AN ALGEBRAIC FRAMEWORK FOR THE INTEROPERATION OF ONTOLOGIES

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Prasenjit Mitra
August 2004

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Gio Wiederhold
(Principal Adviser)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Mark A. Musen

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Natasha F. Noy

Approved for the University Committee on Graduate Studies:

Abstract

With the advent of the World-Wide Web, end-users - individuals and organizations who access information from the Web - have access to large amounts of information. In most cases, a single information source cannot satisfy the needs of end-users. The end-users need to compose information from multiple sources to adequately satisfy their requirements and to get answers to their queries. Today, the end-users are responsible for manually composing information. Ideally, the end-user should have tools that alleviate the burden of manual composition by composing information from diverse sources automatically.

Before composing information from diverse sources, however, the composer needs to resolve any semantic heterogeneity among them. Although considerable progress has been made in resolving heterogeneity related to operating systems, and networking protocols, little progress has been made towards solving problems that arise due to semantic heterogeneity. Such heterogeneity arises due to mismatches in the semantics of terms used in the information sources. Mismatches occur because different organizations have different business needs. Attempts at achieving agreement on the semantics of terms and maintaining such agreement may be successful in small domains but is futile for large ones.

In order to resolve the semantic heterogeneity of information across information sources, one must understand the semantics associated with the information. Despite advances in technology, machines find it difficult, if not impossible, to decipher the exact semantics associated with information. A popular approach to solving the problem of unspecified or under-specified semantics is to use meta-data structures like *ontologies* that can be processed by machines. Information sources increasingly come with ontologies that explicitly define the terms used in the sources and their relationships.

Large information sources use large vocabularies. Large vocabularies, in turn, result in large ontologies. Often, creating an ontology from scratch is unnecessary and more expensive than constructing an ontology by reusing existing ontologies. A large number of

ontologies are available and can be reused. Creators of ontologies need to perform a set of common operations in order to reuse existing ontologies and create new ones from them. In this work, I present an *Ontology-Composition Algebra* that consists of a set of basic operators that can be used to manipulate ontologies. The algebraic operators can be used to declaratively specify how to compose new, derived ontologies from the existing source ontologies. A declarative specification allows easy replay of the composition task when source ontologies change and the change needs to be propagated to the derived ontologies. If there does not exist a means to quickly and easily update the derived ontologies when the source ontologies change, the derived ontologies supply stale and often inconsistent information to its clients.

As argued above, before multiple ontologies can be composed, the semantic heterogeneity among these ontologies must be resolved and a set of *articulation rules* established that specify the correlation among related concepts across source ontologies. I have decoupled the algebraic machinery that is used to manipulate ontologies from the component that derives the semantic correspondence among ontologies to create two distinct components: (1) *Articulation-rule generating functions* generate *articulation rules* among pairs of ontologies, and (2) Algebraic operators use the articulation rules to compose the source ontologies.

Articulation-rule generation functions can be implemented as semi-automatic subroutines that deploy heuristic algorithms to *articulate* ontologies. Empirical evidence shows that semi-automatic articulation generating functions can be implemented and form an useful component of information composition tools.

The Ontology-Composition Algebra has unary and binary operations that enable an ontology composer to select interesting portions of ontologies and compose them.

I characterize the properties of the algebraic operators. I have identified the necessary and sufficient conditions for the optimization of composition tasks. Not surprisingly, most of these properties depend upon properties of the articulation generation function employed to resolve the semantic heterogeneity among the ontologies being composed.

This work is a first-step in implementing an interoperation system that tackles semantic heterogeneity among information sources by using ontologies to explicate the semantic relationships in the information sources. The algebraic machinery introduced provides a framework for the composition of ontologies and the declarative specification of composition tasks makes future maintenance easier when ontologies change..

Acknowledgments

I thank my advisor, Professor Gio Wiederhold, for accepting me as his student and giving me the opportunity to learn a lot from him and the database group at Stanford. He gave me the freedom to decide what research problems I wanted to work on but he was always there to guide me when I needed it. He always answered my emails where I asked various questions ranging from those on scientific research in general, and on publication of scientific ideas and results, to specific ones pertaining to my research.

