision = 0$. Essentially this means that the acquaintance with this peer is of no use.

Calculating the recall and precision from Equations 3.6 requires knowledge about the extensions of the schemas, possibly using statistical techniques. However, in this

condition	$recall(R_1, R_2)$	$precision(R_1, R_2)$
$R_1 \equiv R_2$	1	1
$R_1 \sqsubseteq R_2$	1	0.5
$R_1 \sqsupseteq R_2$	0.5	1
$\neg(R_1 \sqcap R_2 \sqsubseteq \perp)$	0.5	0.5
$R_1 \sqcap R_2 \sqsubseteq \perp$	0	0

Table 3.6: *Different cases of subsumption relationship between two restrictions R_1 and R_2*

paper, we focus on a different approach and calculate these measures based on the semantic information conveyed by the domain ontology, and in particular (a) the class hierarchy, (b) the property hierarchy, and (c) the restrictions on the properties of the classes. As the above approaches may be used together, our future plans include the improvement of the method by using both approaches in a combined way.

In the case of classes that are not described by any properties, the recall and precision can be measured by the ratio of their common ancestors. This is a common approach for measuring the similarity between classes in a taxonomy [97, 133]. Thus, if $A(C)$ denotes the set of superclasses of a class C , then:

$$\begin{aligned}
 recall(C_1, C_2) &= \frac{|A(C_1) \cap A(C_2)|}{|A(C_2)|} \\
 precision(C_1, C_2) &= \frac{|A(C_1) \cap A(C_2)|}{|A(C_1)|}
 \end{aligned} \tag{3.7}$$

Notice that the values of recall and precision obtained from Equations 3.7 adhere to the cases discussed previously.

Similarly, the recall and precision between two properties, p_1 and p_2 , can be measured by the ratio of their common superproperties, as inferred by the property hierarchy in the domain ontology:

$$\begin{aligned}
 recall(p_1, p_2) &= \frac{|A(p_1) \cap A(p_2)|}{|A(p_2)|} \\
 precision(p_1, p_2) &= \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|}
 \end{aligned} \tag{3.8}$$

Concerning restrictions defined on the properties of a class, three different cases can be identified: (a) value restrictions on object properties (e.g., $\forall type.Jazz$); (b) value restrictions on datatype properties (e.g., $\forall year.(\leq 2000)$); and (c) cardinality restrictions (e.g., $\geq 3released$). In the first case, we consider the recall and precision between the two classes to which the values of the property are restricted. The other two cases can be handled by checking for a subsumption relationship between the two restrictions R_1 and R_2 as depicted in Table 3.6. More accurate results may be obtained in the case that statistical knowledge about the value distributions of the underlying data is available.

Thus, Equations 3.8 are updated to account also for the existence of restrictions, as follows:

$$\begin{aligned}
recall(p_1, p_2) &= \frac{|A(p_1) \cap A(p_2)|}{|A(p_2)|} \cdot \prod_{R(p_2)} recall(R'_i(p_1), R_i(p_2)) \\
precision(p_1, p_2) &= \frac{|A(p_1) \cap A(p_2)|}{|A(p_1)|} \cdot \prod_{R(p_1)} precision(R_i(p_1), R'_i(p_2))
\end{aligned} \tag{3.9}$$

where $R(p)$ is the set of restrictions on property p , and $R(p), R(p')$ denote a pair of corresponding restrictions; i.e., of the same type. The product is used in Equations 3.9 as a decreasing monotonic function that captures the intuition that each factor contributes negatively to the final result. If no corresponding restriction is set on one of the compared properties, then, in the case of value restriction on object properties, the restricted range is compared to the default range of the property. In the other two cases, the value of recall or precision, accordingly, is set to a fixed value; the default is 0.5. Here, we do not consider the case of other intermediate values in the range $(0, 1)$ that may capture richer ranking semantics; e.g., $year < 2005$ may be preferable to $year < 2007$ for a request $year < 2004$; currently, both have precision equal to 0.5. (However, in that case one should deal with the issue of data range and distribution; a problem that we have left as a future extension to our approach.)

Therefore, given two classes C_1 and C_2 , the recall and precision is calculated by adding the results for their individual properties and normalizing to the number of properties:

$$\begin{aligned}
recall(C_1, C_2) &= \frac{\sum_{P(C_2)} recall(p(C_1), p'(C_2))}{|P(C_2)|} \\
precision(C_1, C_2) &= \frac{\sum_{P(C_1)} precision(p(C_1), p'(C_1))}{|P(C_1)|}
\end{aligned} \tag{3.10}$$

where $P(C)$ denotes the set of properties of class C , and $p(C_1), p'(C_2)$ a pair of corresponding properties in the compared classes.

Example (Cont'd). We demonstrate the presented approach by applying the derived equations to estimate the semantic similarity between the schemas of the two peers, \mathcal{P}_1 and \mathcal{P}_2 , of the motivating example introduced in the previous section. The schema of each peer comprises a single relation, semantically described by the definitions shown in the formulae 3.5. Given these definitions, and the ontology shown in Figure 3.9, from Equations 3.10 follows that:

$$\begin{aligned}
recall(\mathcal{P}_1, \mathcal{P}_2) &= (3 \cdot 1 + 3 \cdot 0.5) / 6 = 0.75 \\
precision(\mathcal{P}_1, \mathcal{P}_2) &= 1
\end{aligned}$$

The above results reflect, as expected, the fact that the type of information provided by peer \mathcal{P}_2 is more restricted with respect to that provided by peer \mathcal{P}_1 .

The comparison can be extended to sets of classes, by comparing each class in the one set to its matching class in the other set (i.e., the class maximizing recall or precision, accordingly), and then normalizing to the cardinality of the sets. Therefore, for two sets of classes, \mathcal{C}_1 and \mathcal{C}_2 , the following equations hold:

$$\begin{aligned} recall(\mathcal{C}_1, \mathcal{C}_2) &= \frac{\sum_{C_i \in \mathcal{C}_1} \max_{C_j \in \mathcal{C}_2} recall(C_i, C_j)}{|\mathcal{C}_1|} \\ precision(\mathcal{C}_1, \mathcal{C}_2) &= \frac{\sum_{C_j \in \mathcal{C}_2} \max_{C_i \in \mathcal{C}_1} precision(C_i, C_j)}{|\mathcal{C}_2|} \end{aligned} \quad (3.11)$$

3.3.3 Comparison of Rewritten Queries

So far, we have considered only the semantic annotations of the peers, which essentially refer to all the content stored in the peer. Therefore, the previous analysis applies to the case of unrestricted exchange of information between two peers. However, the similarity needs to be evaluated also with respect to a specific query issued at a peer, as well as the mappings between two peers, which determine how the query is rewritten.

When a query Q_o is forwarded from \mathcal{P}_1 to \mathcal{P}_2 , it is rewritten according to the mappings, resulting in a query Q_r . During the rewriting process, some attributes may not be rewritten, or may be approximately rewritten, while conditions may be lost or inserted, due to the nature of the specified mappings (e.g., due to value constraints in the mappings). As a result, the retrieved information may not completely adhere to the initial request. Therefore, the previous analysis needs to be extended to consider two additional factors:

- the portion of attributes that were rewritten and how accurate the rewriting was, and
- the conditions specified, both in the original query Q_o and its rewritten version Q_r .

Any conditions existing in the query, either directly specified by the user or resulting from the mappings, further restrict the information requested or provided by the peer. Hence, these conditions need to be taken into account, together with the restrictions already existing in the classes semantically describing the peer's schema. For this purpose, each condition in the query is translated to a corresponding value restriction, which is added to the respective class in the peer's annotation. The process goes as follows:

1. A query Q_o is issued at peer \mathcal{P}_1 .
2. Based on the peer's semantic annotations, the set of classes \mathcal{C}_{Q_o} is selected, containing the classes annotating the relations in Q_o .
3. Each class in \mathcal{C}_{Q_o} is enhanced with additional value restrictions on its properties, according to the conditions specified in Q_o , resulting in the set $\mathcal{C}_{Q_o,e}$.

Algorithm SSR

Input: The original query Q_o , issued at peer $\mathcal{P}_1 = (\mathcal{R}_1, \mathcal{O}, \mathcal{A}_1)$
The target peer $\mathcal{P}_2 = (\mathcal{R}_2, \mathcal{O}, \mathcal{A}_2)$
The mappings $M_{1,2}$ between the schemas of the two peers

Output: The recall and precision measures between the original query Q_o
and the produced rewritten query Q_r

1. **Begin**
2. $\mathcal{R}_{Q_o} \leftarrow$ the set of relations appearing in Q_o
3. $\mathcal{C}_{Q_o} \leftarrow$ the set of classes annotating the relations in \mathcal{R}_{Q_o}
4. $\mathcal{C}_{Q_o,e} \leftarrow \mathcal{C}_{Q_o}$
5. **Foreach** condition w in Q_o {
6. $R.t \leftarrow$ the attribute to which w is applied
7. $C \leftarrow$ the class corresponding to R
8. $P \leftarrow$ the property corresponding to t
9. $D \leftarrow$ the class or datarange representing the restricted value
10. $C \leftarrow C \sqcap \forall P.D$
11. $\mathcal{C}_{Q_o,e} \leftarrow$ update definition of C
12. }
13. $Q_r \leftarrow \text{rewrite}(Q_o, M_{1,2})$
14. $\mathcal{C}_{Q_r,e} \leftarrow$ repeat lines 2-12 for Q_r
15. $\text{recall} = \text{recall}(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e})$
16. $\text{precision} = \text{precision}(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e})$
17. **End.**

Table 3.7: Algorithm for measuring the semantic similarity for rewritten queries

4. Q_o is sent to peer \mathcal{P}_2 , where it is rewritten as Q_r , according to the corresponding mappings.
5. Steps 2 and 3 are repeated for the rewritten version of the query and the semantic annotations of \mathcal{P}_2 , resulting in the set of classes $\mathcal{C}_{Q_r,e}$.

Afterwards, the first step to evaluate the quality of the performed rewriting is to calculate the recall and precision between the two sets of classes $\mathcal{C}_{Q_o,e}$ and $\mathcal{C}_{Q_r,e}$. This is achieved by Equations 3.11. The aforementioned process is formally described by the algorithm SSR, which is depicted in Table 3.7.

The second factor to consider is the rewriting of the attributes appearing in the SELECT part of the query. If an attribute t was rewritten to an attribute t' , then the quality of this rewriting is measured by the value of recall and precision between the corresponding properties p_t and $p_{t'}$ annotating these attributes. This is achieved by Equations 3.8. The sum over all attributes is then calculated and normalized to the number of attributes in the query. If a SELECT attribute failed to be rewritten, then the value of recall for it is zero, as the corresponding information can not be retrieved. Precision is not affected, since no redundancy in the results is caused.

The results from the two factors are multiplied, since both contribute negatively to the quality of the performed rewriting:

$$\begin{aligned}
recall(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e}, Q_o, Q_r) &= \frac{\sum_{t \in SELECT(Q_o)} recall(p_t, p_{t'})}{|SELECT(Q_o)|} \times recall(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e}) \\
precision(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e}, Q_o, Q_r) &= \frac{\sum_{t' \in SELECT(Q_r)} precision(p_t, p_{t'})}{|SELECT(Q_r)|} \times precision(\mathcal{C}_{Q_o,e}, \mathcal{C}_{Q_r,e})
\end{aligned} \tag{3.12}$$

Example (Cont'd). We revisit the example of Section 3.3.1 to demonstrate two indicative cases.

Case 1. Assume that \mathcal{P}_1 is interested in bands formed at the 80's, so it issues the query:

Q_o : *SELECT name, members, year FROM bands*
WHERE year ≥ 1980 AND year < 1990

Based on this query, the definition of \mathcal{P}_1 is updated as:

$\mathcal{C}_{Q_o,e}$: *Band* $\sqcap \geq_1 released \sqcap \forall year.([1980, 1990))$

The query is then forwarded to \mathcal{P}_2 , and is rewritten as:

Q_r : *SELECT name, singer, year FROM bands*
WHERE year ≥ 1980 AND year < 1990

Notice that the attribute *members*, corresponding to the property *hasMember*, has been rewritten as *singer*, corresponding to the property *hasSinger* \sqsubseteq *hasMember*. The definition of \mathcal{P}_2 is then updated accordingly:

$\mathcal{C}_{Q_r,e}$: *Band* $\sqcap \forall type.Jazz \sqcap \geq_3 released \sqcap \forall year.([1980, 1990))$

Being more restrictive, the condition on attribute *year* overwrites the previously existing restriction in the definition.

Applying Equations 3.12 results in:

$$\begin{aligned}
recall(\mathcal{P}_1, \mathcal{P}_2, Q_o, Q_r) &= [(2 \cdot 1 + 0, 5)/3] \cdot [(4 \cdot 1 + 2 \cdot 0.5)/6] = 0.7 \\
precision(\mathcal{P}_1, \mathcal{P}_2, Q_o, Q_r) &= 1
\end{aligned}$$

Notice how the value of recall is affected negatively by the rewriting of *members* to *singer*, and positively by the presence of the condition on attribute *year* in the issued query, which counteracts the respective restriction in the annotation of \mathcal{P}_2 .

Case 2. Assume now that the attribute *year* was not present in \mathcal{P}_2 's schema or in the mapping. Then, the rewritten query may be:

Q_r : *SELECT name, singer FROM bands*

Since no conditions are present in the rewritten query, \mathcal{P}_2 's definition is not affected. Applying Equations 3.12 results in:

$$\begin{aligned} recall(\mathcal{P}_1, \mathcal{P}_2, Q_o, Q_r) &= [(1 + 0.5 + 0)/3] \cdot [(4 \cdot 1 + 2 \cdot 0.5)/6] = 0.42 \\ precision(\mathcal{P}_1, \mathcal{P}_2, Q_o, Q_r) &= 1 \cdot (5.5/6) = 0.92 \end{aligned}$$

Notice that the failure to rewrite the attribute *year* significantly reduces the value of recall. Due to that failure, the returned bands have the restriction $year \leq 2000$, instead of the requested $year \in [1980, 1990)$, which negatively affects the value of precision.

3.3.4 Extending to Multiple Ontologies

In the previous two sections, we have presented a semantic similarity measure for peer schemas and queries propagated in the P2P network, assuming the existence of a single ontology that is used to annotate the elements of the schemas exposed by each peer. Even though this constitutes a valid assumption for several applications, maintaining an agreement to a common ontology, as the network size grows, becomes increasingly difficult. Therefore, it is important to consider how the proposed approach generalizes to the case of an environment where different peers may employ different ontologies to describe their schemas, as shown in Figure 3.8b. To maintain semantic interoperability in the absence of a common ontology, appropriate mappings need to be established between the terms used in the involved ontologies. In fact, a significant body of work exists addressing the issue of schema/ontology matching, as this constitutes a critical step in a variety of applications, such as data warehouses, catalog integration, agent communication, Web services coordination, and so on. In this section, we give an overview of the basic techniques underlying the state-of-the-art approaches for ontology matching and extend our similarity measure for comparing elements mapped from different ontologies.

Mapping elements from different ontologies. A recent survey of ontology matching techniques is presented in [140]. As in [140], we consider as a mapping between the elements e and e' of two ontologies O and O' , respectively, a tuple (id, e, e', M, c) , where: id is a unique identifier for the particular mapping element; M is the relation between the elements e and e' , i.e. equivalence (\equiv), subsumption (\sqsubset), overlap (\sqcap); and $c \in [0, 1]$ is a measure expressing the degree of confidence that M holds. Typically the matching process exploits, usually in a hybrid or composite way, techniques of the types presented below.

- *Element matching.* Similarity between elements is calculated based on their names and descriptions (i.e. labels, comments). Usually, a pre-processing occurs first, using Natural Language Processing techniques, such as tokenization (for example, *EmployeeRecord* \rightarrow \langle *Employ, Record* \rangle), lemmatization/stemming (e.g., *salaries* \rightarrow *salary*), and stopword elimination (e.g., removing articles or prepositions.) Then, string matching techniques are applied, such as prefix or suffix matching, edit distance, and n-grams (see [42] for a comparison of string matching techniques).

- *Structure matching.* In contrast to the previous case, where two elements are compared in isolation, these techniques consider the relations of these elements to other elements, using tree- or graph-matching techniques. For example, the similarity between two inner nodes can be calculated based on the similarity of their children. Alternatively, model-based approaches encode the intended semantics of each node, together with domain and structural knowledge, in a set of logical formulae, shifting the problem to one of logical satisfiability, that can be solved by employing standard SAT solvers.
- *Auxiliary information.* It is possible that the matcher uses external resources to obtain additional information to guide the matching process. Typical cases are the use of domain-specific thesauri, general-purpose dictionaries (e.g. WordNet [106]), repositories of previously mapped elements/structures, user feedback.

Translating elements to a different ontology. Once appropriate mappings between two ontologies have been established, either manually, semi-automatically or automatically, these mappings can be used to merge the two ontologies or to translate elements from one ontology to the other. Examples of tools for ontology merging are OntoMerge [47] and PROMPT [112]. However, creating and maintaining a merged ontology incurs a significant overhead.

On the other hand, a translation service for OWL ontologies is presented in [109]. The translation relies on a provided mapping between the vocabularies of the two ontologies. Then, a class C_1 from the source ontology can be characterized as *strongly-translatable*, *equivalent*, *identical*, *weakly-translatable* or *approximately-translatable* to a class C_2 from the target ontology, depending on its name mapping and the *translatability* of its associated properties and restrictions.

For the translation process, we follow a similar approach, which however differs in two issues. First, instead of the above categorization of the translatability of a class, we need to derive a quantitative measure for the quality of the translation. Second, the aforementioned work translates classes from the source ontology to already existing classes from the second ontology. For example, assume that:

$$C_1 : Department \sqcap \geq_{20} hasEmployee \text{ and } C_2 : Team \sqcap \geq_{10} hasWorker$$

are two classes from the source and target ontology, respectively. Then, with respect to the name mappings:

$$(Department \rightarrow Team) \text{ and } (hasEmployee \rightarrow hasWorker)$$

C_1 is strongly-translatable to C_2 . However, we translate a class expression by “rewriting” it according to the name mappings, i.e., replacing each term in the expression to its corresponding term in the target ontology, thus allowing the result of the translation to be a potentially new expression in the target ontology. Thus, in the previous example, the translated expression would be:

$$C'_1 : Team \sqcap \geq_{20} hasWorker.$$

The intuition behind that lies in the need to deal with the generated class expressions for annotating user queries.

Therefore, we consider a translation function f_T , that translates a (simple or complex) class or property from a source ontology O_1 to a target ontology O_2 as follows:

- a simple class or property is translated to the class or property specified by the corresponding mapping
- a complex class is translated by translating the terms in its definition; if a term is not translatable, the corresponding part of the expression is omitted.

As discussed at the beginning of this section, the considered mapping elements have the form (id, e, e', M, c) , with $c \in [0, 1]$ and $M = (\textit{equivalent}, \textit{more general}, \textit{less general}, \textit{overlap})$. (For more complex translations, in the presence of mappings which are themselves class expressions, a theoretical investigation is provided in [20].) Apparently, a translation may often result in loss of quality, either due to the (unintended) generalization or specialization of the original concept's meaning or due to the inability to translate (part of) it. Therefore, the quality of the translation has to be measured and taken into account. We use the measures of recall (r_T) and precision (p_T) for this purpose. Specifically, for a property p , given the corresponding mapping element (id, p, p', R, c) , we measure the translation recall and precision by:

$$(r_T(p), p_T(p)) = ((0.5^x \cdot c, 0.5^y \cdot c)) \quad (3.13)$$

where

$$(x, y) = \begin{cases} (0, 0) & \text{if } M = \textit{equivalent} \\ (1, 0) & \text{if } M = \textit{more general} \\ (0, 1) & \text{if } M = \textit{less general} \\ (1, 1) & \text{if } M = \textit{overlap} \end{cases} \quad (3.14)$$

If an element is not translatable then both recall and precision are equal to zero. Notice that the value 0.5 used in Equation 3.13 is a default value. Other values may be used, derived, for example, from knowledge of the application domain or the confidence level of the match provided by the matcher.