I thank the members of my reading committee Professor Mark Musen, Doctor Natasha Noy for their comments on my work which helped to make it better.

I would like to thank Professor Jeff Ullman who introduced me to exciting new problems in the field and whose words of advice and numerous reviews of my badly written papers encouraged me to strive to be a better writer. I have also learnt a lot from working closely with Doctor Stefan Decker, Professor Chen Li and Professor Foto Afrati on research topics of mutual interest.

I am specially grateful to Dr. Jan Jannink who introduced me to the general topic of semantic heterogeneity and explained to me the various problems in enabling interoperation among information systems. I also thank my colleagues Srinivasan Pichai, Danladi Verheijen, Shrish Agarwal, and Alex Carobus with whom I have had numerous discussions about problems in heterogeneous systems.

More generally, I believe that my interactions with the database group at Stanford went a long way in making me a better student and researcher. When the work was at its early stages, I got good feedback on what had already been done before and how to improve my work. When the work was at its later stages, I got numerous good suggestions on how to present it better, so much so that I was very relaxed when actually presenting it outside Stanford at various conferences and institutions.

I am greatly indebted to my good friend, Professor Supratik Chakraborty, currently at

the Indian Institute of Technology, Mumbai, who had more faith in my abilities than I did and greatly encouraged me to apply to Stanford University and pursue a Ph.D. degree in Electrical Engineering Supratik painstakingly spared a lot of his time to answer questions and quell my fears about post-graduate education at Stanford.

At Stanford, I thank Ms. Marianne Siroker and Ms. Sarah Weden for the tireless efforts they undertake to make our life easier.

I must acknowledge all my professors at Indian Institute of Technology, Kharagpur and at the University of Texas at Austin for providing me the basic tools to pursue graduate research. Specifically, I will never forget the help of Professor Partha Pratim Chakraborty, Professor Subhas Chandra Dey Sarkar, and Professor Robert Van de Geijn.

I am grateful for all the help and support I received from my friends and relatives. I am indebted to Subir Chandra Ray, Rebecca Biswas, Indradeep Ghosh and numerous others for encouraging me and for their friendship that helped me get over difficult times. I am especially thankful to my father-in-law, the late Dr. Mrinal Ranjan Ghosh, for the words of encouragement and interest he had in my work. I also thank my mother-in-law, Mrs. Ratna Ghosh and aunt-in-law Mrs. Pratima Ghosh for their help and support during the preparation of my dissertation and its defense.

I dedicate this work jointly to my parents and my wife. I am greatly indebted to my mother Mina Mitra for instilling in me the discipline to work dilligently. My father Pankaj Kumar Mitra always demanded the best of me and encouraged me to pursue my goals in life.

Last, but by no means the least, I would never have achieved what I have without the love of my wife Rini Ghosh. Rini encouraged me to apply to Stanford and supported me during my graduate studies. She never complained even though I could not give her the time I knew she wanted to spend with me. She took care of everything so that I could solely concentrate in my research. She consoled and encouraged me on the days my experiments failed and finally pushed me to overcome my laziness at times I was procrastinating on writing my thesis. She put on a smiling face and heard me out at times even though I later realized I was boring her to death with my enthusiastic and long monologues about my research. I cannot imagine completing graduate school without her help.

This research was partially supported by the Scalable Knowledge Computing and the OntoAgents projects by the AFOSR New World Vistas program and the DARPA DAML program respectively.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Heterogeneity Across Information Systems

The World-Wide-Web has enabled people to publish and access information easily. As a result, the Web has a very large number of sources of information (web pages and their underlying data repositories). *End-users* — people who access the World-Wide-Web — have access to vast amounts of information published on the Web. The end-users are interested in extracting information from this huge repository and deriving answers to queries. Having good and comprehensive information is essential for effective decision making. They can do so by *searching* for information across millions of web pages using keywords [BP98]. However, often no single information source can satisfy the needs of end-users. The end-users have to sift through a large number of web pages — the results of a web-search — and then manually extract, relate and compose the information they need. Relating information from multiple sources is a hard problem because the sources of information are often inconsistent.

1.1.1 Inconsistency Among Information Sources

Heterogeneity among information sources results from their differences in hardware, operating systems, network interfaces, access protocols, query languages, and communication and messaging protocols. While there has been substantial progress on resolving these sources of heterogeneity among information sources, little progress has been made in resolving heterogeneity arising from the differences in semantics of the information sources. Creators of information sources use terms with specific intended meanings in their own contexts. We

will refer to this meaning as *semantics* of terms throughout the thesis. Semantic heterogeneity arises because different sources have different vocabularies and semantics. Different creators use the same term with different meanings. Thus, the semantics of information from different information sources is different. Very often, two different sources use the same term but associate different meanings to it (hyponymy), while different terms are used in two sources to mean the same thing (synonymy).

Often, creators of information sources differ on how to classify objects. A creator of an information source structures the information in the source such that the interests of the creator are best satisfied. The creator is less concerned about the use of the information source by a secondary system that uses the information since catering to the needs of such a system is outside the scope of the creator's responsibilities.

The Problem of Synonyms

In many cases, the creators also use different terms to refer to the same real-world object. For example, one uses the terms *truck* in the United States of America and the term *lorry* in the United Kingdom, respectively, to represent similar vehicles.

Consider the case of an American business owner who wants to expand her business by opening an unit in the U.K. that supplies goods to British customers. She seeks transportation options for her goods. Without knowing that a lorry is a truck, she will miss information about valid transportation options available to her.

Differences of Scope

Often, the author of an information source uses the same term to denote multiple meanings. Even if not entirely different, the scopes of the intended meaning of a term differs. Simple scope differences that create troublesome inconsistencies abound.

Consider the example of a university, whose *Personnel* department and *Payroll* department use the term *employee* with differing semantics. The *Payroll* department considers a graduate research assistant, who receives a monthly salary, a *employee*. The *Personnel* department classifies a graduate research assistant as a *student* and only the faculty and staff of the university as *employee*s. Each department has designed its information source depending upon its needs. Not all graduate students are employees and not all assistants are graduate students.

Lack of agreement among information sources results in inconsistencies. A highway police department, an architect, and a government registration agency defined the term *Vehicle* differently. Registration in California covers houseboats, which are excluded from the concern of the police and architects. The California Department of Motor Vehicles (DMV) uses the term *Vehicle* to include vehicles that move on land as well as on water (like *Watercraft*) whereas the California Highway Patrol (CHP) does not use the term *Vehicle* to refer to vehicles that move on water. Since the CHP is not mandated to enforce the law on California's water bodies, it not interested in watercraft. Each of the two organizations has defined the term *Vehicle* according to the needs of its own organization. Such problems are common but rarely recognized until composition is affected.

In order to compose information obtained from the DMV and that obtained from the CHP, one has to be cognizant of the differences of scope of the term *Vehicle* as used by the two different organizations. The composer needs to know the exact specification of the semantics of the terms used in the two different information sources used by the two organizations to differentiate between the two overlapping semantics of *Vehicle* and avoid mismatch-related inconsistencies.

1.1.2 Reasons for Semantic Heterogeneity Among Information Sources

In this section, we describe the reasons for semantic heterogeneity among information sources. They are as follows:

1. Autonomy of information sources.
2. Variations due to linguistic biases.
3. Different publishing objectives.

Sources are autonomous, and hence, there is no reason for them to be mutually consistent. Publishers of information have different viewpoints and different objectives for publishing the information. In the absence of a global mandate, agreed upon by all producers and consumers of information, publishers disseminate information, whose semantics conforms to their different viewpoints, on the Worldwide Web.

In domains where there are a large number of publishers with different needs, establishing an agreement between all publishers is impossible. Establishing global consistency will involve protracted negotiations. The cost of reaching an agreement in any large domain often

outweighs the benefits to the publishers. Only the end-users combining information from multiple sources stand to gain. Besides, if the domain is growing or changing the negotiated vocabulary will have to be frequently updated. Updating the agreed-upon vocabulary could potentially again require lengthy negotiations.