The measurement for a class is derived similarly, with the difference that the translatability of its properties, including potential restrictions, is also taken into account. Due to the way the translation is performed, cardinality restrictions, as well as value restrictions on datatype properties are always translatable, provided that the property on which the restriction is applied is translatable. For a value restriction R_p on an object property p (e.g. $\forall p.C$), the translatability of the restriction is dependent on the translatability of the class being the filler of the restriction (i.e. C), denoted by $\phi(R_p)$. Thus, the following equations hold.

$$\begin{aligned}
r_T(C) &= 0.5^x \cdot c \cdot \frac{\sum_{P(C)} r_T(p) \cdot r_T(\phi(R_p))}{|P(C)|} \\
p_T(C) &= 0.5^y \cdot c \cdot \frac{\sum_{P(C)} p_T(p) \cdot p_T(\phi(R_p))}{|P(C)|}
\end{aligned} \tag{3.15}$$

where (x,y) as in Equation 3.14. Finally, when translating sets of classes, the quality of the translation can be assessed by the average quality of translation of the classes in the set:

$$r_T(\mathcal{C}) = \frac{\sum_{C \in \mathcal{C}} r_T(C)}{|\mathcal{C}|} \quad , \quad p_T(\mathcal{C}) = \frac{\sum_{C \in \mathcal{C}} p_T(C)}{|\mathcal{C}|} \tag{3.16}$$

Comparing elements across ontologies. Given the above procedures for matching and translating elements between different ontologies, the next step is to extend the introduced similarity measure to cover these cases. In Section 3.3.2, we defined a measure of semantic similarity, in terms of the notions of recall and precision, for pairs of properties, classes, and sets of classes belonging in the same ontology (see Equations 3.9, 3.10 and 3.11, respectively). In the following, we extend these functions (distinguished by the symbol \neq) so that they can be applied to elements from different ontologies. This is based on the observation that additional loss of quality may result due to the inaccurate translation (or no translation) of (parts of) the compared elements. Therefore, assuming that e_1 is an element (i.e., a property p , a class C or a set of classes \mathcal{C}) from the source ontology O_1 , $e'_1 = f_T(e_1)$ its translation to the target ontology O_2 , and e_2 the corresponding (i.e., the most similar) element of e'_1 in the target ontology, then:

$$\begin{aligned}
recall^{\neq}(e_1, e_2) &= r_T(e_1) \cdot recall(e'_1, e_2) \\
precision^{\neq}(e_1, e_2) &= p_T(e_1) \cdot precision(e'_1, e_2)
\end{aligned} \tag{3.17}$$

3.4 Summary

This chapter has dealt with the search of services and data in P2P settings. First an approach for P2P service discovery has been presented. It employs a suitable encoding for the service descriptions, and it indexes these representations to effectively prune the search space, consequently reducing the search engine's response time. To allow for scalability, we describe how the service representations can be distributed in a suitable structured P2P overlay network, and we show how the search is performed in this setting. Second, we have dealt with issues that arise in P2P systems consisting of peer databases, i.e., peers that share structured data through the use of schema mappings. In such systems, information is requested by queries that are issued on local schemas and are rewritten to schemas of acquainted peers through mappings. We have proposed the enhancement of such a system with an ontology describing the domain of interest of the participating peers. We have

discussed the semantic diversity between peer schemas, as well as between queries and their rewritten versions on other peers. The use of domain ontologies enables peers to semantically annotate their elements despite the absence of a global schema. Using these ideas, we have proposed a similarity measure for schemas and queries based on the notions of recall and precision. The measure introduced takes into consideration the semantic annotations of schema elements and the structure and semantics of queries, as well as of the mappings used for the rewriting.

Chapter 4

Ontology-based Design of Extract-Transform-Load Processes

Once appropriate data or services have been discovered, they need to be homogenized or aggregated to meet the final user needs and requirements. In data warehouses, Extract-Transform-Load (ETL) processes are special-purpose processes that integrate data from heterogeneous operational sources to a central repository. A similar case found in the Web is mashups, where information is gathered and combined from several sources and then presented to the user. To facilitate the design of such processes, existing approaches and tools have focused on providing a graphical environment where the user can draw the flow of data and the transformations between the sources and the target. However, this still remains a complex and error-prone activity, and thus there is a need for methods that will semi-automate the design, by identifying and proposing operations to include in the ETL process.

This chapter deals with the ontology-driven conceptual design of ETL scenarios. The next section discusses related work. Section 4.2 describes a graph-based representation for the schemas of the data sources and the target repository, and their annotation through an appropriate ontology. It also shows how correspondences and conflicts between the sources and the target are then automatically inferred, thus driving the conceptual design of the ETL scenario. Section 4.3 presents our approach to the design of ETL processes through graph transformations. It formalizes the construction of the ETL scenario as a series of graph transformations, and presents a set of graph transformation rules that insert appropriate operations into the ETL scenario so that the original data are transformed to meet the target specifications. Finally, Section 4.4 concludes the chapter.

Our results in this chapter have been published in [150, 152, 151, 154, 143].

4.1 Related Work

In this section we discuss related work, focusing first on conceptual models for ETL processes, and, more generally, data warehouses. Then, we consider related work in the areas of data integration, semantic schema matching, and mashups.

4.1.1 Conceptual models for ETL processes and DWs

The problem of ETL process design has been studied in the context of Data Warehouses and several approaches have been proposed [82, 83]. The conceptual modeling of ETL scenarios has been studied in [166]. ETL processes are modeled as graphs composed of transformations, treating attributes as first-class modeling elements, and capturing the data flow between the sources and the targets. In another effort, ETL processes are modelled by means of UML class diagrams [162]. A UML note can be attached to each ETL operation to indicate its functionality in a higher level of detail. The main advantage of this approach is its use of UML, which is a widespread, standard modeling language. In a subsequent work, the above approaches have converged, so as to provide a framework that combines both their advantages, namely the ability to model relationships between sources and targets at a sufficiently high level of granularity (i.e., at the attribute level), as well as a widely accepted modeling formalism (i.e., UML) [98]. However, these approaches are only concerned with the graphical design and representation of ETL processes. Similarly, existing commercial tools facilitate the design of ETL workflows through convenient Graphical User Interfaces and libraries of pre-defined, customizable transformations, without however providing any mechanism for the automatic identification of the appropriate transformations based on the semantics of the datastores involved [71, 72, 104, 113]. More recent work has focused on the aspect of optimization of ETL processes, providing algorithms to minimize the execution cost of an ETL workflow with respect to a provided cost model [144]. Still, it is assumed that an initial ETL workflow is given, while the problem of how to derive it is not addressed. In another line of research, the problem of ETL evolution has been studied [122]. Although that work presents a novel graph model for representing ETL processes, it cannot be used in our framework, since it is not suitable for incorporating our ontology-based design and it concretely deals more with physical aspects of ETL, rather than with the conceptual entities we consider here.

On the other hand, there exist some approaches concerning the (semi-)automation of several tasks of logical data warehouse design from conceptual models, but they do not provide a formal method to specifically determine the flow of data from the source recordsets towards the data warehouse. A couple of approaches concern the development of dimensional models from traditional ER-models [17, 108]. Other approaches focus on generating the logical schema from the conceptual schema. Approaches for deriving the data warehouse schema from the conceptual schemas of operational sources are described in [24, 69]. [55] presents a general methodological framework for data warehouse design, based on the Dimensional Fact Model. BabelFish, a modeling framework concerning the automatic generation of OLAP schemata from conceptual graphical models is presented in [62], discussing also the issues of this automatic generation process for both the OLAP database schema and the front-end configuration. Algorithms for the automatic design of data warehouse conceptual schemata are presented in [126]. An approach based on the Model Driven Architecture is presented in [102]. Finally, the conceptual design of a data warehouse from XML sources has been addressed in [56], where the increased importance of XML data for modern organizations, and consequently the need to incorporate XML data in data warehouses, is discussed, pointing out the main issues arising from the fact that XML is used to model semi-structured data. The Dimensional Fact Model is adopted as the conceptual model for the data warehouse and a semi-automatic

approach for the conceptual design from XML sources is presented, emphasizing on the different ways of representing relationships in DTDs and XML Schemas.

The use of ontologies in data warehouses and related areas has already produced some first research results, as, for example, in data warehouse conceptual design [101] and in On-Line analytical Processing [111]. A semi-automated method exploiting ontologies for the design of multidimensional data warehouses is presented in [134]. These efforts are complementary to our work, as in our case the ontology is used specifically for the design of the ETL process. Ontologies have been used also for data cleaning purposes, e.g., in [81], and in Web data extraction [58].

4.1.2 Data Integration and Semantic Schema Matching

In a different line of research, the theoretical aspects of the problem of data integration from heterogeneous sources have been extensively investigated in the literature [32, 92, 64]. The typical architecture of a data integration system comprises a set of data sources, containing the actual data, and a global schema, providing an integrated view over these underlying sources. The two basic approaches for modeling the relation between the sources and the global schema are global-as-view (GAV), where the global schema is expressed in terms of the data sources, and local-as-view (LAV), where the global schema is defined independently and each source is described as a view over it. One of the main tasks in the design of a data integration system is to establish the mappings between the sources and the global schema.

The use of Description Logics in data integration systems has also been studied in the literature. The problem of rewriting queries using views in Description Logics has been investigated in [20, 33]. The SIMS project addresses the issue of integrating heterogeneous sources, by using a shared ontology to establish a fixed vocabulary for describing the data sets in the domain [13]. Queries are expressed in domain terms and are reformulated into queries to specific information sources. A formal framework for representing inter-schema knowledge in cooperative information systems has been presented in [36]. Another approach towards data integration and reconciliation in a data warehouse environment is based on a conceptual representation of the application domain [34]. The approach follows the local-as-view paradigm, relying on a declarative description of the data sources in terms of the enterprise conceptual model, which is expressed in an Entity-Relationship formalism.

Data exchange is a similar problem, concerning the transformation of data structured under one schema into data structured under another schema. The fundamental theoretical issues of data exchange have been investigated in [48, 49, 86, 95]. Data exchange between XML schemas has also been studied [12, 127].

However, in contrast to the approaches described above, the transformations taking place in a typical ETL scenario usually include operations, such as the application of functions, that can not be captured by a query rewriting process.

Finally, our work has some commonalities with approaches for semantic schema matching [140, 54], which take as input two graph-like structures and produce a mapping between the nodes of these graphs that correspond semantically to each other. First, in a pre-processing phase, the labels at the graph nodes, which are initially written in natural language, are translated into propositional formulas to explicitly and formally codify the label's intended meaning. Then, the matching problem is treated as a propositional unsatisfiability problem, which can be solved

using existing SAT solvers. Due to this formalism, the following semantic relations between source and target concepts can be discovered: equivalence, more general, less general, and disjointness. Instead, our approach can handle a larger variety of correspondences, including, for example, convert, extract or merge operations, which can be further extended to support application-specific needs.

4.1.3 Mashups

Mashups constitute a new paradigm of data integration becoming popular in Web 2.0. Despite their different characteristics and requirements compared to ETL processes –mainly the facts that the latter are typically offline procedures, designed and maintained by database experts, while the former are online processes, targeted largely for end users– both activities share a common goal: to extract data from heterogeneous sources, and to transform and combine them to provide added value services. Recently deployed mashup editors, such as Yahoo! Pipes [173], Microsoft Popfly [105], and the Google Mashup Editor [57], as well as recent research efforts [70], aim at providing an intuitive and friendly graphical user interface for combining and manipulating content from different Web sources, based mainly on the “dragging and dropping” and parametrization of pre-defined template operations. The process is procedural rather than declarative and does not support the use of metadata to facilitate and automate the task. Hence, our proposed formalism and methodology can be beneficial in this direction. In fact, our approach is likely even more readily applicable in such context, in the sense that often the semantic annotation of the sources may already be in place.

An approach for automatically composing data processing workflows is presented in [8]. Data and services are described using a common ontology to resolve the semantic heterogeneity. The workflow components are described as Semantic Web services, using relational descriptions for their inputs and outputs. Then, a planner uses relational subsumption to connect the output of a service with the input of another. To bridge the differences between the inputs and outputs of services, the planner can introduce *adaptor* services, which may be either pre-defined, domain-independent relational operations (i.e., selection, projection, join, and union) or domain-dependent operations. However, in contrast to our approach, this work assumes a relational model and addresses the problem as a planning problem, focusing on the specificities of the planning algorithm.

4.2 Conceptual Design of ETL Processes

4.2.1 Datastore Representation

Preliminaries. A graph is a pair $G = (V, E)$ of sets, with $V \cap E = \emptyset$, where V is a finite, non-empty set of nodes and $E \subseteq V \times V$ is a set of edges. If the edges are ordered pairs of nodes, then G is called a directed graph. Two nodes $v_1, v_2 \in V$ of G are called adjacent if $(v_1, v_2) \in E$. A node v is incident with an edge e if $v \in e$. The set of all edges with which v is incident, is denoted by $E(v)$. The number of edges at v , i.e. $|E(v)|$, is called the degree of node v , denoted by $d(v)$. In the case of directed graphs, $d^+(v)$ denotes the in-degree of node v , i.e. the number of incoming edges at v , while $d^-(v)$ denotes the out-degree of v , i.e. the number of outgoing edges at v .

A node v will be called *internal node* if $d^-(v) > 0$, otherwise it will be called a *leaf node*. Moreover, it is possible to assign labels to the nodes and/or the edges of a graph. A labeled graph can be defined as $G = (\Sigma_V, \Sigma_E, V, E, l_V, l_E)$, where $\Sigma_V, \Sigma_E, V, E, l_V, l_E$ are, respectively, finite alphabets of the available node and edge labels, the sets of nodes and edges, and two maps describing the labeling of the nodes and edges.

Datastore graph. The schema S_D of a datastore can be viewed as consisting of a set of elements, which can be distinguished in two types: (a) elements that contain the actual data (e.g., the attributes in a relational database or the leaf nodes in an XML document), and (b) elements that contain or refer to other elements, creating the structure of the schema (e.g., relations having attributes or XML nodes having children; foreign keys or IDREFs in XML). In this sense, the schema can be depicted by a directed graph, with nodes corresponding to the elements of the schema and edges representing containment or reference of one element by another. Additionally, labels may be assigned to edges denoting the corresponding cardinality. Therefore, we consider a graph, termed *datastore graph*, representing a datastore schema as an edge-labeled directed graph $G_D = (V_D, E_D, l_E)$ such that:

- Each element e defined in the schema S_D is represented by a node $v_e \in V_D$.
- Each containment relationship is represented by an edge (v_1, v_2) , where v_2 corresponds to the element being contained by the element represented by v_1 .
- Each reference is represented by an edge (v_1, v_2) , where v_1 corresponds to the element containing the reference and v_2 corresponds to the referenced element.
- Each edge is assigned a label of the form $[min, max]$, where min and max denote, respectively, the minimum and maximum cardinality of the reference or containment relationship represented by the edge.

Elements containing the *actual data* are represented by *leaf nodes*. A graph-based representation may be derived for any schema type, based on the above specification. In the cases that no schema is explicitly exposed by a datastore, an appropriate wrapper needs to be constructed first, to provide an interface for retrieving the relevant data based on the underlying structure. Given that the relational schema and XML constitute the most typical models for structured and semi-structured data, respectively, in the following, we elaborate on the construction of the graph representation for these two cases.

Relational schema to graph. A relational schema S_R consists of a set of relations R , each one comprising a set of attributes A_R . The relational schema can be depicted by a directed graph $G_R = (V_R, E_R)$, where the nodes V_R correspond to the set of relations and the (non foreign key) attributes of the schema, while the edges E_R represent the containment of attributes in relations, as well as references between relations, i.e. foreign keys. Additionally, for representing cardinality constraints, labels of the form $[min, max]$ may be assigned to the edges of the graph (with null denoting that no constraint is specified). Thus G_R is essentially an edge-labeled graph $G_R = (V_R, E_R, l_{E_r})$.

XML schema to graph. XML is the most commonly used format for the exchange of semi-structured data [4]. An XML document consists of nested element structures,

Source schema	Target schema
<i>emp</i>	<i>employee</i>
<i>fname</i>	<i>name</i>
<i>lname</i>	<i>project</i>
<i>prj</i>	<i>salary</i>
<i>salary</i>	<i>city</i>
<i>address</i>	<i>street</i>

Table 4.1: *Sample source and target schemas*

starting with a root element. Each element may contain other elements and/or attributes. Often a DTD or an XML Schema is associated to the XML document, defining a set of constraints to which the XML document should conform in order to be considered valid. An XML Schema consists of element declarations and type definitions. Relationships between entities are represented by nesting elements or by references. The constraints *minOccurs* and *maxOccurs* are provided to restrict the cardinality of an element. Thus, an XML Schema S_X may be represented by a directed edge-labeled graph $G_X = (V_X, E_X, l_{E_x})$, where (a) nodes represent *elements*, *attributes*, *complexTypees* and *simpleTypes* (b) edges represent *nesting* or *referencing* of elements, and (c) labels denote the min and max *cardinality* allowed for an element.

Example. At this point, we introduce a reference example that will be used to better motivate and illustrate the described approach. The example comprises a source and target schema regarding data about employees, as shown in Table 4.1. The following assumptions are made regarding the schemas:

- The name of an employee in the source schema is stored in two separate attributes: *fname* and *lname*, holding, respectively, the first and last name, while in the target schema a single attribute is used: *name*. Similarly, the address of an employee is stored in a single attribute in the source schema: *address*, while two separate attributes are used in the target schema: *city* and *street*.
- The source contains information about employees working in at least one project (attribute *prj*), while in the target only employees working in at least two projects are considered.
- The currency used for salaries is Euro for the source schema and U.S. Dollars for the target schema.

These assumptions need to be taken into account when transferring data from the source to the target. \square

4.2.2 Application Ontology Construction

A suitable application ontology is constructed to semantically annotate the datastores. This ontology should provide the ability to describe the semantics of the datastore schemas, so that data transformation and integration can be accomplished with a high degree of automation, based on the use of automated reasoning. The

Web Ontology Language (OWL) [103] is chosen as the language for representing the ontology. OWL, and in particular OWL-DL, is based on Description Logics [15], a decidable fragment of First Order Logic, constituting the most important and commonly used knowledge representation formalism. Therefore, it provides a formal and explicit representation, allowing existing reasoners to be used for automating several tasks of the process, such as checking subsumption relationships between classes. In our approach, only a subset of the features provided by the language is required. These features are summarized in Table 4.2. Specifically, for the ontology creation process it is only needed to create a set of classes and properties, to specify the domain and range of each property, and to organize the classes in an appropriate hierarchy. Therefore, given that several tools exist providing an intuitive and comprehensive Graphical User Interface for the development of ontologies, this task requires only a basic understanding of ontologies and OWL. On the other hand, a precise understanding of the intended semantics of the datastore schemata, and generally the application requirements, is critical in constructing an appropriate ontology and correctly annotating the datastores.

For our purpose, a suitable application ontology should provide the ability for modeling the following types of information: (a) the concepts of the domain, (b) the relationships between those concepts, (c) the attributes characterizing each concept, and (d) the different representation formats and (ranges of) values for each attribute. The concepts of the domain are represented by classes, while the relationships between concepts, as well as the attributes of the concepts are represented by properties. The different types of values for each attribute are also represented by classes appropriately organized in a hierarchy. Finally, as the need for aggregate operations appears frequently, specific elements are included in the ontology, to support the specification of such operations.

Formally, the constructed ontology can be modeled as $O = (C, P, A)$ comprising the following:

- $C = C_C \cup C_T \cup C_G$
- $P = P_P \cup \text{convertsTo, aggregates, groups}$
- A , a set of axioms used to (a) assert subsumption relationships between classes, (b) specify domain and range constraints on properties, (c) specify cardinality constraints, (d) assert disjointness of classes, and (e) define a new class in terms of other classes and properties.

The above notations are explained in more detail in the following. C_C is a set of classes representing the concepts of the domain. C_G is a set of classes describing aggregate operations. Each class in C_G denotes an aggregate function, e.g., *AVG*, *SUM*, *COUNT*, *MAX*. P_P is a set of properties representing attributes of the concepts or relationships between them.