1.1.3 The Problem with Semantic Heterogeneity

Whenever the information that is required to satisfy applications is inconsistent — since it is obtained from inconsistent information sources — the inconsistency needs to be rectified before an end-user can use the information. Instead of manually visiting several websites and determining the price of a book, an end-user prefers an automated search engine that searches multiple bookseller's websites and then presents the prices and other information regarding the book (like shipping and handling information) requested by the end-user on one summary page. Ideally, an end-user should have access to an automated tool that will reduce the manual labor involved in fully integrating the information. The automated tool should gather and compose the information required by the end-user from multiple sources and present the composed information to the end-user in the format the end-user desires.

When applications defer wholesale composition closer to the time of use, the end-user must use the available tools to bring the sources together. For researchers this task is natural; they do not expect that linkages among concepts have been provided to them. For business users, as envisaged for the semantic web, that deferred approach is clearly impractical; there is neither sufficient time nor the broad expertise at the point of need. Besides, in the case where there are many end-users with similar applications, e.g., in a travel-reservation application to book flight tickets and hotel rooms. the end-users have to repeat the same task of linking concepts across sources before they can use the sources. If the desired semantics of the applications, say as in several travel-reservation applications, is simple, well defined and match well it makes more sense to establish the linkages between the concepts correctly once and for all. The task of the end-users is then greatly simplified since they do not have to understand the semantics of the sources and bring them together, but can simply log-on to the application and have all the information from the different sources available to them.

1.1.4 A Hard Problem to Solve

Unfortunately, the composition of information from diverse, autonomous information sources is a hard problem. It is not just a matter of matching words. For example, the organization of class-subclass hierarchies in different sources are substantially different. This difference in semantics results in inconsistencies when information is composed from heterogeneous information sources. To achieve consistent information by composing information from heterogeneous information sources, the differences between the class-subclass hierarchies in the different sources must be identified. After identifying the differences, rules must be established that allow consistent composition of information.

In the next section, I define a few terms that is used in the rest of the dissertation and describe the problem that I address.

1.2 Scope of the Dissertation

In this work, I refer to databases, knowledge bases, and other sources of information including information on the World-Wide-Web collectively as *information sources*. The *creator* of an information source decides the contents, and the structure of the information that is published.

The *owner* of an information source owns the information and controls how the information can be accessed. The *publisher* of the information source publishes the information and has the onus of deciding how the information is published. Oftentimes, especially in scenarios on the web, the role of creator, publisher, and owner are all played by a single individual or organization. An *end-user* is a person who seeks to use information sources to access information (as opposed to those who developed or support the information sources). An end-user or a group of end-users with similar objectives can deploy an *application* — a program that is built to satisfy their requirements. I use the term *expert* to refer to a person who understands the semantics of relevant parts of information sources from which information needs to be composed.

In this work, I use the phrase, *integration of information sources*, to refer to an operation that takes multiple information sources and composes them together to create one new, unified information source. I use the phrase, *interoperation among information sources*, to refer to an operation that retrieves information from multiple information sources and composes them. As a shorthand, I will refer to the phrase simply using the term “interoperation”

when the context makes it clear.

1.2.1 Using Information from Multiple Information Sources

An operation using information from multiple sources can only be enabled meaningfully based on an understanding of the semantics and the structure of the different information sources. If an application wants to allow end-users to compose information from multiple sources, it needs semantic-correspondence rules among the terms used in those sources. I refer to these rules as *articulation rules* and a tool that generates the articulation rules as an *articulation rule generator* in the rest of the dissertation. The generator of the articulation rules between terms used in different information sources must know the exact semantics of the terms used in the sources.

1.2.2 Use of Ontologies to Specify Semantics

Unfortunately, the semantics of terms in information sources is not very well documented. Where documentation is available, it is often written in a natural language and cannot be interpreted by a computer program. Despite substantial advances made in the field of natural-language processing, computers still remain unable to interpret the full semantics of natural languages. Therefore, even automated agents that use natural-language processing remain inadequate to automatically resolve semantic heterogeneity among information sources with high precision. Besides, even human experts sometimes misinterpret the semantic intentions of the creator of an information source if those intentions are not well understood.

The World-Wide-Web community has recognized that there is not sufficient semantics available along with information sources that are published on the Web. In order to remedy the situation, a vision of the *Semantic Web* has been proposed [HBLM01]. In order to achieve that vision, there has been a move towards using markup languages like XML [xml99], DAML [damb], or OWL [owl04] that contain meta-data that specifies the semantics of the data contained in documents. Marking up documents with more semantic information than provided in HTML [htm] is a step in the right direction. However, semantic markup, as is available today, is still inadequate to satisfy the requirements of many applications. The semantic information provided by information markup is quite rudimentary when compared to the semantic information provided by a carefully constructed

ontology. Gruber defines an ontology as "a specification of a conceptualization" [Gru93]. Ontologies can provide precise definitions of concepts and objects used in an information source.

The inadequacy of semantic markups is acute especially when end-users cannot tolerate errors, e.g., in most business applications. Errors in information used can be very costly to businesses. For example, the home address and the office address of a customer needs to be clearly identifiable. A customer will not accept delivery meant for his office at his home or vacation cottage. Such an error might require a costly re-route or result in missing delivery deadlines.

To automatically compose information from diverse sources well-defined ontologies that define and explicate the terms and relationships used in information sources is needed. The tool matches the terms in the ontologies associated with the information sources and establishes the semantic correspondence between terms across the ontologies in order to enable interoperation among the information sources.

Cognizant of the need for specifications, increasingly, publishers of information are also publishing *ontologies* that contain metadata intended to capture the semantics of the existing information sources. Publishers are also building new information sources that use only terms and relationships defined in an associated ontology.

1.2.3 Composition of Ontologies

When information has to be obtained from diverse sources, it becomes necessary to compose ontologies (corresponding to the diverse sources) that can serve the application domain. Only in very small domains does one ontology satisfy the needs of the various organizations in the domain. Furthermore, if an application spans multiple domains, the application needs to use terms from multiple ontologies. Instead of creating a new ontology right from scratch containing the terms that the application needs, it is cheaper to create an ontology for an application by selecting and reusing portions of existing ontologies and composing them to form a new application ontology. Reusing existing ontologies often results in significant savings. Applications need the ability to select portions of existing matched ontologies that are of interest, compose them, and create new ontologies [ME].

1.2.4 Synchronizing Composed Ontolgies with Changes in their Sources

The information in information sources change rapidly. Organizations introduce new terminology or expand the terminology they use in the information they publish. Thus, they have an ongoing need to update the ontologies associated with these information sources.

When such an update occurs, any ontology derived from the source ontologies also needs to change. Updating such derived ontologies in an *ad hoc* manner increases the cost of maintenance. More importantly, there is always the chance that due to human errors of omission the derived ontologies are not updated along with the source ontologies. As a result, the derived ontology contains stale terminology. An ideal solution to the synchronization problem between source and derived ontologies when source ontolgies are updated is to have a tool that automatically triggers a systematic update to the derived ontology when its sources change [OSSM98]. Such a tool requires an algebraic foundation based upon which the updates can be systematically handled.

1.2.5 The Central Questions

In this dissertation, I attempt to solve the *Ontology Composition Problem*. Informally, the Ontology Composition Problem is defined as follows: Given a set of ontologies O, how can we select portions of existing ontologies from O and compose them to create a new ontology?

Any attempt to solve the ontology composition problem must answer the two following two questions satisfactorily. I believe these questions are fundamental ones that need an answer before one can establish an interoperation system for information sources. They are:

- How can we build a semi-automatic toolkit that generates articulation rules between two source ontologies?
- Can we formally design an ontology composition system that
    1. enables systematic composition of existing (source) ontologies to derive new ontologies based on the articulation rules and
    2. helps in automatically updating derived ontologies when source ontologies change?

I outline my approach to addressing these problems in the next section. I have the former question in Chapter 5, where I describe SKAT — the Semantic-Knowledge Articulation-rule-generation Toolkit that we have built. The latter question is addressed in Chapter 6, where I describe the algebraic basis of an ontology composition system — OntoComp.

1.3 A Solution to the Ontology Composition Problem

In this section, I outline the strategy that I have designed to address the Ontology Composition Problem and describe the setting for the dissertation. I have designed a tool-kit called ONION (ONtology compositION) that can be used to resolve semantic heterogeneity among ontologies using SKAT and compose existing ontologies to create new ones using OntoComp.