For each property $p \in P_P$, an axiom exists in A specifying a class $c \in C_C$ as the domain of p , thus, associating the property with the concept it describes. If p relates a concept to another concept, represented by class $c' \in C_C$, then c' is specified as the range of p . C_T is the union of a set of classes, $C_T = C_{TP} \cup C_{TF} \cup C_{TR} \cup C_{TG}$, used to represent different kinds of values for a property that corresponds to an attribute of a concept. For each such property p , a class in C_{TP} is declared to be the range of this property. That is, this class represents all the possible (types

Notation	Name	Description
C	class	A group of individuals sharing some properties. Classes represent the concepts of the domain, as well as types of values of their attributes.
$C_1 \equiv C_2$	equivalent	States that two classes are equivalent, i.e. each instance of the one is also instance of the other.
$C_1 \sqsubseteq C_2$	subClassOf	Used to create class hierarchies.
$C_1 \sqcap C_2 = \emptyset$	disjointWith	States that two classes are disjoint, i.e. an instance may not belong to both classes. This is used to prevent the integration of data records from sources with conflicting constraints.
$C_1 \sqcup C_2$	unionOf	Denotes the union of two classes.
$C_1 \sqcap C_2$	intersectionOf	Denotes the intersection of two classes.
P	property	Relate an instance of a class to an instance of another class (<i>ObjectProperty</i>). They represent attributes of concepts and relationships between concepts.
$dom(P)$	domain	Specifies the class(-es) to which the individuals the property applies to, belong.
$range(P)$	range	Specifies the class(-es) to which the individuals being the values of the property, belong.
$\forall P.C$	allValuesFrom	Used to restrict the range of a property, when this property is applied to individuals of a specific class.
$\geq_n P, \leq_n P$	min/max cardinality	Specifies the min/max cardinality of a property w.r.t. to a specific class. It is used to denote the cardinality of attributes and relationships.

Table 4.2: OWL features used in our approach

of) values for this property. Every other class in C_T denoting a specific type of values of p is a subclass of this class. C_{TF} refers to the set of classes used to denote different representation formats, while C_{TR} to the set of classes denoting different (ranges of) values for a property. Classes in C_{TG} represent values that result from aggregate operations. The classes in C_T are organized in an appropriate hierarchy according to the intended semantics. For instance, if a value interval, represented by class $c_r \in C_{TR}$, refers to a particular representation format, represented by class $c_f \in C_{TF}$, then c_r is asserted to be a subclass of c_f . If two types of values are mutually exclusive, then an axiom exists in A stating that the corresponding classes are disjoint.

Property *convertsTo* is used to relate a class $c_1 \in C_{TF}$ to another class $c_2 \in C_{TF}$, indicating that a function exists for transforming data from the representation format represented by c_1 to the representation format represented by c_2 . Property *aggregates* is used to relate a class $c_1 \in C_G$ to another class $c_2 \in C_{TG}$, indicating that the aggregate function represented by c_1 is used to calculate the values represented













	Type	Represents	Symbol
Nodes	concept-node	a class $c \in C_C$	
	aggregation-node	a class $c \in C_G$	
	type-node	a class $c \in C_{TP}$	
	format-node	a class $c \in C_{TF}$	
	range-node	a class $c \in C_{TR}$	
	aggregated-node	a class $c \in C_{TG}$	
Edges	property-edge	a property $p \in P_P$	
	convertsTo-edge	property convertsTo	
	aggregates-edge	property aggregates	
	groups-edge	property groups	
	subclass-edge	class hierarchy	
	disjoint-edge	disjointness of classes	

Table 4.3: Notation used for the ontology graph

by c_2 . Finally, property groups relates a class $c_1 \in C_{TG}$ to another class $c_2 \in C_T$, indicating that the aggregate operation for calculating the values represented by c_1 is applied over the values represented by c_2 .

Ontology graph. After the ontology has been created by the administrator, a corresponding graph representation, called *ontology graph*, is derived. The ontology graph is a directed edge-labeled graph $G_O = (V_O, E_O, l_E)$, where nodes represent classes in the ontology, while edges represent properties. Table 4.3 depicts the different types of nodes and edges in the ontology graph and their corresponding visual notation.

Formally, the construction of the ontology graph is described by the algorithm *O2G* depicted in Table 4.4. The algorithm iterates over the properties defined in the ontology, and creates: (a) one *concept-node* for each class appearing in the domain of a property, (b) one *type-node* for each class appearing in the range of a property, and (c) a *property-edge* connecting the concept-node with the type-node. For each *convertsTo* restriction, a *format-node* is created and a *convertsTo-edge* is added between the node having the restriction and the format-node. Similarly, the same procedure is repeated for restrictions *aggregates* and *groups*. Finally, *subclass-edges* and *disjoint-edges* are created according to subsumption and disjointness of classes, respectively.

Example (cont'd). The application ontology for the reference example presented before is depicted in Figure 4.1. The domain of interest comprises the concept *Employee*, with attributes *hasName*, *worksAt*, *receives*, and *lives*. For each of these attributes, a corresponding property is created in the ontology, as well as a generic class to denote its values: *Name*, *Project*, *Salary*, and *Address*. According to the assumptions made in the example, the classes *Dollars* and *Euros* are used to represent values referring to these two different currencies, while the property

Input: The application ontology O
Output: The ontology graph $G_O = (V_O, E_O, l_E)$

1. **Begin**
2. **Foreach** property p in P_P {
3. $c_1 \leftarrow$ the class being the domain of p ;
4. $c_2 \leftarrow$ the class being the range of p ;
5. **If** ($n(c_1) \notin V_O$) create concept-node $n(c_1)$;
6. **If** ($n(c_2) \notin V_O$) {
7. **If** ($\exists p' : c_2 \sqsubseteq \text{domain}(p')$)
8. create concept-node $n(c_2)$;
9. **Else**
10. create type-node $n(c_2)$;
11. }
12. create property-edge ($n(c_1), n(c_2)$) with label p ;
13. }
14. **Foreach** range restriction on property *convertsTo* {
15. $c_1 \leftarrow$ the class having the restriction;
16. $c_2 \leftarrow$ the class being the filler of the restriction;
17. **If** ($n(c_1) \notin V_O$) create format-node $n(c_1)$;
18. **If** ($n(c_2) \notin V_O$) create format-node $n(c_2)$;
19. create convertsTo-edge ($n(c_1), n(c_2)$);
20. }
21. **Foreach** range restriction on property aggregates {
22. $c_1 \leftarrow$ the class having the restriction;
23. $c_2 \leftarrow$ the class being the filler of the restriction;
24. **If** ($n(c_1) \notin V_O$) create aggregation-node $n(c_1)$;
25. create aggregated-node $n(c_2)$;
26. create aggregates-edge ($n(c_1), n(c_2)$);
27. }
28. **Foreach** range restriction on property groups {
29. $c_1 \leftarrow$ the class having the restriction;
30. $c_2 \leftarrow$ the class being the filler of the restriction;
31. create groups-edge ($n(c_1), n(c_2)$);
32. }
33. **Foreach** subclass relation: $c_1 \sqsubseteq c_2$ {
34. **If** ($n(c_1) \notin V_O$) create range-node $n(c_1)$;
35. **If** ($n(c_2) \notin V_O$) create range-node $n(c_2)$;
36. create subclass-edge ($n(c_2), n(c_1)$);
37. }
38. **Foreach** disjointness axiom: $c_1 \sqcap c_2 = \emptyset$ {
39. create disjoint-edge ($n(c_1), n(c_2)$);
40. create disjoint-edge ($n(c_2), n(c_1)$);
41. }
42. **End.**

Table 4.4: Algorithm for graph representation of the ontology

convertsTo is used to indicate the ability to convert from one currency to the other. Moreover, the class *AboveN* is created to represent salaries exceeding the specified limit of N euros. □

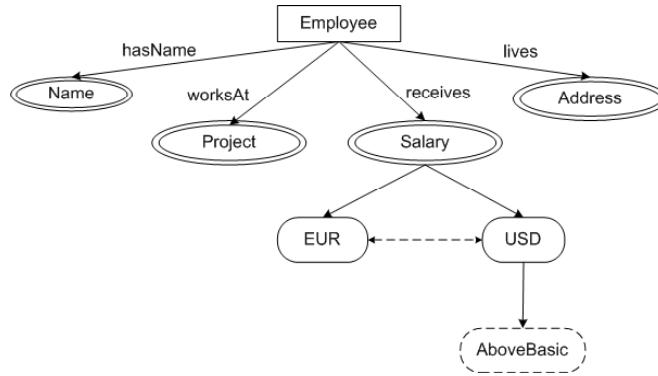


Figure 4.1: *The ontology graph for the reference example*

4.2.3 Semantic Annotation of Datastores

The semantic annotation of the datastores constitutes in mapping the datastore schemas to the application ontology. Based on these mappings, a set of defined classes representing the datastores is generated and added to the ontology. This procedure reveals the semantics of the elements contained in the involved schemas, allowing a reasoning process to be applied for identifying necessary operations for transforming data from the sources to the target.

The annotation of each datastore is accomplished by defining mappings between the corresponding datastore graph G_S and the ontology graph G_O . These mappings are specified by the administrator and are pairs of the form (v_S, v_O) , where v_S and v_O denote nodes of G_S and G_O , respectively. Specifically, each internal node of G_S may be mapped to one concept-node of G_O . It is not required for all internal nodes to be mapped, since some of these nodes exist for structural purposes and do not represent a concept of the domain being modeled. A leaf node of G_S may be mapped to one or more nodes of G_O of the following types: *type-node*, *format-node*, *range-node* or *aggregated-node*. In this way, the elements of the datastore schema containing the actual data are semantically annotated, e.g., specifying the representation format or value ranges, for the underlying data. It is possible to specify more than one mappings for a leaf node, e.g., in the case that the datastore allows for more than one representation format to be used for the values of the corresponding property. If no mapping is specified for a leaf node, it means that the corresponding attribute is of no interest for the specific application and does not participate in the integration process, i.e. the node is ignored from any further process.

Note that an additional advantage of using a graph-based representation for both the datastores and ontology is the fact that graphical tools can be developed to allow for a visual representation and specification of these mappings, e.g., by using a “drag-and-drop” technique between graph nodes to create a mapping, thus facilitating and speeding up the mapping process. Furthermore, it is possible to incorporate schema matching techniques to automatically detect candidate mappings [129]. The specified mappings can be represented as labels assigned to the nodes of the source graph for which a mapping is specified.

Example (cont’d). The appropriate mappings for the datastores of the reference example are presented in Figure 4.2. \square

After the appropriate mappings between the datastore graphs and the ontology

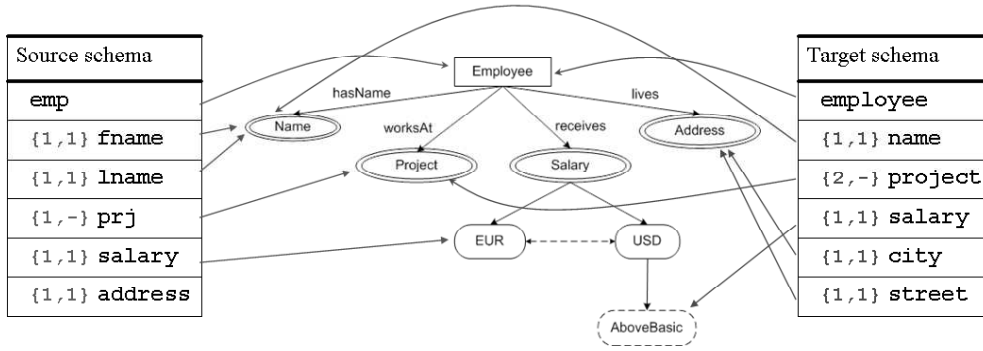


Figure 4.2: Semantic annotation of the datastores

graph have been considered, a set of definitions are created to semantically describe each datastore. In particular, for each labeled internal node n of the datastore graph, a defined class $c(n)$ is created and added to the ontology. The label of node n is then updated to indicate this defined class. The class $c(n)$ is a subclass of the class indicated by the label of node n , containing also a set of restrictions derived from the specified mappings of the neighbor labeled nodes of n . A *neighbor labeled node* of n is each node n' for which the following conditions hold:

- n' is labeled; i.e. it is mapped to one or more nodes of the ontology graph,
- there is a path p in the source graph from mode n to node n' , and
- p contains no other labeled nodes, except n and n' .

The intuition for using the notion of neighbor labeled nodes lies in the fact that nodes without associated mappings are of no interest, and therefore need to be ignored. Thus, each labeled internal node is linked to a set of other, internal or leaf, labeled nodes. The cardinality for each one of these links is computed by multiplying the corresponding cardinalities along the path connecting the two nodes. Note that if the *min* (*max*) cardinality of an edge in the path equals to *null*, then the final *min* (*max*) cardinality is also *null*. The process for deriving the defined class $c(n)$ for an internal labeled node n is formally specified by the algorithm *CDI* shown in Table 4.5.

Example (cont'd). The defined classes derived by the application of the algorithm *CDI* to the reference example are the following:

$$\begin{aligned}
S.Emp &\equiv Employee \sqcap \forall hasName.Name \sqcap =_1 hasName \sqcap \\
&\quad \forall worksAt.Project \sqcap \geq_1 worksAt \sqcap \forall receives.EUR \sqcap \\
&\quad =_1 receives \sqcap \forall lives.Address \sqcap =_1 lives \\
T.Employee &\equiv Employee \sqcap \forall hasName.Name \sqcap =_1 hasName \sqcap \\
&\quad \forall worksAt.Project \sqcap \geq_2 worksAt \sqcap \forall receives.AboveBasic \\
&\quad \sqcap =_1 receives \sqcap \forall lives.Address \sqcap =_1 lives \quad \square
\end{aligned}$$

4.2.4 Identification of ETL Operations

The semantic annotation of the datastores is used, in conjunction with the application ontology, to infer correspondences and conflicts among them and propose a set

Algorithm Create the Definition for an Internal labeled node (CDI)

Input: An internal labeled node n of the datastore graph G_S and the application ontology O

Output: A defined class $c(n)$ representing node n

1. **Begin**
2. **If** ($c(n) \notin O$) create class $c(n)$;
3. $n_O(c_O) \leftarrow$ the ontology graph node indicated by the label of n ;
4. set $c(n)$ subclass of c_O : $c(n) \sqsubseteq c_O$;
5. **Foreach** neighbor labeled node n' of n {
6. $min, max \leftarrow$ cardinalities for $path(n, n')$;
7. **Foreach** label of n' {
8. $c' \leftarrow$ the class indicated by the label;
9. $p \leftarrow$ the property relating class c_O to c' or to any superclass of c' ;
10. **If** ($min \neq null$)
11. $c(n) \leftarrow$ add min cardinality restriction: $\geq_{min} p$;
12. **If** ($max \neq null$)
13. $c(n) \leftarrow$ add max cardinality restriction: $\leq_{max} p$;
14. **If** (n' is internal node) {
15. **If** ($c(n') \notin O$) create class $c(n')$;
16. $c(n) \leftarrow$ add restriction: $\forall p.c(n')$;
17. }
18. **Else**
19. $c(n) \leftarrow$ add restriction: $\forall p.c'$;
20. }
21. }
22. **End.**

Table 4.5: Algorithm for creating the definition for an internal labeled node of the datastore graph

of conceptual operations for transforming recordsets from the source datastores to the target datastore.

Table 4.6 presents generic types of conceptual operations that are typically encountered in an ETL scenario. Note that these abstract operations constitute core operators in practically every frequently used ETL transformation. For example, a *RETRIEVE* operation may vary from an SQL query to an XPath query or even a Web Service call, a *FILTER* operation may range from a simple comparison of values to a complex regular expression, and a *CONVERT* operation resembles a generic function application operation which may be implemented as e.g., a simple conversion between different units of measurements or be carried out by a composite business process. This work does not anticipate the formal determination of the functionality of each individual ETL transformation; rather, it aims at the identification of a generic conceptual transformation, whose functionality will be determined later by the administrator through a template library similar to the one proposed in [165].

Given the constructed application ontology and the set of defined classes that semantically describe the datastores, a reasoning process is applied to identify and propose a set of generic transformations for designing the respective conceptual ETL scenario. The whole procedure may be broken down to two main objectives: (a) identifying the relevant data sources, and more precisely the relevant elements of

Operation	Description
RETRIEVE(n)	Retrieves records from the underlying provider node n
EXTRACT(c)	Extracts from incoming records the part denoted by c
MERGE	Merges records from two or more provider nodes
FILTER(c)	Filters incoming records, allowing only records with values of the template type specified by c
CONVERT(c_1, c_2)	Converts incoming records from the template type denoted by c_1 to the template type denoted by c_2
AGGR(f_g, g_1, \dots, g_n)	Aggregates incoming records over the attributes g_1, \dots, g_n , applying the aggregate function denoted by f_g
MINCARD(p, min)	Filters out incoming records having cardinality less than min on property p
MAXCARD(p, max)	Filters out incoming records having cardinality more than max on property p
UNION	Unites records from two or more sources
DD	Detects duplicate values on the incoming records
JOIN	Joins incoming records
STORE	Stores incoming records to the target datastore

Table 4.6: *Generic types of conceptual transformations frequently used in an ETL process*

these data sources, for populating a specific element of the target; and (b) proposing appropriate generic conceptual transformations of data stemming from the identified source elements, so that the constraints and requirements of the target are satisfied.

Selecting relevant sources. Given a source datastore and an element of the target datastore, the goal is to identify the relevant element(s) of the source schema from which data should be extracted in order to populate the target element. That is, given a labeled node n_T of the target graph G_T , we aim to identify which labeled nodes from the source graph G_S can be used as providers for n_T . This is achieved by reasoning on the mappings of the graph nodes to the common application ontology. Specifically, for a source node n_S to be identified as provider for a target node n_T , the following conditions must hold for their respective classes $c(n_S), c(n_T)$:

- $c(n_S)$ and $c(n_T)$ have a common superclass, and
- $c(n_S)$ and $c(n_T)$ are not disjoint

The first condition ensures that the two nodes are semantically related, i.e. the contained data records refer to the same concept of the domain. The second condition ensures that the constraints of one node do not contradict the constraints of the other.

The algorithm *PNS* that formally describes a method for the identification of the provider nodes is shown in Table 4.7. Based on the semantic annotation of the source and target nodes, the reasoner parses the ontology and infers whether the two above specified conditions are met or not.

Algorithm Provider Node Selection (PNS)

Input: A labeled node n_T of a target datastore graph G_T , a source datastore graph G_S , and the application ontology O

Output: A list L containing the provider nodes for n_T

1. **Begin**
 2. **Foreach** label of n_T {
 3. $c_T \leftarrow$ the class indicated by the label;
 4. **Foreach** labeled node $n_S \in G_S$ {
 5. **Foreach** label of n_S {
 6. $c_S \leftarrow$ the class indicated by the label;
 7. **If** $((\exists c_0 \in O : c_S \sqsubseteq c_0 \wedge c_T \sqsubseteq c_0) \wedge (c_S \sqcap c_T \neq \emptyset))$
 8. $L \leftarrow$ add n_S ;
 9. }
 10. }
 11. }
 12. **End.**
-

Table 4.7: *Algorithm for provider node selection*

In the case that no provider node is identified, two cases are distinguished: (a) a source node with a common superclass was found, but the corresponding classes were disjoint; this means that the constraints of the source contradict those of the target, or (b) no node was found having a common superclass with the target node; this means that the source does not contain information regarding the concept in question. In such cases, an appropriate log entry is generated and it is up to the administrator to decide whether this particular source should be discarded or to provide by an extra operation the missing information; e.g., by means of default values.

After the provider nodes are identified, a *RETRIEVE*(n) operation is required for each provider node n to retrieve the corresponding data records. If more than one provider nodes are identified, then the data records of these nodes need to be merged, by means of a *MERGE* operation. If a provider node has more than one labels, meaning that the data records contain information regarding more than one entities or attributes, then the data records from this node need to be split appropriately, using an *EXTRACT*(c) operation, so that the appropriate portion of the data record is selected.