In this work, I consider only information sources whose semantics are specified precisely using ontologies. These ontologies are structures represented using graphs whose nodes are terms used in the information sources and edges define relationships among the terms. (for a formal definition and examples, see Chapter 4).

1.3.1 The Need for a Semi-Automatic Approach

On one hand, since ontologies can be fairly large, establishing articulation rules manually is a very expensive and laborious task. On the other hand, fully automating the process of generating articulation rules is also not feasible.

Problems Faced while Automating Ontology Matching

First, as pointed out earlier, there is the analog of the symbolic grounding problem [Har90]. Can the semantics of the terms used in a system be deduced from the information contained intrinsically in the system? Most ontologies have some descriptions of concepts in natural languages and various flavors of rule-based languages like first-order logic [Smu95], description logics [BCM+03], etc..

Second, the structure of an ontology varies significantly from one ontology to another. I will show in Chapter 5 that matching of ontologies based on structure alone produces very poor results. Furthermore, even though ontologies explicitly specify some of the semantics of the terms and their relationships, and some structural meta-data, the specifications are often incomplete or inadequate to satisfy the needs of the various applications that use them. Consider the scenario where the semantic information necessary for the application to meaningfully and accurately use information from an information source is not available. Without access to the required semantics of the concepts in the source ontologies, an automated articulation-rule generator cannot generate a precise and complete set of articulation rules. Also, if the complete semantics of concepts in ontologies need to be captured,

the ontologies need to be coded using more expressive languages. The more expressive a language is, the less tractable it is. Thus, in my opinion, building a totally automated articulation-rule generator is impossible [Rus12].

Expert-Aided Matching of Ontologies

End-users using a search engine still have to filter the useful information manually from the results of the search. End-users performing similar searches have to repeat the filtration step that is largely similar for similar searches. The person filtering out the necessary information from the search results must understand the semantics of the information in the information sources as well as the semantics and the purpose of the application the end-user has in mind. Incomplete understanding of either semantics results in costly errors. Oftentimes, individual end-users do not have the expertise to understand the semantics of the information sources. Nor do they have the ability to incur the effort and the costs involved in fully understanding the semantics an information source and then using information from them.

Thus, it is preferable to have an automated tool, aided by a human *expert*, set up the interopeation environment. Of course, the human expert must understand the semantics of the information sources as well as that of the application. The costs of deploying the expert can then be shared among multiple end-users or applications. In such a scenario, end-users can simply perform the search and get back exactly the information necessary for their applications. Keeping this scenario in mind, I have designed a semi-automated interoperation system.

In this setting, I assume that an expert knows the semantics of information across sources and with the help of a tool identifies the *articulation* (a term first used in [Guh91]) — the semantic linkages across the ontologies associated with the information sources. An expert will have to put in less effort to extract and compose information from information sources if a good automated tool is available than when the expert has no such tool available.

1.3.2 The Workflow of the Interoperation System

ONION has the following components:

1. Articulation-Rule Generator: SKAT generates articulation rules
2. Ontology-Composition Engine: OntoComp composes existing ontologies to create new ontologies using the articulation rules and an ontology-composition algebra

3. Query Processor: QueProc takes user queries and uses the source ontologies, the articulation rules and the information sources to answer the queries.

4. Ontology and Articulation Rule Repository: OntoStore stores articulation rules and newly created ontologies

Figure 1.1 shows the main components of the interoperation system and its interactions with the expert and the end-user. OntoComp takes in the source ontologies along with the algebraic expression specifying the composition task and generates new ontologies as defined by the algebraic specification. Similarly, the queries posed by the end-users might have terms from the source ontologies but this information has not been explicitly shown in the figure.

In the rest of the section, I will first outline the types of articulation rules used in Onion, and then I will briefly describe the components of the system.

1.3.3 Ontology Articulation Rules

An articulation of two ontologies consists of the terms in one ontology that have semantic correspondences in the other and vice versa, and the relationships among these terms.

Articulations are expressed as a set of rules, which I refer to as articulation rules. They can be expressed in any logic-based language. In this work, I have chosen the language of conjunctive queries to represent articulation rules because inference using conjunctive queries is computationally faster than inference using rules expressed in other languages, e.g., in full first-order logic. A conjunctive query is of the form:

$$h(\bar{X}) \ \rightarrow \ g_1(\bar{X}_1), \ldots, g_k(\bar{X}_k)$$

In each subgoal $g_i(\bar{X}_i)$, predicate g_i is a *base relation*, and every argument is either a variable or a constant.

The expressiveness of conjunctive queries is less than that of first-order logic — that is, all expressions in first-order logic do not have a semantically equivalent expression in the language of conjunctive queries. In order to keep the system tractable, I chose the language of conjunctive queries. In the next sub-section, I illustrate the types of articulation rules used in the system with an example.

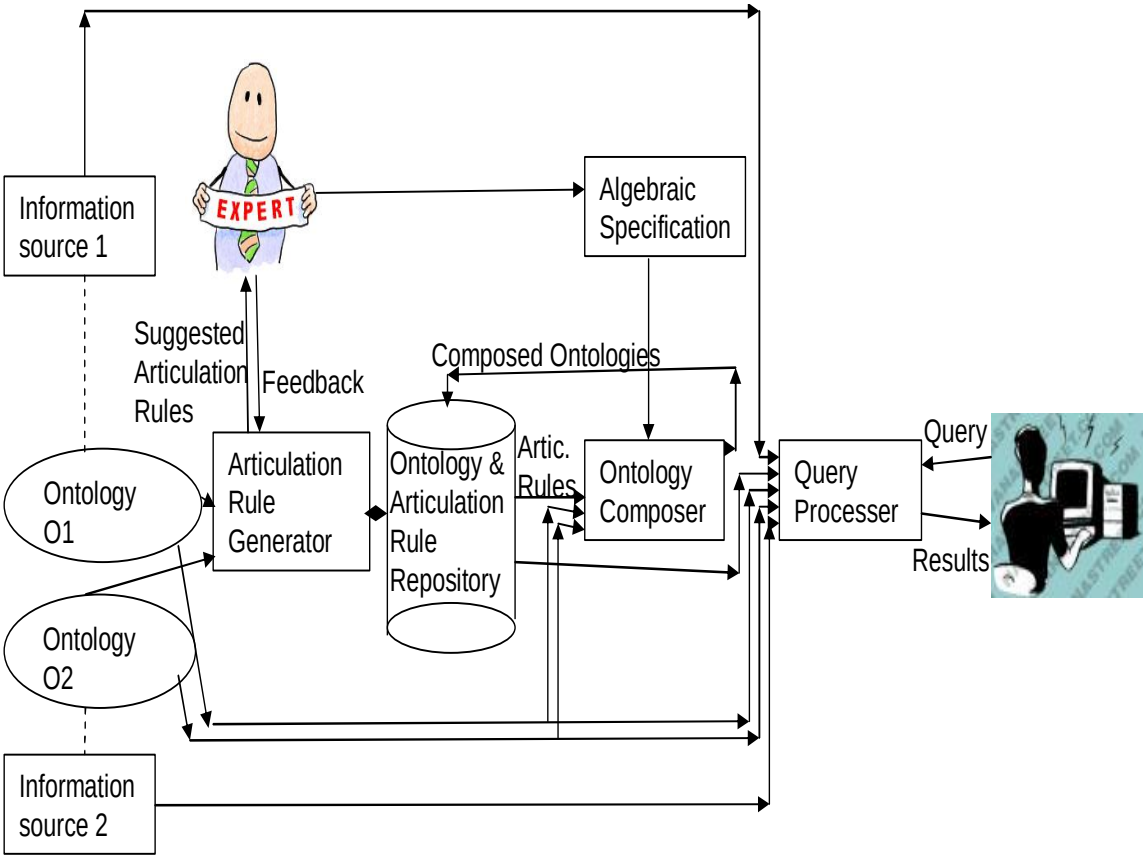
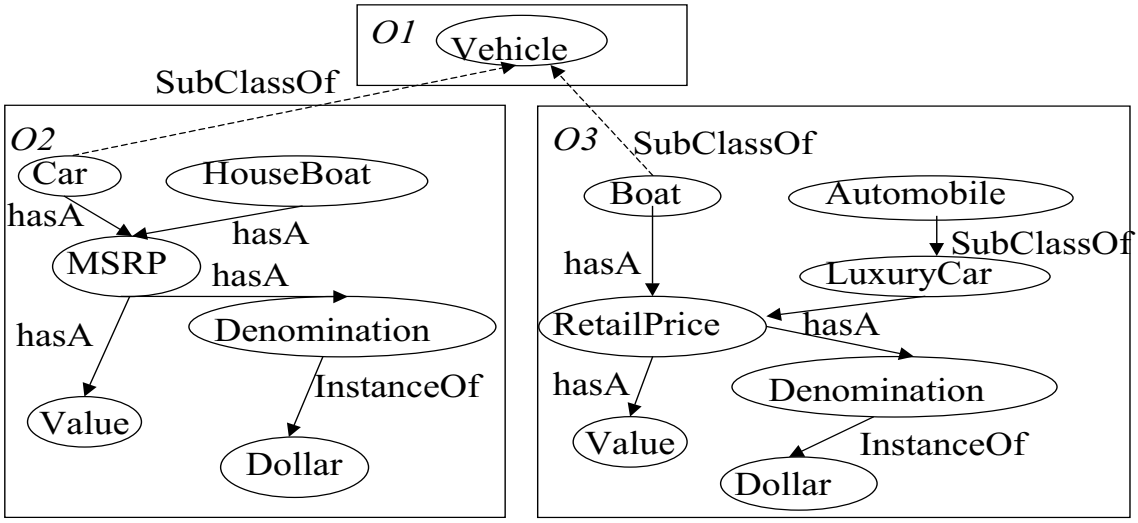