Data transformation. At this stage, the data records extracted from the source need to be appropriately filtered, transformed and/or aggregated, so as to satisfy the target constraints and requirements. In the case of two labeled leaf nodes, a source n_S and a target n_T , the required transformations are identified based on the relative position of their corresponding classes, c_S and c_T , in the class hierarchy defined in the ontology. Formally, this process is described by the algorithm *DTL* in Table 4.8. The algorithm works as described in the following. Given a source class c_S and a target class c_T , if $c_S \sqsubseteq c_T$ holds, then no transformations are required. Otherwise, if $c_T \sqsubset c_S$ holds, only a subset of the source records are compatible with the target constraints. Therefore, an appropriate filtering of the source records is required. In the particular case that the target class represents an aggregated type, then an aggregate operation is required instead of filtering. In different case, the

Algorithm Derive Transformations from Labeled leaf nodes (DTL)

Input: A target labeled leaf node n_T , a provider labeled leaf node n_S , and the application ontology O

Output: A list L containing transformation operations between the two nodes

1. **Begin**
 2. $c_S \leftarrow$ class corresponding to n_S ;
 3. $c_T \leftarrow$ class corresponding to n_T ;
 4. **If** ($c_S \sqsubseteq c_T$)
 5. $L \leftarrow \emptyset$;
 6. **Else** {
 7. **If** ($c_T \sqsubset c_S$) {
 8. **Foreach** class c_i in the path from c_S to T {
 9. **If** ($\exists c_g : \text{aggregates}(c_g, c_i)$) {
 10. $c' \leftarrow$ one or more classes c such that: $\text{groups}(c_i, c)$;
 11. $L \leftarrow$ add AGGREGATE(c_g, c');
 12. }
 13. **Else** { $L \leftarrow$ add FILTER(c_i);}
 14. }
 15. }
 16. **Else** {
 17. **If** ($\exists c_1, c_2 : c_S \sqsubseteq c_1 \wedge c_T \sqsubseteq c_2 \wedge \text{convertsTo}(c_1, c_2)$) {
 18. $L \leftarrow$ add CONVERT(c_1, c_2);
 19. $c_S \leftarrow c_2$;
 20. repeat lines 8-15;
 21. }
 22. **Else** {
 23. $c_S \leftarrow$ the class c_0 such that: $c_S \sqsubseteq c_0 \wedge c_T \sqsubseteq c_0$;
 25. repeat lines 8-15;
 26. }
 27. }
 28. }
 29. **End.**
-

Table 4.8: Algorithm for deriving transformations between two labeled leaf nodes

reasoner searches for a superclass of c_S that can be converted to a superclass of c_T or for a common superclass of c_S and c_T , and then, it repeats the previous step.

In the case of internal nodes, the transformation is based again on the neighboring labeled nodes. That is, the transformations for the leaf neighboring labeled nodes are first identified using the previously described mechanism, and then the resulting recordsets are combined to form the final flow of data from the source to the target store. Recordsets from nodes, whose corresponding classes are related by a property, are combined by means of a *JOIN* operation, while recordsets from nodes, whose corresponding classes have a common superclass, are combined by means of a *UNION* operation, followed by a *DD* operation. For neighboring labeled nodes that are themselves internal nodes, the process proceeds recursively.

Furthermore, for internal nodes, minimum and maximum cardinality filters, i.e. *MINCARD* and *MAXCARD* operations, may also be required, based on the cardinality constraints specified for the corresponding links. These are automatically

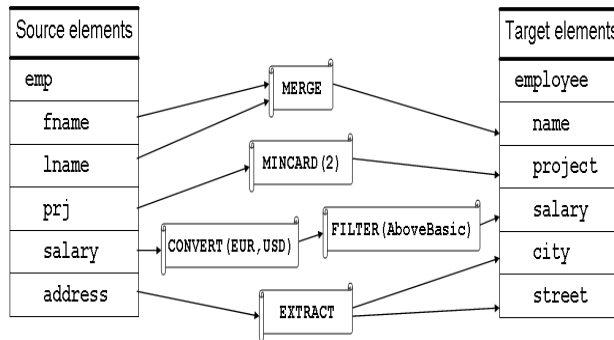


Figure 4.3: Identified transformations for the reference example

identified as follows. If the defined class for the target node contains a min cardinality restriction of the form $\geq_{min} p$ and the defined class for the source node either does not contain a min cardinality restriction on this property or contains one with $min' < min$, then a $MINCARD(p, min)$ operation is added (similarly for $MAXCARD$ operations).

The procedure terminates with a *STORE* operation that is added at the end of the workflow to represent the loading of the transformed data to the target datastore. This operation resembles any loading technique such as bulk loading.

In a following step, the transformations produced should be ordered so that they may be successfully populated; i.e. each transformation should be placed in the ETL design as long as all its providers already exist in the design. For this procedure, we adopt the approach presented in [142].

Example (cont'd). Figure 4.3 shows the transformations proposed for the reference example, according to the algorithm *DTL*. \square

4.2.5 Generation of Reports

The process of designing an ETL scenario involves a diverse group of participants, ranging from database administrators and software developers to business managers. However, existing models for the conceptual design and representation of ETL processes typically require some level of technical knowledge to understand and use them, thus causing inconveniences in the communication of the involved parties. To overcome this deficiency, we leverage our ontology-based approach for the conceptual design of ETL processes and describe a comprehensive and customizable mechanism for generating Natural Language-style reports for the datastores and the ETL activities.

Notice that, even though there have been efforts towards the generation of textual representations from ontologies [25, 26, 171, 172], these approaches constitute general-purpose ontology verbalizers; they are agnostic of the types of classes, properties and operations used in our case for semantically describing the datastores and inferring correspondences among them. Consequently, although it would be possible, in principle, to use one of these approaches, the resulting output would be too verbose and redundant, failing to focus on the aspects of interest from the perspective of the ETL design task. It would also be more difficult to customize the output and achieve different levels of granularity according to particular information needs.

Instead, we present a mechanism for the derivation of reports describing the

outcomes of the ETL design phase, namely the datastore annotations, and the generated ETL scenario. To provide a comprehensive, flexible, and easily customizable mechanism for generating textual representations, a template-based technique is followed. A template-based system is a natural language generating (NLG) system that maps non-linguistic input directly (i.e., without intermediate representations) to the linguistic surface structure. The use of templates also has the benefit of providing a comprehensive and intuitive way to customize the produced output, without requiring that the DW designer should understand complex NL processing techniques.

Template language. The translation process is realized by suitable templates, which are constructed composing elements provided by our template language. This is a typical template language, supporting constructs such as variables and directives, but also extended with built-in functions and macros suited to the ETL design task. Notice that the generated text may also contain HTML tags, so that highly formatted output can be produced. Next, we describe the aforementioned constructs in more detail.

Variables. A template may contain variables, which are denoted by their name preceded by the symbol \$. When the template is instantiated, the template engine that processes it replaces each variable with a corresponding, provided, value.

Directives. A set of typical directives is supported, allowing for a high degree of flexibility in specifying templates. Specifically, the directives #set, #if / #elseif / #else, and #foreach are provided to set the value of a parameter, allow conditional output and iterate through a list of objects, respectively. The standard arithmetic, logic, and comparison operators are also supported.

Functions. The template language supports the usual arithmetic, date, and string manipulation functions. In addition, we provide the template designer with a set of built-in functions specifically tailored for the ETL environment, as shown in Table 4.9.

Function	Output
HEAD(D)	The primitive class appearing in the annotation D
PARSE_ANNOT(D)	The list of restrictions R appearing in the annotation D
PARSE_RES(R)	The type of restriction R
TEXT(X)	The textual description of entity X
RANGE(P)	The class being the range of property P
INTERVAL(C)	The lower/upper bounds of the value interval specified by class $C \in C_{TR}$
ENUM(C)	The list of the members of class $C \in C_{TE}$
AGGR_FUNC(C)	The class related to $C \in C_{TG}$ via the property “aggregates”
AGGR_ATTR(C)	The list of classes related to $C \in C_{TG}$ via property “groups”
PARSE_FLOW(W)	The list of operations constituting an ETL flow W
PARSE_OP(F)	The type of operation F
PARAMS(F)	The list of parameters of an ETL operation F
SIZE(L)	The size of the list L

Table 4.9: A set of provided built-in functions

Macros. Macros allow simpler templates to be reused and/or combined to define more complex ones. Thus, they significantly facilitate the creation of templates. For instance, a typical use for macros is to specify how the elements of an array

should be rendered. In our work, macros are also defined for the different types of restrictions, as well as for the different types of ETL operators, to specify the textual description of these elements. The designer may customize and extend the translation mechanism, by modifying these macros or defining new ones. An example general-purpose macro that renders the elements of a list follows:

```
Macro: LIST(L)
#set ( $size = #SIZE($L) )
#set ( $counter = 0 )
#foreach( $item in $L )
  #if ( $counter == 0 ) #TEXT($item)
  #elseif ( $counter == $size-1) and #TEXT($item)
  #else , #TEXT($item)
  #set ( $counter = $counter + 1 )
#end
#end
```

In the result, a comma follows each list item, but the word “and” comes before the last one. Observe that standard programming knowledge is enough for the macro creation. Apart from such macros that concern generic functionalities, we do provide as well default macros for translating data store annotations and conceptual ETL operations, as shown in Tables 4.10 and 4.11, respectively. Having constructed appropriate templates based on macros ensures the reusability and extensibility of the reporting mechanism.

Macros for Class Definitions	
HEADER(S,C)	Each tuple in \$S contains information about a \$#TEXT(C). It:
EXACT_CARD(=nP)	#TEXT(\$P) exactly \$n #TEXT(#RANGE(\$P)).
MIN_CARD(≥nP)	#TEXT(\$P) at least \$n #TEXT(#RANGE(\$P)).
MAX_CARD(≤nP)	#TEXT(\$P) at most \$n #TEXT(#RANGE(\$P)).
RANGE_RES(∀P.C)	#TEXT(\$P) #TEXT(#RANGE(\$P)) in #INTERVAL(\$C).
ENUM_RES(∀P.C)	#TEXT(\$P) #TEXT(#RANGE(\$P)) one of: #LIST(#ENUM(\$C)).
FORMAT_RES(∀P.C)	#TEXT(\$P) #TEXT(#RANGE(\$P)) of type #TEXT(\$C).
AGGR_RES(∀P.C)	contains the #TEXT(#AGGR_FUNC(\$C)) #TEXT(#RANGE(\$P)) per #LIST(#AGGR_ATTR(\$C)).

Table 4.10: A set of provided built-in macros for datastore annotations

Template instantiation. According to the reporting needs, an appropriate template is created, or an existing one is chosen, and a narrative is produced automatically out of it. This procedure is performed by the template engine, and it requires that the template is used in synergy either with the formal expression annotating a data store or with (a part of) the ETL specification. The template is instantiated by expanding any contained macros, evaluating any contained functions and directives, and assigning concrete values to its parameters.

Report production. The format of a report is highly dependent on each application. In our experience and understanding, for the early phases of a DW design project, it is advisable to have reports expressing the parts of the ETL process by simple means. For that reason, we favor the presentation of the results as bullet

Macros for ETL Operations	
RETRIEVE(V,C)	Retrieve #TEXT(\$C) from \$V.
EXTRACT(V,C)	Extract #LIST(\$C) from \$V.
MERGE(V,C)	Compose #TEXT(\$C) from #LIST(\$V).
RANGE_FILTER(P,C)	Select #TEXT(\$P) #TEXT(#RANGE(\$P)) in #INTERVAL(\$C).
VALUE_FILTER(P,C)	Select #TEXT(\$P) #TEXT(#RANGE(\$P)) one of: #LIST(#ENUM(\$C)).
CONVERT(P,C ₁ ,C ₂)	Convert #TEXT(#RANGE(\$P)) from #TEXT(\$C ₁) to #TEXT(\$C ₂).
AGGR(P,C _f ,C _g)	Calculate #TEXT(\$C _f) #TEXT(#RANGE(\$P)) per #LIST(\$C _g).
CARD(P,min,max)	Select tuples having #TEXT(\$P) at least \$min and at most \$max #TEXT(#RANGE(\$P)).
DD(P)	Remove duplicate tuples for #TEXT(#RANGE(\$P)).
JOIN(C)	Join tuples from #LIST(\$C).
STORE(C,V)	Store #TEXT(\$C) to #LIST(\$V).

Table 4.11: A set of provided built-in macros for generic ETL operations

lists. However, the format of a report is defined by the creator of the respective template. Our mechanism is generic enough to support fairly rich representation formats. Next, we present a series of indicative templates to illustrate the use of the proposed mechanism and demonstrate its usefulness and flexibility.

Case 1. A template for rendering the annotation D of a datastore S is structured as follows:

```

Template: PRINT_ANNOT(S, D)
#set ( $head = HEAD($D) )
#HEADER( $S, $head )
#set ( $res_list = #PARSE_ANNOT($D) )
#foreach( $res in $res_list)
  #if ( #PARSE_RES( $res ) == "EXACT_CARD" )
    #EXACT_CARD( $res )
  #elseif ( #PARSE_RES( $res ) == "MIN_CARD" )
    #MIN_CARD( $res )
  ... // calls to macros for other types of restrictions
#end
#end

```

As example, consider a source and a target datastore, denoted by DS_SPart and $DW_SupPart$, respectively, containing information about parts provided by several suppliers. We make the following assumptions. Each part and supplier is identified by a unique ID. The stores contain information about the date the parts were purchased and their price. $DW_SupPart$ keeps records only of parts purchased after the year 2000 and having from 2 to 5 suppliers. Prices in DS_SPart and $DW_SupPart$ are recorded in Dollars and Euros, respectively. DS_SPart records the quantity of stored parts for each individual storage location, while $DW_SupPart$ records the total quantity over all locations. Lastly, DS_SPart contains information for parts in the categories software, hardware or accessories (S/H/A), while $DW_SupPart$ contains only software or hardware parts (S/H).

Using an appropriate domain ontology, the the two datastores could be semantically annotated by expressions such as those shown below:

$$DS_SPart \equiv SuppliedPart \sqcap =_1 hasPartID \sqcap \geq_1 SuppliedBy \\ \sqcap \forall hasPrice.Dollars \sqcap \forall belongsTo.software, hardware, accessories$$

$$DW_SupPart \equiv SuppliedPart \sqcap =_1 hasPartID \sqcap \geq_2 suppliedBy \\ \sqcap \leq_5 suppliedBy \sqcap \forall purchasedAtYear.LaterThan2000 \\ \sqcap \forall hasPrice.Euros \sqcap \forall belongsTo.software, hardware \\ \sqcap \forall hasQuantity.TotalQuantity$$

Given these annotations, the above template produces the following reports:

Each tuple in DS_SPart contains information about a supplied part. It:

- has exactly 1 part id
- is supplied by at least 1 supplier
- has price of type dollars
- belongs to category one of: software, hardware, accessories

Each tuple in DW_SupPart contains information about a supplied part. It:

- has exactly 1 part id
- is supplied by at least 2 and at most 5 supplier
- purchased at year in [2000,-]
- has price of type euros
- contains the total quantity per part id, supplier, day, month, year, price, category
- belongs to category one of: software, hardware

Case 2. A template for generating textual representations of generic ETL operations is similarly specified, as shown below:

```
Template: PRINT_ETL(W)
Transformations from $source to $target :
#set ( $op_list = #PARSE_FLOW($W) )
#foreach( $op in $op_list)
  #if ( #PARSE_OP( $op ) == "RETRIEVE" )
    #RETRIEVE( #PARAMS( $op ) )
  #elseif ( #PARSE_OP( $op ) == "EXTRACT" )
    #EXTRACT( #PARAMS( $op ) )
  ... // calls to macros for other types of operations
#end
#end
```

Assume the following ETL operations for transforming data from DS_SPart to DW_SupPart:

```
MERGE([model,number],PartID)
RETRIEVE(sCode,Supplier)
RETRIEVE(value,Price)
```

```

EXTRACT(date, [Day, Month, Year])
RETRIEVE(amount, Quantity)
RETRIEVE(category, Category)
CARD(suppliedBy, 2, 5)
FILTER(purchasedAtYear, LaterThan2000)
CONVERT(hasPrice, Dollars, Euros)
AGGREGATE(hasQuantity, SUM, [PartID, Supplier, Day, Month, Year, Price, Category])
FILTER(belongsTo, {software, hardware})
STORE(PartID, partID)
STORE(Supplier, supCode)
STORE(Day, day)
STORE(Month, month)
STORE(Year, year)
STORE(Price, price)
STORE(Quantity, quantity)
STORE(Category, category)

```

Then, this template produces the report shown below:

Transformations from DS_SPart to DW_SupPart:

- Compose part id from SPart.model, SPart.number
- Retrieve supplier from SPart.sCode
- Retrieve price from SPart.value
- Extract day, month, year from SPart.date
- Retrieve quantity from SPart.amount
- Retrieve category from SPart.category
- Select tuples supplied by at least 2 and at most 5 supplier
- Select purchase year in [2000,-]
- Convert price from dollars to euros
- Calculate total quantity per part id, supplier, day, month, year, price, category
- Select belongs to category one of: software, hardware
- Store part id to SupPart.partID
- Store supplier to SupPart.supCode
- Store day to SupPart.day
- Store month to SupPart.month
- Store year to SupPart.year
- Store price to SupPart.price
- Store quantity to SupPart.quantity
- Store category to SupPart.category

Case 3. Apart from rendering a complete description of the ETL flow, a template can focus on information regarding specific aspects. For instance, the following template prints the total number of operations in a given flow, as well as the number of CONVERT operations.

```

Template: PRINT_ETL_STATS(W)
#set ( $n1 = 0 )
#set ( $n2 = 0 )
#set ( $op_list = #PARSE_FLOW($W) )
#foreach( $op in $op_list)
    #set ( $n1 = $n1 + 1 )

```



```

    #if ( #PARSE_OP($op) == "CONVERT'' )
        #set ( $n2 = $n2 + 1 )
    #end
#end
This ETL flow contains a total of $n1 operations.
$n2 of these are CONVERT operations.

```

As shown from the previous case, the resulting output may often contain repeated information. Even though this does not necessarily constitute a problem, since the output is often presented in a tabular form or as a bulleted list, in other cases a more concise representation is preferable. This issue is addressed by grouping together related pieces of information, a process that is commonly referred to as sentence aggregation [45, 132] in Natural Language Processing. In our case, aggregation can be achieved based on the following criteria: (a) grouping together restrictions of the same type, (b) grouping together restrictions on the same property, (c) grouping together operations of the same type, and (d) grouping together operations on the same attribute.

Although this can be done programmatically by specifying the appropriate conditions in the corresponding templates, to reduce coding effort and simplify the templates, we overload the built-in functions `PARSE_ANNOT(D)` and `PARSE_FLOW(W)` providing two new variations for each one: `PARSE_ANNOT(D,R)`, `PARSE_ANNOT(D,P)`, `PARSE_FLOW(W,F)`, `PARSE_FLOW(W,P)`. The first returns only restrictions of type R, while the second returns only restrictions applied on property P. Similarly, the third and the fourth return only operations of a given type F and on a specific property P, respectively.

Hence, using in addition the function `SIZE(L)` shown in Table 4.9, the previous template can be significantly simplified as follows.

```

Template: PRINT_ETL_STATS_SHORT(W)
This ETL flow contains a total of
#SIZE(#PARSE_FLOW($W)) operations.
#SIZE(#PARSE_FLOW($W,"CONVERT")) of these are CONVERT operations.

```

Finally, other characteristic cases are, for instance, to verbalize the first n operations of the workflow or the operations concerning a specific property. The latter case is especially useful to track the transformations occurring on a specific attribute throughout the workflow. Another practical case is to list groups of order-equivalent transformations to help the administrator to design the execution order of an ETL workflow [142].

4.3 ETL Design through Graph Transformations

4.3.1 General Framework

In the previous section we have proposed an approach for using ontologies to facilitate the conceptual design of ETL processes. This section further builds on this idea, exploring an alternative direction to the problem, based on the theory and tools for graph transformations. This allows for two main advantages. First, in the former approach, customization and extensibility, although possible, are not very easy to

accomplish, as the process of deriving the ETL transformations is tightly coupled to the ontology reasoner. Instead, in the current approach there is a clear separation between the graph transformation rules, which are responsible for creating the ETL design, and the graph transformation engine. Second, in the former approach, the whole ETL flow between a given pair of source and target node sets is produced in a single run. On the contrary, the current approach provides, in addition to that, the ability to proceed with the ETL design in an interactive, step-by-step mode. For example, the designer can select a set of source nodes to begin with, select a set of rules and execute them for a number of steps, observe and possibly modify the result, and continue the execution; this exploratory behavior is very often required, since the design of an ETL process is a semi-automatic task.

First, we describe the representation model used for the source and target data stores, as well as for the domain ontology and the ETL process. Then, we state the problem of deriving the design of an ETL process at the conceptual level, via a series of graph transformations, based on the semantic knowledge conveyed by the domain ontology attached to the source and target schemata.

In particular, our approach is based on appropriate manipulation of a graph that contains all the involved information, namely the *datastore schemata*, the *domain ontology*, the *semantic annotations*, and the *ETL operations*. These modules are described in the following.

Datastore subgraph. Traditional ETL design tools employ a relational model as an interface to the data repositories. The relational model has widespread adoption and an RDBMS constitutes the typical solution for storing an organization's operational data. Nevertheless, the increasingly important role of the Web in e-commerce, and business transactions in general, has led to semi-structured data playing a progressively more important role in this context. The adoption of XML as a standard for allowing interoperability requires that data crossing the borders of the organization is structured in XML format. For instance, Web services, which enable enterprises to cooperate by forming dynamic coalitions, often referred to as Virtual Organizations, are described by documents in XML format, and they exchange information in XML format, too. These facts significantly increase the amount of heterogeneity among the data sources, and hence, the complexity of the ETL design task.

To abstract from a particular data model, we employ a generic, graph-based representation, that can effectively capture both structured and semi-structured data. In particular, we model a datastore as a directed graph, i.e., $G = (V, E)$, where V is a set of nodes and $E \subseteq V \times V$ is a set of edges (i.e., ordered pairs of nodes). Graph nodes represent schema elements, whereas graph edges represent containment or reference relationship between those elements.

Note that the same model is used for both source and target datastores. Given that the ETL process may involve multiple source datastores, nodes belonging to different sources are distinguished by using different prefixes in their identifiers.

Ontology subgraph. Our approach is based on the use of an ontology to formally and explicitly specify the semantics of the data contained in the involved datastores. Leveraging the advances in Semantic Web technology, we can use RDF Schema [100, 29] or OWL [103] as the language for the domain ontology. Hence, the knowledge for the domain associated with the application under consideration can be represented by a set of classes and properties, structured in an appropriate hier-

archy. These classes and properties correspond to the concepts of the domain, and the relationships and attributes of these concepts. In addition, for the purpose of ETL design, it is commonly required to express some specific types of relationships, such as different representation formats (e.g., different currencies or different date formats) or different levels of granularity when structuring the information (e.g., representing a particular piece of information either as a single attribute or as a set of attributes). Therefore, apart from the provided `isa` relationship that can be specified among classes (i.e., `<rdfs:subClassOf>`), we assume in addition a set of pre-defined properties, comprising the properties `typeOf` and `partOf`. This set of pre-defined properties can be further extended to accommodate application-specific or domain-specific needs. In the employed representation, classes are represented by nodes, whereas properties by edges.

Datastore annotations. Using the ontology to semantically annotate the datastores is achieved by establishing edges directed from nodes of the datastore subgraph towards corresponding nodes of ontology subgraph.

ETL process subgraph. An ETL process comprises a series of operations that are applied to the source data and transform it appropriately, so as it meets the target specifications. Given the previously described graph-based representation of the source and target datastores, we represent the specification of the ETL process as a set of paths directed from source datastore nodes towards target datastore nodes. The nodes along these paths denote ETL operations; there are also intermediate nodes as we discuss in Section 4.3.3. The edges connecting the nodes indicate the data flow.

In general, it is not straightforward to come up with a close set of well-defined primitive ETL operations. Normally, such effort would result in the set of relational operators extended by a generic function operator. However, this would not be too useful in real world applications that usually comprise a large variety of built-in or user-defined functions. Hence, it is essential to provide a generic and extensible solution that could cover the frequent cases and that could be enriched by additional transformations when needed. Hence, we consider the following set of operations: **Load**, **Filter**, **Convert**, **Extract**, **Split**, **Construct**, **Merge**. These correspond to common operations frequently encountered in ETL processes. A detailed discussion of these operations, as well as their applicability in a given context, are presented in Section 4.3.4.

Problem statement. We consider the problem of ontology-based conceptual design of ETL processes as follows: *starting from an initial graph comprising the source and target datastore subgraphs, the ontology subgraph, and the semantic annotations, produce a final graph that contains also the ETL process subgraph.*

4.3.2 Graph Transformations

Graph transformations were first introduced as a means to address the limitations in the expressiveness of classical approaches to rewriting, especially dealing with non-linear structures [135], and they are widely used in software engineering. The basic idea is to generate a new graph, H , starting from an initial given graph, G , by means of applying a set of *transformation rules*. The graphs G and H , which are also called instance graphs, may be *typed* over a *type graph* TG . A type graph specifies the types

of nodes and edges, and how they are connected. Then, the structure of the instance graphs should conform to the type graph, in order for them to be valid. That is, the relationship between an instance graph and a corresponding type graph is similar to that between an XML document and its associated XML Schema. Additionally, the graphs may be *attributed*, i.e., graph nodes and edges may have attributes. An attribute has a name and a type, specifying the values that can be assigned to it. Graph objects of the same type share their attribute declarations. Transformations of the original graph to a new graph are specified by transformation rules.

A *graph transformation rule*, denoted by $p : L \rightarrow R$ consists of a name p and two instance graphs L and R , which are also typed over TG and represent, respectively, the *pre-conditions* and the *post-conditions* of the rule. This means that (a) the rule is triggered whenever a structure matching L is found, and (b) the execution of the rule results in replacing the occurrence of the left-hand side (LHS) of the rule, L , with the right-hand side (RHS), R . Therefore, a *graph transformation* from a given graph G to a new graph H is denoted by $G \xrightarrow{p(o)} H$, and it is performed in three steps:

- i. Find an occurrence o of the left-hand side L in the given graph G .
- ii. Delete from G all the nodes and edges matched by $L \setminus R$ (making sure that the remaining structure is a graph, i.e., no edges are left dangling.)
- iii. Glue to the remaining part a copy of $R \setminus L$.

Apart from pre-conditions, i.e., patterns whose occurrence triggers the execution of the rule, a rule may also have *negative application conditions* (NACs), i.e., patterns whose occurrence prevents its execution.

A *graph transformation sequence* consists of zero or more graph transformations. Notice that two kinds of non-determinism may occur. First, several rules may be applicable. Second, given a certain rule, several matches may be possible. This issue can be addressed with different techniques, such as organizing rules in *layers*, setting rule priorities, and/or assuming human intervention in choosing the rule to apply or the match to consider.

4.3.3 The Type Graph

As discussed in Section 4.3.1, the design of the ETL process is built in a step-by-step manner through a series of graph transformations. Essential to this is the role of the ontology, which determines the context (i.e., the semantics) at each transformation step, thus determining which ETL operations are applicable (and in what order). The selected ETL operations are represented as additional nodes and edges forming paths (flows) that lead from the nodes of the source subgraph to the nodes of the target subgraph.

The process of addressing this problem by means of graph transformations is outlined in the following. We consider as starting point a graph comprising three subgraphs, namely the source, the target, and the ontology subgraphs. The main goal is then to define an appropriate set of rules, determining where, when, and how a flow of operations from a source to a target node can be created. Essentially, each rule is responsible for inserting an operator in the ETL flow. (Additionally, as we discuss at a later point, some rules aim at replacing or removing operators from the

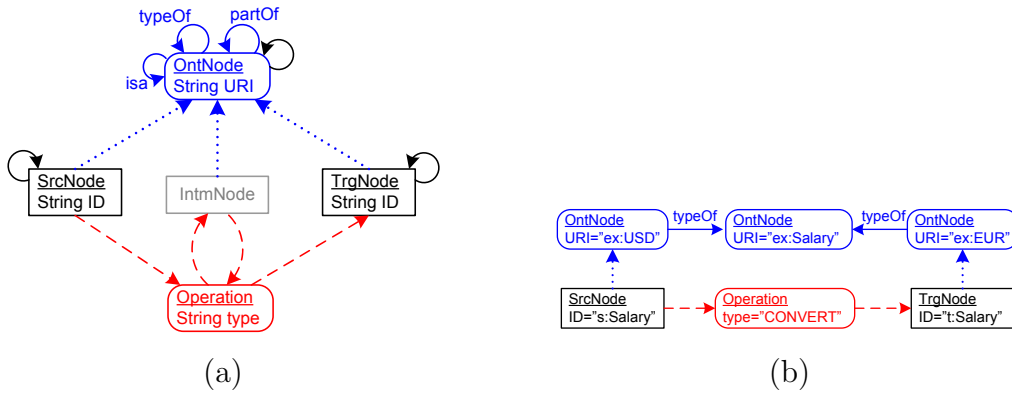


Figure 4.4: (a) *The type graph and (b) a sample instance graph*

flow.) The finally obtained graph is a supergraph of the initial graph, depicting the choice and order of the aforementioned required operations.

In the generated graph, ETL operations are represented by nodes, with incoming and outgoing edges corresponding, respectively, to the inputs and outputs of the operation. These form flows between source nodes and target nodes. Since populating a target element with data from a source element often requires more than one transformation to be performed on the data, in the general case these flows will have length higher than 1. To allow for such functionality, we use the notion of *intermediate nodes*. These refer to intermediate results produced by an ETL operation and consumed by a following one. Consequently, the incoming edges of a node representing an ETL operation may originate either from source nodes or from intermediate nodes, while outgoing edges may be directed either to target nodes or to intermediate nodes.

To formally capture such relationships, we introduce the type graph illustrated in Figure 4.4 and explained in detail below. The *type graph* specifies the types of nodes and edges that the instance graphs (i.e., those constructed to model data store schemata, annotations, and ETL flows) may contain, as well as how they are structured. The type graph is depicted in Figure 4.4(a) and distinguishes the following types of nodes and edges:

- *Ontology nodes (OntNode)*: they represent concepts of the considered application domain. An ontology node may connect to other ontology nodes by means of `isa`, `partOf`, `typeOf` or `connects` edges. The latter correspond to generic relationships between concepts of the domain, and they are represented in Figure 4.4(a) by continuous, unlabeled arrows; the former are represented by continuous arrows with a corresponding label to distinguish the type of the relationship. Each ontology node has an associated URI that uniquely identifies it.
- *Source nodes (SrcNode)*: they correspond to elements of the source data store schemata (e.g., tables or attributes in the case of relational schemata, or XML tree nodes in the case of XML documents.) Each source node has a unique ID (i.e., a URI), prefixed accordingly to indicate the data store it belongs to. Source nodes may relate to each other by `connects` edges (corresponding, for example, to foreign keys in the case of relational sources or to containment

relationships in the case of XML.) Source nodes are annotated by ontology nodes, as shown by the dotted edge in Figure 4.4(a), to make explicit the semantics of the enclosed data.

- *Target nodes* (**TrgNode**): they are similar to source nodes, except from the fact that they refer to elements of the target data stores instead.
- *Intermediate nodes* (**IntmNode**): they are nodes containing temporary data that are generated during ETL operations. They are also annotated by ontology nodes. This is necessary for continuing the flow of operations once an intermediate node has been created. Notice however the difference: source and target nodes are annotated manually (or perhaps semi-automatically) and these annotations need to be in place a-priori, i.e., at the beginning of the ETL design process. In fact, these annotations constitute the main driving force for deriving the ETL scenario. On the contrary, the annotations of the intermediate nodes are produced automatically, when the intermediate node is created, and are a function of the type of ETL operation that created this node, as well as of the (annotation of the) input used for that operation.
- *Operation nodes* (**Operation**): they represent ETL operations. The attribute **type** identifies the type of the operation (e.g., **filter** or **convert**). The inputs and outputs of an operation are denoted by dashed edges in Figure 4.4(a). In particular, the input of an operation is either a source node or an intermediate node, whereas the output of an operation is either an intermediate node or a target node. Each ETL operation must have at least one incoming and one outgoing edge.

Example. A sample instance of the considered type graph is illustrated in Figure 4.4(b). It depicts a typical scenario where an ETL operation converts the values of a source element containing salaries expressed in U.S. Dollars to populate a target element with the corresponding values in Euros.

4.3.4 The Transformation Rules

Having the type graph introduced in the previous section, we can create instances of this graph to represent specific instances of the ETL design problem, i.e., to model a given source graph, a given target graph, and their annotations with respect to an associated domain ontology. The initial graph does not contain any **Operation** nodes. Instead, the goal of the transformation process is exactly to add such nodes in a step-by-step manner, by applying a set of corresponding transformation rules. Recall from Section 4.3.2 that each such rule comprises two basic parts: a) the left-hand-side (LHS), specifying the pattern that triggers the execution of the rule, and b) the right-hand-side (RHS), specifying how the LHS is transformed by the application of the rule. Optionally, a rule may have a third part, specifying one or more *negative application conditions* (NACs). These are patterns preventing the triggering of the rule. A common usage of NACs is as stop conditions, i.e., to prevent the same rule from firing multiple times for the same instance. This occurs when the RHS of the rule also contains the LHS.

In the following, we introduce a set of rules used to construct ETL flows based on the operations (and their conditions) described in Section 4.3.1, and describe each

rule in detail. Essentially, these rules are divided into groups, each one responsible for the addition of a certain type of ETL operation. We consider two kind of rules, referring, respectively, to *simple* and *composite* ETL operations.

Rules for simple operations. This set of rules handles the `LOAD`, `FILTER`, `CONVERT`, `EXTRACT`, and `CONSTRUCT` operations.

LOAD. This is the simplest operation: it simply loads data records from a source to a target element. For such a direct data flow to be valid, one of the following conditions must apply: either a) the source element must correspond to a concept that is the same with that of the target element, or b) the source element must correspond to a concept that is subsumed (i.e., has an `isa` link) by that of the target element. In the former case the rule pattern searches for a pair of source and target nodes that point to the same `OntNode`, as shown in Figure 4.5 (the numbers indicate matched nodes/edges.) If a match is found, the rule is triggered and a `LOAD` operation is inserted.

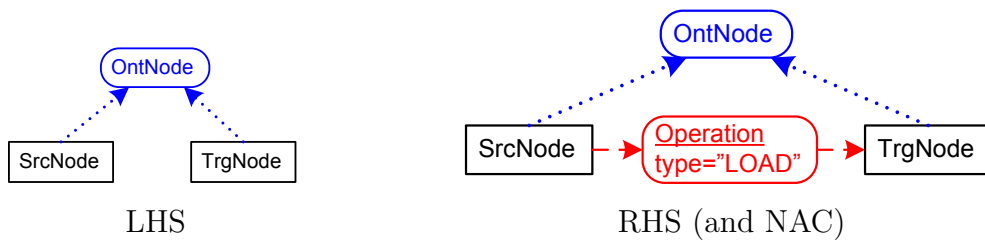


Figure 4.5: Rules for inserting `LOAD` operations in the presence of direct relationship

In the latter case the pattern searches for a `SrcNode` that is annotated by an `OntNode` which has an `isa` relationship to another `OntNode` annotating a `TrgNode` (Figure 4.6.) Again, the transformation performed by the rule is to insert an `Operation` node of type `LOAD`, connecting the source and target nodes. Additionally, in the second case, it is also useful to have data flow to (or from) an intermediate node, which will then be further transformed to meet the target node specifications (or respectively that has resulted from previous transformations). Thus, for this latter case we have four individual rules corresponding to the pairs source-to-target, source-to-intermediate, intermediate-to-intermediate, and intermediate-to-target (rules i - iv in Figure 4.6.) Finally, NACs are used accordingly, to prevent the same rule firing repeatedly for the same pattern, as mentioned previously. Hence, NACs replicating the RHS of the corresponding rule are inserted. An exception to this can be observed in rule iii of Figure 4.6, handling the case intermediate-to-intermediate. Here, in addition to the NAC replicating the RHS of the rule, two other NACs are used to ensure that a `LOAD` operation will not be inserted to an intermediate node, if this node was produced as a result of a previous `FILTER` operation from another intermediate or source node (this will become clearer in the description of `FILTER` operation below.)

FILTER. This operation applies a filtering, such as arithmetic comparisons or regular expressions on strings, on data records flowing from the source to the target data store. The LHS of this rule searches for a target node pointing to a concept that is a subconcept (i.e., more restricted) of a concept corresponding to a source node. Whenever a match is found, the rule is triggered and it inserts a `FILTER` operation between the source and target nodes. Analogously to the previous case, three other

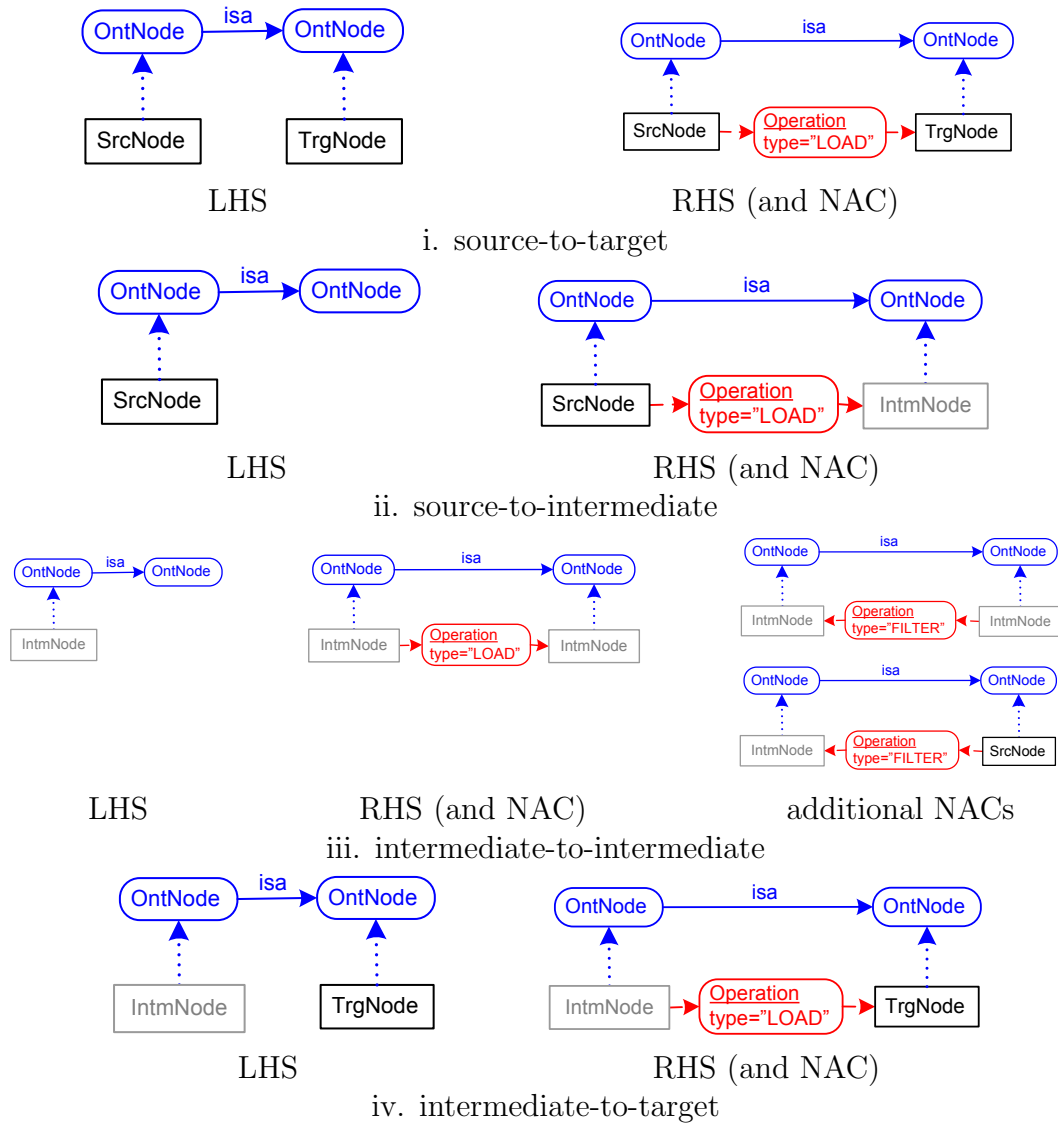


Figure 4.6: Rules for inserting *LOAD* operations via *isa* link

“versions” of this rule are also considered, dealing with the cases of intermediate nodes. The rules for the cases source-to-target and intermediate-to-intermediate are illustrated in Figure 4.7. Notice the additional NACs used again in the latter case. The necessity of these NACs (and of those used previously in the corresponding rule for *LOAD* operations) becomes evident if we consider the following situation. Assume two ontology concepts *C* and *D* related via an *isa* link, $isa(C, D)$, and an intermediate node *V* pointing at (i.e., annotated by) *C*. Then, rule iii of Figure 4.6(b) will fire, inserting a *LOAD* operation leading to a new intermediate node *U*. Subsequently, in the absence of the aforementioned NACs, the rule ii of Figure 4.7 will fire, inserting a *FILTER* operation leading back to node *V*.

CONVERT. This operation represents conceptually the application of arbitrary functions used to transform data records, such as arithmetic operations or operations for string manipulation. It can be thought of as transforming the data between different representation formats. In the ontology this knowledge is captured by means of concepts related to a common concept via *typeOf* links. Thus, the LHS for this

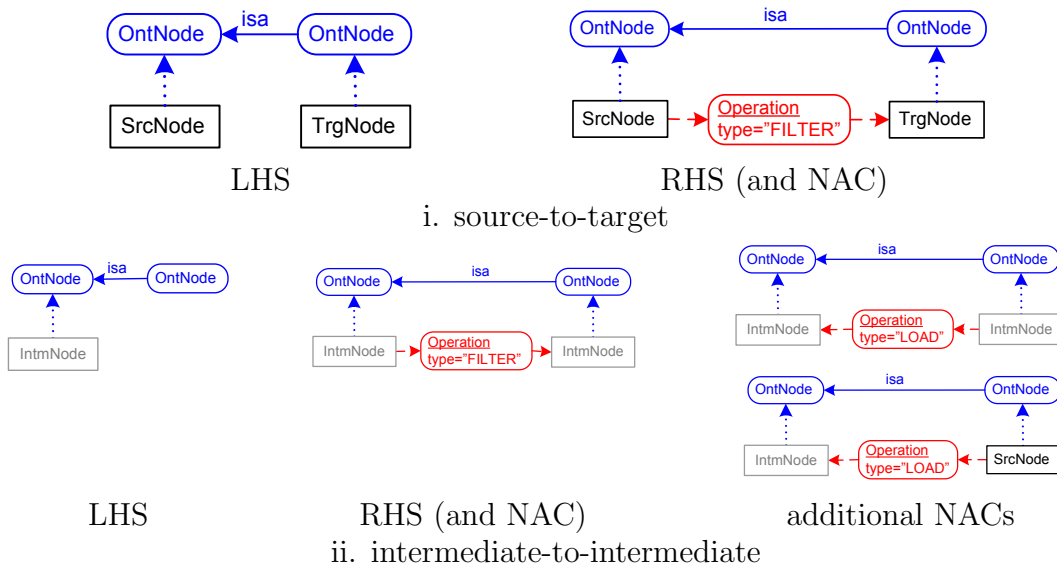


Figure 4.7: Rules for inserting *FILTER* operations

rule is as shown in Figure 4.8, while the RHS inserts, as expected, a **CONVERT** operation between the matched nodes. Due to space considerations, only the transition between intermediate nodes is shown. The derivation of the corresponding rules involving source or target nodes is straightforward. Notice the additional NACs used here. This is to prevent loops converting repeatedly among the same types. For instance, consider the case of three concepts C_1 , C_2 and C_3 , which are all “type of” C . In the absence of these NACs, this would lead to a series of conversions starting, e.g., from C_1 , going to C_2 , then to C_3 , and then back to either C_1 or C_2 , and so on. Instead, this is prevented by the two NACs checking whether the considered intermediate node is itself a product of another **CONVERT** operation.

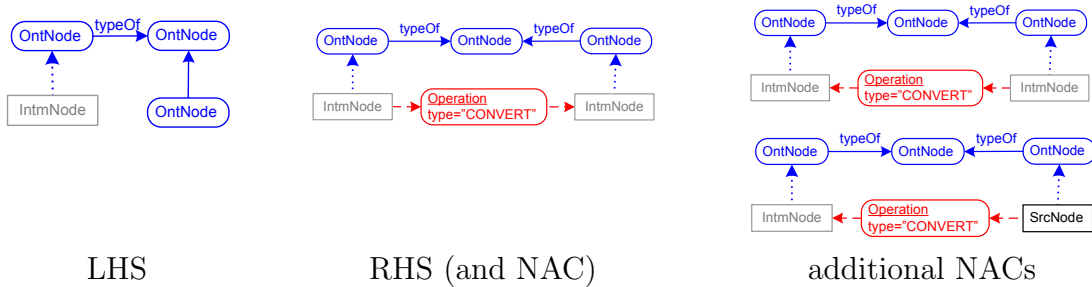


Figure 4.8: Rules for inserting *CONVERT* operations

EXTRACT. This operation corresponds to the case of extracting a piece of information from a data record (e.g., a substring from a string). In this case we search for a pair of source and target nodes, where the latter corresponds to an ontology concept that is related via a **partOf** link to that of the former. When a match is found, the RHS of the rule inserts an **EXTRACT** operation. Three similar rules are constructed again to handle intermediate nodes. Figure 4.9 depicts the rule for the case of transition between intermediate nodes. As described before for the rules **LOAD** and **FILTER**, appropriate NACs are introduced to prevent loops that may occur in

combination with **CONSTRUCT** operations (see below).

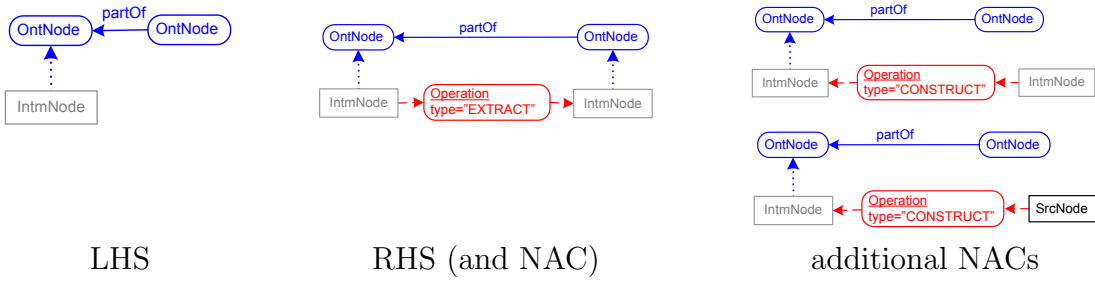


Figure 4.9: Rules for inserting *EXTRACT* operations

CONSTRUCT. This operation corresponds to the case that a larger piece of information needs to be constructed given a data record (typically by filling in the missing part(s) with default values). This is represented by a pair of source and target nodes, where the corresponding source **OntNode** is **partOf** the corresponding target **OntNode**. When triggered, the rule inserts a **CONSTRUCT** operation. Rules for dealing with intermediate nodes operate similarly. In this case, care needs to be taken to avoid loops created by transitions back and forth a pair of **OntNodes** linked with a **partOf** edge, i.e., interchanging **EXTRACT** and **CONSTRUCT** operations. The rule referring to a pair of intermediate nodes is depicted in Figure 4.10.

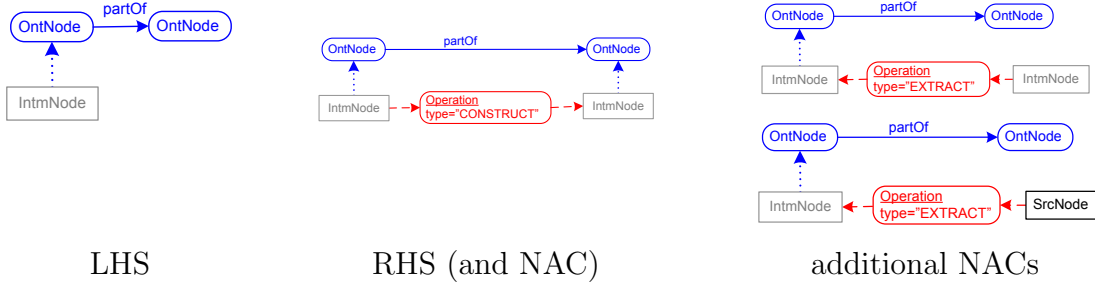


Figure 4.10: Rules for inserting *CONSTRUCT* operations

Rules for composite operations. Our approach is generic and extensible. It is possible to combine simple operations in order to construct composite ones. We present two transformation rules, dealing with such operations, namely the **SPLIT** and **MERGE** operations; this allows to demonstrate the extensibility of the proposed framework.

SPLIT. This operation can be used in the place of multiple **EXTRACT** operations, when multiple pieces of information need to be extracted from a data record in order to populate different elements in the target data store. A typical example is a string tokenizer. However, since the number of resulting elements is not fixed, it is not possible to construct a rule that directly inserts **SPLIT** operations in the ETL flow (unless some appropriate pre-processing on the domain ontology and the data store schemata is performed). Therefore, we insert such operations indirectly, by first applying temporary **EXTRACT** operations, and then replacing multiple **EXTRACT** operations originating from the same node with a **SPLIT** operation. Notice that

having in these cases a single **SPLIT** operation instead of multiple related **EXTRACT** operations, apart from reflecting more closely the human perception regarding the intended transformation, also has the benefit that results in more compact ETL flows. Hence, the LHS of the rule for inserting **SPLIT** operations searches for two **EXTRACT** operations originating from the same source node, and replaces them with a **SPLIT** operation. Observe however that in the case that more than two **EXTRACT** operations existed, this rule would only merge two of them. Still, the others also need to be merged with the substituting **SPLIT** operation. For this purpose, an additional rule is required, that merges an **EXTRACT** operation to a **SPLIT** operation. This rule is executed iteratively, until all **EXTRACT** operations have been “absorbed” by the **SPLIT** operation. However, since the execution order of rules is non-deterministic, if more than three **EXTRACT** operations exist, originating from the same node, it is possible to end up with multiple **SPLIT** operations. Thus, a third rule that combines two **SPLIT** operations in a single one is employed. The aforementioned rules are presented in Figure 4.11. Similar rules are devised to apply this process for cases involving intermediate nodes.

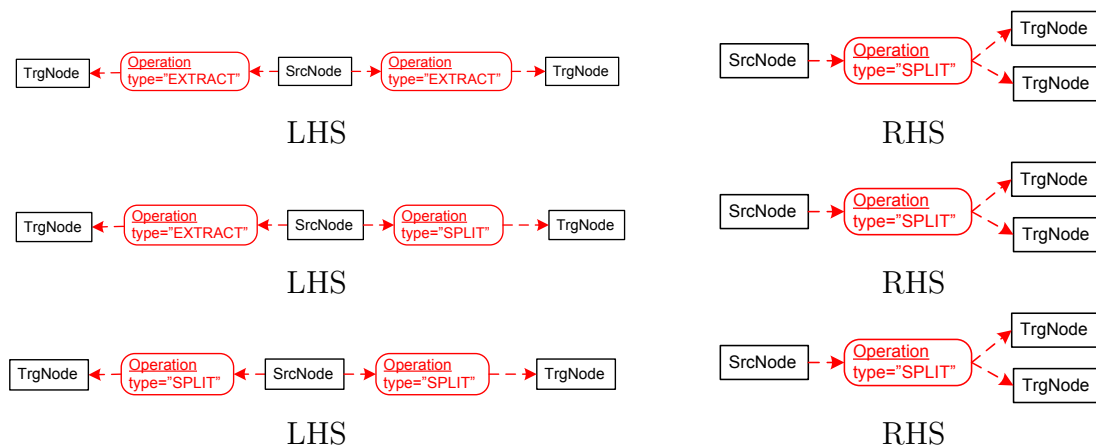


Figure 4.11: Rules for inserting *SPLIT* operations

MERGE. As mentioned earlier, in **CONSTRUCT** operations some external information needs to be provided to construct from a given data item the required data to populate a target element. In this case the missing data is provided by other source elements. That is, two or more source elements complement each other in producing the data records for populating a given target element. As with the case of **SPLIT** mentioned above, since the number of cooperating source nodes is not fixed, this operation is also handled indirectly, in a similar manner. In particular, the rules for **MERGE** search for two **CONSTRUCT** operations or for a **MERGE** and a **CONSTRUCT** operation or for two previously inserted **MERGE** operations, and incorporate them in a single **MERGE** operation. As previously, multiple **CONSTRUCT** operations are iteratively absorbed into a single **MERGE** operation by consecutive executions of the corresponding rules. The corresponding rules are shown in Figure 4.12.

Additional rules. As it may have become clear from the description of the transformation rules previously presented, when there is not a one-step transformation between a source and a target node, the graph transformation engine will simulate a (random) search for creating paths of operations that may lead from the source node to the target. (The randomness is due to the two kinds of non-determinism

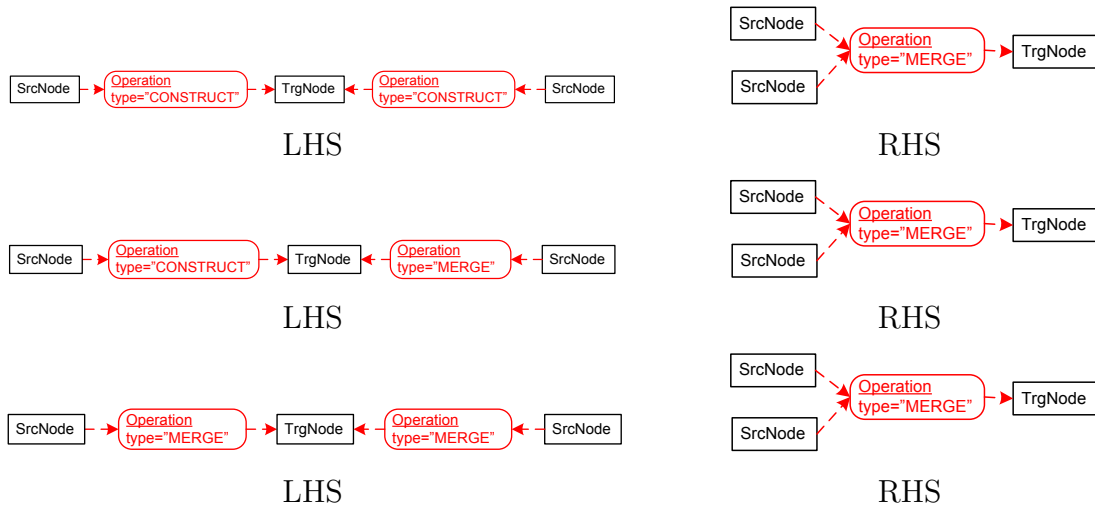


Figure 4.12: Rules for inserting *MERGE* operations

mentioned in Section 4.3.2.) It is most likely that for most (or even all) of these paths after a few transformation steps no more rules can be applied, without having reached a target node. To avoid overloading the resulting graph, and consequently the ETL designer, with these “false positives”, we employ an additional rule that aims at “cleaning up” the final ETL flow. This rule, illustrated in Figure 4.13, essentially removes intermediate nodes, together with the operation producing them, that do not have a next step in the ETL flow (i.e., that fail to reach a target node).



Figure 4.13: “Clean-up” rule

4.3.5 Creation of the ETL Design

Next, we discuss ordering issues in the execution of the transformation rules. It is evident from the description of the functionality of the introduced rules that some of the rules should be considered before or after other ones have been applied. In particular, we consider the following requirements:

- Rules referring to one-step transformations, i.e., involving source and target nodes, should be considered before rules involving intermediate nodes.
- The rule “clean-up” should be performed only after the examination of any rules adding new operations has been completed.
- Rules regarding composite operations (e.g., *SPLIT* and *MERGE*) should be considered after all the rules for the corresponding simple operations (e.g., *EXTRACT* and *CONSTRUCT*) have been triggered.

Ensuring that this ordering is respected is both a necessary condition for the method to produce the desired results and a matter of improving performance. For instance, allowing clean-up operations to be performed before rules inserting new operations have been completed, it may result in an infinite loop, i.e., repeatedly adding and removing the same operation(s). On the other hand, checking for the applicability of rules regarding **SPLIT** or **MERGE** operations before all **EXTRACT** or **CONSTRUCT** operations have been identified, leads to redundant matching tests. Consequently, we organize the rules described above into 4 layers, as follows:

- The first layer comprises those rules inserting ETL operations that directly connect a source node to a target node.
- The second layer comprises rules inserting operations from or to intermediate nodes.
- The third layer contains the clean-up rule.
- Finally, the last, fourth, layer comprises the rules for composite operations (i.e., **SPLIT** and **MERGE**).

These layers are executed in the above order, starting from the first layer. The execution of rules from a layer i starts only if no more rules from the layer $i - 1$ can be applied. The whole process terminates when no rules from the last layer can be applied. Within the same layer, the order in which the rules are triggered is non-deterministic.

Hence, given the presented set of rules, organized appropriately in the aforementioned layers, and the problem instance, comprising the source graph, the target graph, the ontology and the annotations, the creation of the ETL design proceeds as follows:

- Step 1: Identify single operations that can connect a source node to a target node. This is accomplished by the graph transformation engine applying the rules of the first layer.
- Step 2: This step accomplishes the rules of the second layer and it comprises two tasks, which may be executed interchangeably:
 - Starting from source nodes, introduce ETL operations that transform data leading to an intermediate node.
 - Starting from the *created* intermediate nodes, continue introducing additional transformations, until either the target nodes are reached or no more rules can be applied.
- Step 3: Remove paths of ETL operations and intermediate nodes that have not reached a target node. This is performed by the rule in layer 3.
- Step 4: Search for groups of **EXTRACT** or **CONSTRUCT** operations that can be substituted by **SPLIT** or **MERGE** operations, respectively.

sources	s_customers	{ cid, name, country, city, street }
	s_orders	{ oid, cid, date, amount, price }
targets	t_customers	{ cid, firstName, lastName, address }
	t_orders	{ oid, cid, date, amount, price }

Table 4.12: Source and target schemata for the example

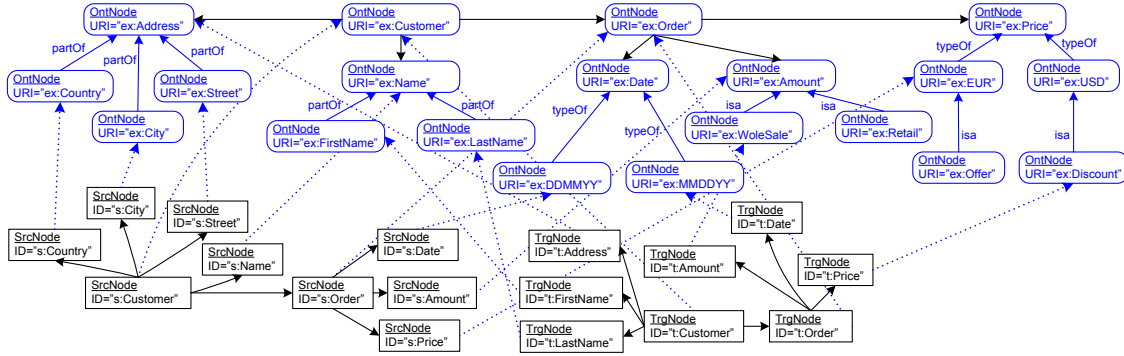


Figure 4.14: Example

Correctness of the produced flow. Within a flow of ETL operations, the execution order of the operations is significant, as different orderings may produce semantically very different results. In [144], the issue of correctness of the execution order of operations in an ETL workflow has been introduced, and formal rules have been presented that ensure such correctness. In the same spirit, we work in the approach presented in this work. For instance, assume two pairs of operations. The first one involves a function that converts Euro values to Dollar values for an hypothetical attribute Cost, and a filter that allows only cost values over 100 Dollars; i.e., $c : E \rightarrow \$$ and $f : \$ > 100$. In that case, it is necessary to have the function c , represented as a **CONVERT** operation in our approach, before the filter operation f . The second pair involves, let's say, the same function $c : E \rightarrow \$$ and another one that transforms dates from European to American format; i.e., $c' : EDate \rightarrow ADate$. In that case, both orderings either $\{c, c'\}$ or $\{c', c\}$ are correct, since the two operations are applied to different attributes (see [144] for more details.) Our method captures both cases, as the desired ordering is determined by the (relative) position in the ontology graph of the ontology nodes annotating the transformed data records.

4.3.6 Illustrative Example

We demonstrate the presented methodology by means of an example. The source and target schemata used for this example have been chosen appropriately from the TPC-H schema [160] to resemble typical real-world scenarios. We keep the example concise, tailoring the source and target graphs so that a small number of schema elements will suffice for demonstrating the main aspects of our framework.

We assume two main entities, namely customers and orders, while the whole setting is represented in Table 4.12. A customer has a name, comprising his/her first and last name, and an address, which consists of his/her country, city and

street. An order is placed in a particular date, which can be recorder in either the “DD/MM/YY” or the “MM/DD/YY” format. It also refers to an amount of items. This amount can be categorized as “retail” or “wholesale”, according to whether it exceeds a specific threshold. Finally, the price of the order can be recorder in either USD or EUR. We also assume the existence of special offers and discounts, and suppose that the currency for the former is EUR, while for the latter it is USD. This information is reflected in the sample ontology shown in Figure 4.14, where ontology concepts are represented by round rectangles. The figure also illustrates a source and a target schema (nodes prefixed with “s” and “t”, respectively), with their elements being annotated by elements of the ontology (dotted lines). Notice, for example, the structural differences in representing the customer’s name and address, as well as the different formats and currencies used in the two data stores for an order’s date and price.

This graph constitutes the starting point for the graph transformation process. The annotations make explicit the semantics of the data in the corresponding elements, and are obtained either manually or semi-automatically (e.g., through automatic schema matching techniques [129, 140]) by the administrator through processes ranging from oral communication with the administrators of the corresponding data stores to study of the elements’ comments and/or accompanying documentation. Note that due to the size of the involved schemata, such graphs can be quite large. This is not a disadvantage of our proposed approach, but an inherent difficulty of the ETL design task. Nevertheless, we can tackle this issue either by exploiting existing advanced techniques for visualization of large graphs (e.g., [164]) or by using simple zoom-in/out techniques for exploring certain interesting parts of the graph [165].

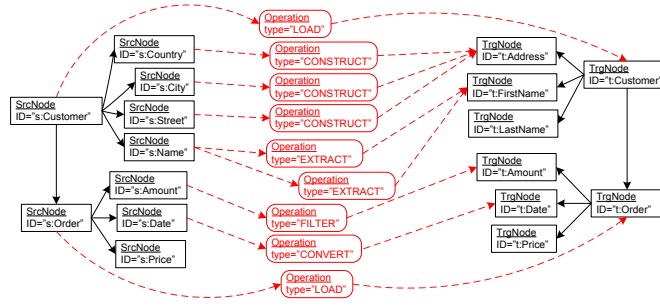
Next, the ETL flow is computed by the graph transformation engine, starting from the above input graph and applying the rules presented in Section 4.3.4. To better illustrate the process, we separately display the result produced by the execution of each layer of rules.

The result of the first layer is depicted in Figure 4.15(a). For brevity, we omit the ontology nodes. Recall that the first layer is responsible for one-step transformations. Hence, no data flow between the elements `s:Price` and `t:Price` has been determined, as no single operation is sufficient to meet the required target specification.

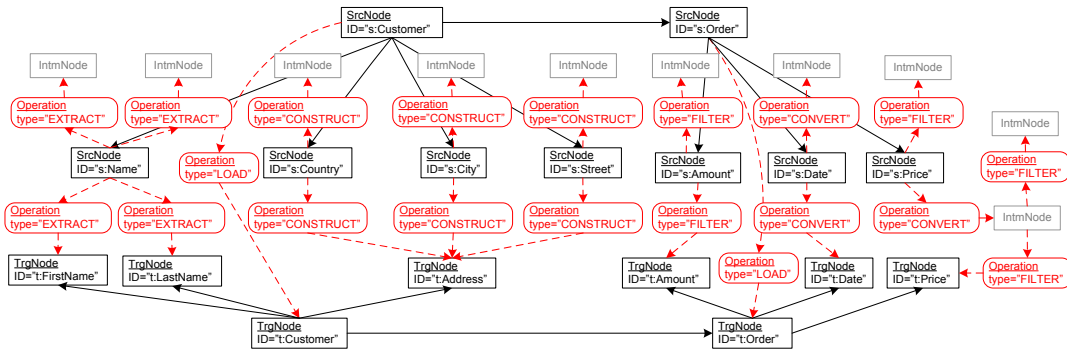
Afterward, the rules involving intermediate nodes are executed. The corresponding output is shown in Figure 4.15(b). Notice the data flow that has now been created between the elements `s:Price` and `t:Price`, comprising one `CONVERT` and one `FILTER` operation. On the way, some intermediate nodes not leading to a target node, have been introduced. These are removed after the execution of layer 3 (Figure 4.15(c).) Finally, the `EXTRACT` and `CONSTRUCT` operations are incorporated into `SPLIT` and `MERGE` operations, respectively, during the execution of layer 4. The final result is presented in Figure 4.15(d).

4.4 Summary

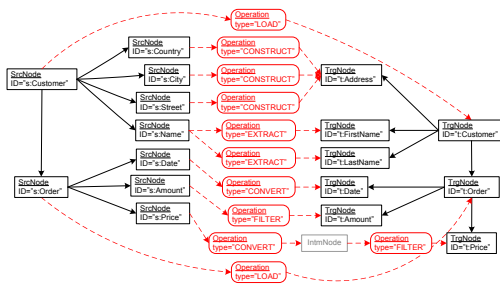
In this chapter, we have proposed the use of an ontology to facilitate the conceptual design of ETL processes. The available data sources are represented conceptually by a graph and are annotated using an appropriate domain ontology. Then, we have



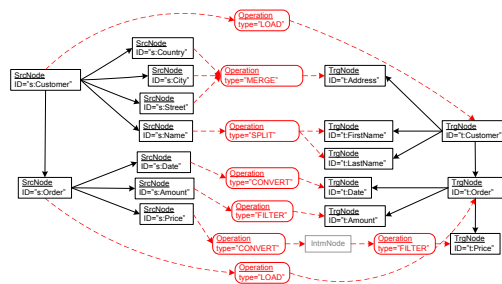
(a) layer 1



(b) layer 2



(c) layer 3



(d) layer 4

Figure 4.15: Output of the graph transformation process

explored two directions for the design of the ETL process. In the first approach, the semantic annotations and the ontology are used by an OWL reasoner to infer correspondences and conflicts between the sources and the target. This allows to identify the appropriate inter-schema mappings and transformations that should drive the flow of data from the data sources to the target Data warehouse. The second approach considers the design of an ETL process as a series of conditional graph transformations. For this purpose, we have proposed a customizable and extensible set of graph transformation rules, which drive the construction of the ETL process, again in conjunction with the semantic information conveyed by the associated ontology.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis has focused on the discovery and selection of Semantic Web services, as well as the discovery and integration of data from heterogeneous, distributed sources, using ontologies. In the last years there has been an enormous and continuous growth of the amount of the information available on the Web, as well as its diversity. At the same time, the number of Web users has also increased significantly, and their needs have become more complex. These facts have raised new challenges in the way information is organized, searched, and delivered to the user. In this direction, this thesis has proposed novel techniques for discovering relevant information sources and services, and for reconciling and integrating data from distributed, heterogeneous sources to meet the user needs. The presented work exploits the advances of the research towards the transition to the Semantic Web, and relies on the semantic annotation of data sources and services with appropriate ontologies.

First, we have dealt with the problem of service discovery and selection on the Semantic Web. Previous research efforts have focused on approaches to enhance the service descriptions with semantic information, and then to exploit this information to address the matchmaking between requested and offered services through logic inference. Our work elaborates on these approaches and extends them providing more advanced capabilities. We have argued for the importance of ranking the match results, and we have proposed a semantic similarity measure to quantify the degree of match between the descriptions of requested and offered services. In contrast to previous approaches, which aggregate the degrees of match between individual service parameters to compute the overall degree of match, thus resulting in information loss, we have proposed a method to select the best matches based on the notion of skyline queries. Moreover, in contrast to previous work that recognises the need to use multiple matching criteria for service discovery but treats them as alternatives, we propose methods that select the best matches taking into consideration all the available matching criteria simultaneously. The proposed methods have been validated through experimental analysis and comparison to existing approaches, showing that they achieve a significant improvement of the precision of the retrieved results.

Following that, we have focused on the search of services and data in distributed environments. Motivated by the benefits in the precision of service retrieval due to our ranking methods, as well as the need to improve the efficiency of the search

engine, we have emphasized on retrieving results progressively. For this purpose we have exploited an encoding of the service descriptions based on the use of a labeling scheme for concept hierarchies, which allows the matchmaking to be performed without invoking the reasoner at query time. Then we have shown how these encoded representations can be stored and search in a suitable, structured P2P overlay network. Our experimental results support the case for progressive search and demonstrate the resulting savings in the search cost. Next, we have considered an alternative distributed settings, where peers in an unstructured network share semantically annotated, structured data. In contrast to traditional approaches which typically emphasize on (strict) query rewriting algorithms, our work considered the exploitation of the additional semantic information to identify peers with relevant content or peers that are potentially more capable of answering a particular request.

Finally, we have considered the problem of reconciling and integrating data from heterogeneous sources that have been previously discovered, in order to meet the user needs and specifications. We have considered the use of Extract-Transform-Load processes for this purpose, and in particular we have focused on the use of ontologies to facilitate their conceptual design. Our approach assumes that the ETL designer uses an appropriate domain ontology to annotate the involved data sources, and then, based on these annotations, our method automatically infers several operations that are required to transform data from the sources to the target. Instead, existing tools and approaches for this task focus on providing the user with the means to graphically specify the ETL scenario, but the identification of the operations required is done manually. In fact, our approach provides two alternative methods. In the first case, a reasoner is used directly to infer correspondences and conflicts between the data sources and the target. In the latter case, the ETL scenario is constructed in a step-by-step manner, based on a set of graph transformation rules, which gives to the user more control of the process.

All the aforementioned techniques proposed in this thesis, combined, address and facilitate several major tasks necessary to achieve the overall goal which is to use semantics to discover services on the Web and to integrate data coming from heterogeneous sources.

5.2 Future Work

In the previous chapters we have presented in detail the outcomes of the work conducted in the context of this thesis. Still, several research issues remain open. We conclude by identifying and outlining the most prominent ones:

- The problems considered in this thesis are very relevant to trends and requirements that can be identified in other emerging paradigms as well, most notably sensor networks and grid computing. The former are networks of spatially distributed autonomous devices using sensors to cooperatively monitor the environment. The Semantic Sensor Web [139] is an attempt to annotate sensor data with spatial, temporal, and thematic metadata. This will facilitate the discovery and analysis of these data, and, consequently, will allow the monitoring of complex events and situations. In grid computing, a cluster of networked, loosely-coupled computers share resources and collaborate to

perform a very large task. The Semantic Grid [1] is an extension where computing resources and services are annotated with metadata to facilitate their discovery and integration. Thus, it would be a very interesting challenge to study how the techniques proposed in this thesis can be adapted or extended to provide solutions to such environments.

- Another important issue when transforming data from a set of sources to a target repository, is related to the notion of provenance [30]. Provenance refers essentially to capturing, representing, and managing metadata for tracing the origin of data elements that take part in various operations. It can be viewed at different levels of detail, such as: which sources contribute to a given element; which particular elements within these sources; which transformations has this element undergone. Provenance plays a significant role in several aspects, most notably it can be used to: determine why a specific data value exists in the target repository, for instance explaining whether a recorded profit loss is due to an actual decrease on sales or due to changes on currency exchange rates; assign default values to missing data elements, based on their source of origin, creation time, etc.; determine the accuracy, timeliness, and confidence of the information presented to the user.
- When data are gathered from several distributed, heterogeneous sources, potentially of different levels of quality, they may be uncertain, incomplete or inconsistent. This requires advanced techniques in order to manage and reason with such data. Hence, it would be a challenging issue to study how the techniques proposed in this thesis can be adapted to take into account this additional aspect. An overview of basic concepts and a survey of existing techniques regarding this topic can be found in [124].
- Our work on service discovery has assumed that a match result comprises a single available service. However, it is likely in some cases that no single service exists that meets the specified criteria, but instead it may be possible to provide the required functionality by composing available services. Hence, the service discovery engine could be enhanced with service composition capabilities. Indeed, service composition is a research topic that has been attracting a lot of research interest over the recent years, and a variety of techniques have been proposed, ranging from workflows to AI planning [131].
- Our approach for data sharing on ontology-enhanced Peer Data Management Systems can be further extended in two main aspects. First, our current approach takes into consideration only information at the schema level. This could be complemented by techniques that take into account data-level information, i.e., summaries and statistics about the peers' contents, as well as statistics and cached results from previous queries. Second, the comparison is currently performed between pairs of neighboring peers. It would be desirable to extend this comparison to a cluster of peers, so that when a peer chooses the direction to forward a query, it would take into account not only the peer in the next hop but other subsequent peers along that path as well. This would allow to build routing indexes for query forwarding, along the lines of the work presented in [99]. Finally, another direction is to explore the application of

the proposed semantic similarity measure in the context of social networks, in order to identify users with relevant profiles and interests.

- The work concerning the design of ETL processes has focused on their design at the conceptual level. However, the uttermost goal of such a task is to specify also the design at the logical and physical levels. Hence, an extension of the work would be to study the transition from the conceptual to the logical level, along the lines of the work presented in [142]. Also, another interesting and important issue is to study the effect of evolution to the design of the ETL process, i.e., how the ETL scenario can be automatically adapted when changes to the schemas of the sources or the target occur [121].

Bibliography

- [1] *The Semantic Grid: A Future e-Science Infrastructure*, John Wiley & Sons, 2003.
- [2] Karl Aberer, Philippe Cudré-Mauroux, and Manfred Hauswirth, *Start Making Sense: The Chatty Web Approach for Global Semantic Agreements*, J. Web Sem. **1** (2003), no. 1, 89–114.
- [3] Karl Aberer, Philippe Cudré-Mauroux, Manfred Hauswirth, and Tim Van Pelt, *GridVine: Building Internet-Scale Semantic Overlay Networks*, ISWC, 2004, pp. 107–121.
- [4] Serge Abiteboul, Peter Buneman, and Dan Suciu, *Data on the Web: From Relations to Semistructured Data and XML*, Morgan Kaufmann Publishers Inc., 2000.
- [5] Sanjay Agrawal, Surajit Chaudhuri, Gautam Das, and Aristides Gionis, *Automated Ranking of Database Query Results*, CIDR, 2003.
- [6] Rama Akkiraju and et. al., *Web Service Semantics - WSDL-S*, W3C Member Submission, November 2005.
- [7] Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Sascha Roeder, *A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI.*, IIWeb, 2003, pp. 87–92.
- [8] José Luis Ambite and Dipsy Kapoor, *Automatically Composing Data Workflows with Relational Descriptions and Shim Services*, ISWC/ASWC, 2007, pp. 15–29.
- [9] Stephanos Androutsellis-Theotokis and Diomidis Spinellis, *A Survey of Peer-to-Peer Content Distribution Technologies*, ACM Comput. Surv. **36** (2004), no. 4, 335–371.
- [10] Grigoris Antoniou and Frank van Harmelen, *A Semantic Web Primer, 2nd Edition*, 2 ed., The MIT Press, March 2008.
- [11] Marcelo Arenas, Vasiliki Kantere, Anastasios Kementsietsidis, Iluju Kiringa, Renée J. Miller, and John Mylopoulos, *The Hyperion Project: From Data Integration to Data Coordination*, SIGMOD Record **32** (2003), no. 3, 53–58.
- [12] Marcelo Arenas and Leonid Libkin, *XML Data Exchange: Consistency and Query Answering*, 24th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2005, pp. 13–24.

- [13] Yigal Arens, Craig A. Knoblock, and Wei-Min Shen, *Query Reformulation for Dynamic Information Integration*, J. Intell. Inf. Syst. **6** (1996), no. 2/3, 99–130.
- [14] Javed A. Aslam and Mark H. Montague, *Models for Metasearch*, SIGIR, 2001, pp. 275–284.
- [15] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider (eds.), *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press, 2003.
- [16] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto, *Modern Information Retrieval*, ACM Press / Addison-Wesley, 1999.
- [17] C. Ballard, D. Herreman, D. Schau, R. Bell, K. Eunsang, and A. Valencic, *Data Modeling Techniques for Data Warehousing*, IBM Red Book, 1998.
- [18] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella, *SaLSa: Computing the Skyline Without Scanning the Whole Sky*, ACM CIKM, 2006, pp. 405–414.
- [19] Ulrich Basters and Matthias Klusch, *RS2D: Fast Adaptive Search for Semantic Web Services in Unstructured P2P Networks*, ISWC, 2006, pp. 87–100.
- [20] Catriel Beeri, Alon Y. Levy, and Marie-Christine Rousset, *Rewriting Queries Using Views in Description Logics*, PODS, 1997, pp. 99–108.
- [21] Umesh Bellur and Roshan Kulkarni, *Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching*, ICWS, 2007, pp. 86–93.
- [22] T. Bellwood, L. Clément, D. Ehnebuske, A. Hately, Maryann Hondo, Y.L. Husband, K. Januszewski, S. Lee, B. McKee, J. Munter, and C. von Riegen, *The Universal Description, Discovery and Integration (UDDI) protocol. Version 3*, UDDI.org, 2003.
- [23] Tim Berners-Lee, James Hendler, and Ora Lassila, *The Semantic Web*, Scientific American **284** (2001), no. 5, 34–43.
- [24] Michael Böhnlein and Achim Ulbrich vom Ende, *Deriving Initial Data Warehouse Structures from the Conceptual Data Models of the Underlying Operational Information Systems*, ACM 2nd International Workshop on Data Warehousing and OLAP, 1999, pp. 15–21.
- [25] Kalina Bontcheva, *Generating Tailored Textual Summaries from Ontologies*, ESWC, 2005, pp. 531–545.
- [26] Kalina Bontcheva and Yorick Wilks, *Automatic Report Generation from Ontologies: The MIAKT Approach*, NLDB, 2004, pp. 324–335.
- [27] W. N. Borst, *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, Ph.D. thesis, University of Enschede, 1997.
- [28] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker, *The Skyline Operator*, ICDE, 2001, pp. 421–430.

- [29] Dan Brickley and R.V. Guha, *RDF Vocabulary Description Language 1.0: RDF Schema*, W3c recommendation, W3C, 2004.
- [30] Peter Buneman and Wang Chiew Tan, *Provenance in Databases*, SIGMOD Conference, 2007, pp. 1171–1173.
- [31] Mark Burstein and et. al., *OWL-S: Semantic Markup for Web Services*, W3C Member Submission, November 2004.
- [32] Andrea Calì, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini, *Data Integration Under Integrity Constraints.*, *Inf. Syst.* **29** (2004), no. 2, 147–163.
- [33] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini, *Answering Queries Using Views in Description Logics*, 6th International Workshop on Knowledge Representation meets Databases, 1999, pp. 6–10.
- [34] Diego Calvanese, Giuseppe De Giacomo, and Riccardo Rosati, *Data Integration and Reconciliation in data Warehousing: Conceptual Modeling and Reasoning Support*, *Networking and Information Systems* **2** (1999), no. 4, 413–432.
- [35] Jorge Cardoso, *Discovering Semantic Web Services with and without a Common Ontology Commitment*, IEEE SCW, 2006, pp. 183–190.
- [36] Tiziana Catarci and Maurizio Lenzerini, *Representing and Using Interschema Knowledge in Cooperative Information Systems*, *Int. J. Cooperative Inf. Syst.* **2** (1993), no. 4, 375–398.
- [37] Suleyman Cetintas and Luo Si, *Exploration of the Tradeoff Between Effectiveness and Efficiency for Results Merging in Federated Search*, SIGIR, 2007, pp. 707–708.
- [38] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana, *Web Services Description Language (WSDL) Version 2.0*, W3C Technical Report, 2007.
- [39] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang, *Skyline with Presorting*, ICDE, 2003, pp. 717–816.
- [40] Vassilis Christophides, Gregory Karvounarakis, Dimitris Plexousakis, Michel Scholl, and Sotirios Tourtounis, *Optimizing Taxonomic Semantic Web Queries Using Labeling Schemes*, *J. Web Sem.* **1** (2004), no. 2, 207–228.
- [41] Wesley W. Chu and Guogen Zhang, *Associative Query Answering via Query Feature Similarity*, IIS, 1997, p. 405.
- [42] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg, *A Comparison of String Metrics for Matching Names and Records*, Data Cleaning Workshop in conjunction with KDD, 2003, pp. 13–18.
- [43] John Colgrave, Rama Akkiraju, and Richard Goodwin, *External Matching in UDDI*, ICWS, 2004, p. 226.

- [44] Arturo Crespo and Hector Garcia-Molina, *Semantic Overlay Networks for P2P Systems*, AP2PC, 2004, pp. 1–13.
- [45] Hercules Dalianis and Eduard H. Hovy, *Aggregation in Natural Language Generation*, European Workshop on Natural Language Generation (EWNLG), 1993, pp. 88–105.
- [46] Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang, *Similarity Search for Web Services*, VLDB, 2004, pp. 372–383.
- [47] Dejing Dou, Drew V. McDermott, and Peishen Qi, *Ontology Translation on the Semantic Web*, J. Data Semantics **2** (2005), 35–57.
- [48] Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa, *Data Exchange: Semantics and Query Answering*, Theor. Comput. Sci. **336** (2005), no. 1, 89–124.
- [49] Ronald Fagin, Phokion G. Kolaitis, and Lucian Popa, *Data Exchange: Getting to the Core*, ACM Trans. Database Syst. **30** (2005), no. 1, 174–210.
- [50] Ronald Fagin, Amnon Lotem, and Moni Naor, *Optimal Aggregation Algorithms for Middleware*, PODS, 2001.
- [51] Mohamed Farah and Daniel Vanderpooten, *An Outranking Approach for Rank Aggregation in Information Retrieval*, SIGIR, 2007, pp. 591–598.
- [52] Edward A. Fox and Joseph A. Shaw, *Combination of Multiple Searches*, 2nd TREC, NIST, 1993, pp. 243–252.
- [53] Antara Ghosh, Jignashu Parikh, Vibhuti S. Sengar, and Jayant R. Haritsa, *Plan Selection Based on Query Clustering*, VLDB, 2002, pp. 179–190.
- [54] Fausto Giunchiglia, Mikalai Yatskevich, and Pavel Shvaiko, *Semantic Matching: Algorithms and Implementation*, J. Data Semantics **9** (2007), 1–38.
- [55] Matteo Golfarelli and Stefano Rizzi, *Methodological Framework for Data Warehouse Design*, ACM 1st International Workshop on Data Warehousing and OLAP, 1998, pp. 3–9.
- [56] Matteo Golfarelli, Stefano Rizzi, and Boris Vrdoljak, *Data Warehouse Design from XML Sources*, ACM 4th International Workshop on Data Warehousing and OLAP, 2001, pp. 40–47.
- [57] Google Mashup Editor, <http://www.googlemashups.com>.
- [58] Georg Gottlob, *Web Data Extraction for Business Intelligence: The Lixto Approach*, BTW, 2005, pp. 30–47.
- [59] Antonin Guttman, *R-Trees: A Dynamic Index Structure for Spatial Searching*, SIGMOD Conference, 1984, pp. 47–57.
- [60] H. Lausen, A. Polleres, and D. Roman (eds.), *Web Service Modeling Ontology (WSMO)*, W3C Member Submission, June 2005.

- [61] Peter Haase, Björn Schnizler, Jeen Broekstra, Marc Ehrig, Frank van Harmelen, Maarten Menken, Peter Mika, Michal Plechawski, Pawel Pyszlak, Ronny Siebes, Steffen Staab, and Christoph Tempich, *Bibster - a semantics-based bibliographic Peer-to-Peer system*, J. Web Sem. **2** (2004), no. 1, 99–103.
- [62] Karl Hahn, Carsten Sapia, and Markus Blaschka, *Automatically Generating OLAP Schemata from Conceptual Graphical Models*, ACM 3rd International Workshop on Data Warehousing and OLAP, 2000, pp. 9–16.
- [63] Alon Y. Halevy, *Answering Queries Using Views: A Survey*, VLDB J. **10** (2001), no. 4, 270–294.
- [64] ———, *Answering queries using views: A survey*, VLDB J. **10** (2001), no. 4, 270–294.
- [65] Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov, *Piazza: Data Management Infrastructure for Semantic Web Applications*, WWW, 2003, pp. 556–567.
- [66] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov, *Schema Mediation in Peer Data Management Systems*, ICDE, 2003, pp. 505–516.
- [67] Sidath B. Handurukande, Anne-Marie Kermarrec, Fabrice Le Fessant, and Laurent Massoulié, *Exploiting Semantic Clustering in the eDonkey P2P Network*, ACM SIGOPS European Workshop, 2004, p. 20.
- [68] Jeffrey Hau, William Lee, and John Darlington, *A Semantic Similarity Measure for Semantic Web Services*, Web Service Semantics Workshop at WWW2005 (2005).
- [69] Bodo Hüsemann, Jens Lechtenböcker, and Gottfried Vossen, *Conceptual Data Warehouse Modeling*, DMDW, 2000, p. 6.
- [70] David F. Huynh, Robert C. Miller, and David R. Karger, *Potluck: Data Mash-Up Tool for Casual Users*, ISWC/ASWC, 2007, pp. 239–252.
- [71] IBM Data Warehouse Manager, <http://www.ibm.com/software/data/db2/datawarehouse>.
- [72] Informatica PowerCenter, <http://www.informatica.com/products/powercenter>.
- [73] Krzysztof Janowicz, *Sim-DL: Towards a Semantic Similarity Measurement Theory for the Description Logic CNR in Geographic Information Retrieval*, OTM Workshops (2), 2006, pp. 1681–1692.
- [74] Thorsten Joachims and Filip Radlinski, *Search Engines that Learn from Implicit Feedback*, IEEE Computer **40** (2007), no. 8, 34–40.
- [75] Holger Lausen Joel Farrell, *Semantic Annotations for WSDL and XML Schema*, W3C Recommendation, August 2007.
- [76] Yannis Kalfoglou and Marco Schorlemmer, *Ontology Mapping: The State of the Art*, Knowl. Eng. Rev. **18** (2003), no. 1, 1–31.

- [77] V. Kantere, S. Skiadopoulos, and T. Sellis, *Storing and Indexing Spatial Data in P2P Systems*, IEEE Transactions on Knowledge and Data Engineering (TKDE), (to appear).
- [78] Vasiliki Kantere, Iluju Kiringa, John Mylopoulos, Anastasios Kementsietidis, and Marcelo Arenas, *Coordinating Peer Databases Using ECA Rules*, DBISP2P, 2003, pp. 108–122.
- [79] Frank Kaufer and Matthias Klusch, *WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker*, ECOWS, 2006, pp. 161–170.
- [80] Takahiro Kawamura, Jacques-Albert De Blasio, Tetsuo Hasegawa, Massimo Paolucci, and Katia P. Sycara, *Public Deployment of Semantic Service Matchmaker with UDDI Business Registry.*, ISWC, 2004, pp. 752–766.
- [81] Zoubida Kedad and Elisabeth Métais, *Ontology-Based Data Cleaning*, NLDB, 2002, pp. 137–149.
- [82] Ralph Kimball and Joe Caserta, *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*, John Wiley & Sons, 2004.
- [83] Ralph Kimball, Laura Reeves, Warren Thornthwaite, Margy Ross, and Warren Thornwaite, *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*, John Wiley & Sons, 1998.
- [84] Matthias Klusch, Benedikt Fries, and Katia P. Sycara, *Automated Semantic Web service discovery with OWLS-MX*, AAMAS, 2006, pp. 915–922.
- [85] George Kokkinidis, Eleytherios Sidirourgos, and Vassilis Christophides, *Semantic Web and Peer-to-Peer*, Springer-Verlag, 2006.
- [86] Phokion G. Kolaitis, Jonathan Panttaja, and Wang Chiew Tan, *The Complexity of Data Exchange*, 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2006, pp. 30–39.
- [87] Donald Kossmann, Frank Ramsak, and Steffen Rost, *Shooting Stars in the Sky: An Online Algorithm for Skyline Queries*, VLDB, 2002, pp. 275–286.
- [88] H. T. Kung, F. Luccio, and F. P. Preparata, *On Finding the Maxima of a Set of Vectors*, J. ACM **22** (1975), no. 4, 469–476.
- [89] Jong-Hak Lee, *Analyses of Multiple Evidence Combination*, SIGIR, 1997, pp. 267–276.
- [90] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee, *Approaching the Skyline in Z Order*, VLDB, 2007, pp. 279–290.
- [91] Maurizio Lenzerini, *Data Integration: A Theoretical Perspective*, PODS, 2002, pp. 233–246.

- [92] ———, *Data Integration: A Theoretical Perspective*, 21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2002, pp. 233–246.
- [93] Lei Li and Ian Horrocks, *A Software Framework for Matchmaking based on Semantic Web Technology.*, WWW, 2003, pp. 331–339.
- [94] Yong Li, Sen Su, and Fangchun Yang, *A Peer-to-Peer Approach to Semantic Web Services Discovery*, ICCS (4), 2006, pp. 73–80.
- [95] Leonid Libkin, *Data Exchange and Incomplete Information*, 25th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 2006, pp. 60–69.
- [96] David Lillis, Fergus Toolan, Rem W. Collier, and John Dunnion, *ProbFuse: A Probabilistic Approach to Data Fusion*, SIGIR, 2006, pp. 139–146.
- [97] Dekang Lin, *An Information-Theoretic Definition of Similarity.*, ICML, 1998, pp. 296–304.
- [98] Sergio Luján-Mora, Panos Vassiliadis, and Juan Trujillo, *Data Mapping Diagrams for Data Warehouse Design with UML*, ER, 2004, pp. 191–204.
- [99] Federica Mandreoli, Riccardo Martoglia, Simona Sassatelli, and Wilma Penzo, *SRI: Exploiting Semantic Information for Effective Query Routing in a PDMS*, WIDM, 2006, pp. 19–26.
- [100] Frank Manola and Eric Miller, *RDF Primer*, W3C Recommendation, W3C, 2004.
- [101] Jose-Norberto Mazón and Juan Trujillo, *Enriching Data Warehouse Dimension Hierarchies by Using Semantic Relations*, BNCOD, 2006, pp. 278–281.
- [102] Jose-Norberto Mazón, Juan Trujillo, Manuel Serrano, and Mario Piattini, *Applying MDA to the Development of Data Warehouses*, ACM 8th International Workshop on Data Warehousing and OLAP, 2005, pp. 57–66.
- [103] Deborah L. McGuinness and Frank van Harmelen, *OWL Web Ontology Language Overview*, W3C Recommendation, W3C, Feb. 2004, <http://www.w3.org/TR/2004/REC-owl-features-20040210>.
- [104] Microsoft Data Transformation Services, <http://www.microsoft.com/sql/prodinfo/features>.
- [105] Microsoft Popfly, <http://www.popfly.com>.
- [106] George A. Miller, *WordNet: A Lexical Database for English*, Commun. ACM **38** (1995), no. 11, 39–41.
- [107] Mark H. Montague and Javed A. Aslam, *Condorcet Fusion for Improved Retrieval*, ACM CIKM, 2002, pp. 538–548.
- [108] Daniel L. Moody and Mark A. R. Kortink, *From Enterprise Models to Dimensional Models: a Methodology for Data Warehouse and Data Mart Design.*, 2nd International Workshop on Design and Management of Data Warehouses, 2000, pp. 5.1–5.12.

- [109] Luís Mota and Luís Botelho, *OWL Ontology Translation for the Semantic Web*, Proceedings of the Semantic Computing Workshop in conjunction with WWW, 2005.
- [110] MySpace, <http://www.myspace.com>.
- [111] Tapio Niemi, Santtu Toivonen, Marko Niinimäki, and Jyrki Nummenmaa, *Ontologies with Semantic Web/Grid in Data Integration for OLAP*, Int. J. Semantic Web Inf. Syst. **3** (2007), no. 4, 25–49.
- [112] Natalya Fridman Noy and Mark A. Musen, *PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment*, AAAI/IAAI, 2000, pp. 450–455.
- [113] Oracle Warehouse Builder, <http://www.oracle.com/technology/products/warehouse>.
- [114] Orkut, <http://www.orkut.com>.
- [115] OWL-S Service Retrieval Test Collection (OWLS-TC), <http://www-ags.dfki.uni-sb.de/klusck/owls-mx>.
- [116] Jeff Z. Pan and Ian Horrocks, *OWL-Eu: Adding customised datatypes into owl*, ESWC, 2005.
- [117] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara, *Importing the Semantic Web in UDDI.*, WES, 2002, pp. 225–236.
- [118] ———, *Semantic Matching of Web Services Capabilities.*, ISWC, 2002, pp. 333–347.
- [119] Massimo Paolucci, Katia P. Sycara, Takuya Nishimura, and Naveen Srinivasan, *Using DAML-S for P2P Discovery*, ICWS, 2003, pp. 203–207.
- [120] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger, *Progressive Skyline Computation in Database Systems*, TODS **30** (2005), no. 1, 41–82.
- [121] George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou, *Design Metrics for Data Warehouse Evolution*, ER, 2008, pp. 440–454.
- [122] Giorgos Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou, *Policy-regulated Management of ETL Evolution*, J. Data Semantics, (to appear).
- [123] Peter F. Patel-Schneider and Ian Horrocks, *OWL 1.1 Web Ontology Language*, W3C Member Submission, W3C, Dec. 2006.
- [124] Jian Pei, Ming Hua, Yufei Tao, and Xuemin Lin, *Query Answering Techniques on Uncertain and Probabilistic Data: tutorial summary*, SIGMOD Conference, 2008, pp. 1357–1364.
- [125] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan, *Probabilistic Skylines on Uncertain Data*, VLDB, 2007, pp. 15–26.
- [126] Cassandra Phipps and Karen C. Davis, *Automating Data Warehouse Conceptual Schema Design and Evaluation*, 4th International Workshop on Design and Management of Data Warehouses, 2002, pp. 23–32.

- [127] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Mauricio A. Hernández, and Ronald Fagin, *Translating Web Data*, 28th International Conference on Very Large Data Bases, 2002, pp. 598–609.
- [128] Franco P. Preparata and Michael I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag New York, Inc., 1985.
- [129] Erhard Rahm and Philip A. Bernstein, *A Survey of Approaches to Automatic Schema Matching*, VLDB J. **10** (2001), no. 4, 334–350.
- [130] Random Dataset Generator for SKYLINE Operator Evaluation, <http://randdataset.projects.postgresql.org>.
- [131] Jinghai Rao and Xiaomeng Su, *A Survey of Automated Web Service Composition Methods*, SWSWPC, 2004, pp. 43–54.
- [132] Mike Reape and Chris Mellish, *Just What is Aggregation Anyway*, European Workshop on Natural Language Generation (EWNLG), 1999.
- [133] Philip Resnik, *Using Information Content to Evaluate Semantic Similarity in a Taxonomy.*, IJCAI, 1995, pp. 448–453.
- [134] Oscar Romero and Alberto Abelló, *Automating Multidimensional Design from Ontologies*, DOLAP, 2007, pp. 1–8.
- [135] Grzegorz Rozenberg (ed.), *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, World Scientific, 1997.
- [136] Vincent Schickel-Zuber and Boi Faltings, *OSS: A Semantic Similarity Function based on Hierarchical Ontologies.*, IJCAI, 2007, pp. 551–556.
- [137] Mario T. Schlosser, Michael Sintek, Stefan Decker, and Wolfgang Nejdl, *A Scalable and Ontology-Based P2P Infrastructure for Semantic Web Services*, P2P Computing, 2002, pp. 104–111.
- [138] Cristina Schmidt and Manish Parashar, *A Peer-to-Peer Approach to Web Service Discovery*, WWW **7** (2004), no. 2, 211–229.
- [139] Amit Sheth, Cory Henson, and Satya Sahoo, *Semantic Sensor Web*, IEEE Internet Computing **12** (2008), no. 4, 78–83.
- [140] Pavel Shvaiko and Jérôme Euzenat, *A Survey of Schema-Based Matching Approaches*, J. Data Semantics IV (2005), 146–171.
- [141] Luo Si and Jamie Callan, *CLEF 2005: Multilingual Retrieval by Combining Multiple Multilingual Ranked Lists*, Proceedings of the 6th Workshop of the Cross-Language Evaluation Forum, 2005, pp. 121–130.
- [142] Alkis Simitsis, *Mapping Conceptual to Logical Models for ETL Processes*, ACM 8th International Workshop on Data Warehousing and OLAP, 2005, pp. 67–76.
- [143] Alkis Simitsis, Dimitrios Skoutas, and Malú Castellanos, *Natural Language Reporting for ETL Processes*, DOLAP, 2008, pp. 65–72.

- [144] Alkis Simitsis, Panos Vassiliadis, and Timos K. Sellis, *State-Space Optimization of ETL Workflows*, IEEE Trans. Knowl. Data Eng. **17** (2005), no. 10, 1404–1419.
- [145] David E. Simmen, Mehmet Altinel, Volker Markl, Sriram Padmanabhan, and Ashutosh Singh, *Damia: Data Mashups for Intranet Applications*, SIGMOD, 2008, pp. 1171–1182.
- [146] Dimitrios Skoutas, Verena Kantere, Alkis Simitsis, and Timos Sellis, *Ontology-Based Data Sharing in P2P Databases*, SWDB-ODDIS, 2007, pp. 117–137.
- [147] Dimitrios Skoutas, Dimitris Sacharidis, Verena Kantere, and Timos Sellis, *Efficient Semantic Web Service Discovery in Centralized and P2P Environments*, International Semantic Web Conference, 2008, pp. 583–598.
- [148] Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, Verena Kantere, and Timos Sellis, *Top-k Dominant Web Services Under Multi-Criteria Matching*, EDBT, 2009.
- [149] Dimitrios Skoutas, Dimitris Sacharidis, Alkis Simitsis, and Timos Sellis, *Serving the Sky: Discovering and Selecting Semantic Web Services through Dynamic Skyline Queries*, ICSC, 2008, pp. 222–229.
- [150] Dimitrios Skoutas and Alkis Simitsis, *Designing ETL Processes Using Semantic Web Technologies*, DOLAP, 2006, pp. 67–74.
- [151] ———, *Flexible and Customizable NL Representation of Requirements for ETL processes*, NLDB, 2007, pp. 433–439.
- [152] ———, *Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data*, Int. J. Semantic Web Inf. Syst. **3** (2007), no. 4, 1–24.
- [153] Dimitrios Skoutas, Alkis Simitsis, and Timos Sellis, *A Ranking Mechanism for Semantic Web Service Discovery*, IEEE SCW, 2007, pp. 41–48.
- [154] ———, *Ontology-driven Conceptual Design of ETL Processes Using Graph Transformations*, JoDS Special Issue on "Semantic Data Warehouses" (2009).
- [155] Naveen Srinivasan, Massimo Paolucci, and Katia P. Sycara, *An Efficient Algorithm for OWL-S Based Semantic Search in UDDI.*, SWSWPC, 2004, pp. 96–110.
- [156] Kunwadee Sripanidkulchai, Bruce M. Maggs, and Hui Zhang, *Efficient Content Location Using Interest-Based Locality in Peer-to-Peer Systems*, INFOCOM, 2003.
- [157] Steffen Staab and Heiner Stuckenschmidt (eds.), *Semantic Web and Peer-to-Peer*, Springer-Verlag, 2006.
- [158] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi, *Efficient Progressive Skyline Computation*, VLDB, 2001, pp. 301–310.

- [159] Chunqiang Tang, Zhichen Xu, and Sandhya Dwarkadas, *Peer-to-peer information retrieval using self-organizing semantic overlay networks*, SIGCOMM, 2003, pp. 175–186.
- [160] The TPC-H benchmark, <http://www.tpc.org/tpch>.
- [161] TREC, <http://trec.nist.gov>.
- [162] Juan Trujillo and Sergio Luján-Mora, *A UML Based Approach for Modeling ETL Processes in Data Warehouses*, ER, 2003, pp. 307–320.
- [163] Amos Tversky, *Features of Similarity*, Psychological Review, vol. 84, 1977, pp. 327–352.
- [164] Yannis Tzitzikas and Jean-Luc Hainaut, *How to Tame a Very Large ER Diagram (Using Link Analysis and Force-Directed Drawing Algorithms)*, ER, 2005, pp. 144–159.
- [165] Panos Vassiliadis, Alkis Simitzis, Panos Georgantas, Manolis Terrovitis, and Spiros Skiadopoulos, *A generic and customizable framework for the design of ETL scenarios*, Inf. Syst. **30** (2005), no. 7, 492–525.
- [166] Panos Vassiliadis, Alkis Simitzis, and Spiros Skiadopoulos, *Conceptual Modeling for ETL Processes*, DOLAP, 2002, pp. 14–21.
- [167] Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller, *METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services*, Inf. Tech. and Manag. **6** (2005), no. 1, 17–39.
- [168] Christopher C. Vogt and Garrison W. Cottrell, *Fusion Via a Linear Combination of Scores*, Information Retrieval **1** (1999), no. 3, 151–173.
- [169] Spyros Voulgaris, Anne-Marie Kermarrec, Laurent Massoulié, and Maarten van Steen, *Exploiting Semantic Proximity in Peer-to-Peer Content Searching*, FTDCS, 2004, pp. 238–243.
- [170] Le-Hung Vu, Manfred Hauswirth, and Karl Aberer, *Towards P2P-Based Semantic Web Service Discovery with QoS Support*, BPM Workshops, 2005, pp. 18–31.
- [171] Graham Wilcock, *Talking OWLs: Towards an Ontology Verbalizer*, Human Language Technology for the Semantic Web and Web Services in conjunction with ISWC, 2003, pp. 109–112.
- [172] Graham Wilcock and Kristiina Jokinen, *Generating Responses and Explanations from RDF/XML and DAML+OIL*, Knowledge and Reasoning in Practical Dialogue Systems in conjunction with IJCAI, 2003.
- [173] Yahoo Pipes, <http://pipes.yahoo.com/pipes>.
- [174] Man Lung Yiu and Nikos Mamoulis, *Efficient Processing of Top-k Dominating Queries on Multi-Dimensional Data*, VLDB, 2007, pp. 483–494.