Figure 1.1: The Workflow of the Interoperation System



Articulation Rules:

true => (O2.Car SubClassOf O1.Vehicle)

(X InstanceOf O1.Car),(X hasA X.MSRP),(Y InstanceOf X.MSRP),
(Y hasA Y.Value), (Z InstanceOf Y.Value), (Y.Value > 40,000)
=> (X InstanceOf O3.LuxuryCar)

Figure 1.2: Ontologies and Articulation Rules

1.3.4 An Example

EXAMPLE 1.3.1 I introduce a running example in Figure 1.2, which I will use throughout the dissertation. *O1*, *O2*, and *O3* are the ontology graphs for three ontologies. I only show selected portions of the ontology graphs. A set of articulation rules that express the relationship between *O2.Car* and O1.Vehicle and a rule that expresses the relationship between *O1.Car* and O3.LuxuryCar is given below the figure. As a shorthand, we often write a rule of the form: $true \Rightarrow (Subj\ Prop\ Object)$ simply as $(Subj\ Prop\ Object)$.

SKAT generates the following additional articulation rules that are not shown in the figure:

```
(O1.HouseBoat SubClassOf O3.Boat)
(O1.Value Equals O3.Value)
(O1.Denomination Equals O3.Denomination)
(O1.Car SubClassOf O3.Automobile)
(O1.Dollar Equals O3.Dollar)
```

After these rules are generated, they can be used by an application to compose information from multiple sources and answer queries or perform tasks given by end-users. Without matching the ontologies and establishing the articulation rules as shown in the figure, the differences in vocabularies at the different information sources will make it difficult for information to be composed from them. Thus, for the successful adoption and realization of the Semantic Web, ontology matching and articulation technologies play a pivotal role. □

1.3.5 Articulation Rule Generation

Before one can compose ontologies, they need to be articulated (also referred to as matched or aligned). Articulation of ontologies establishes the semantic correspondence between concepts across the source ontologies. Various ontology matching and alignment techniques have been designed to articulate ontologies [NM00], [MFRW00], [DDH01]. I refer generically to these algorithms as *articulation-generation functions*. Each function takes in two ontologies and generates a set of articulation rules between them. An articulation-rule generator is the program that is an implementation of an articulation-rule-generation function, with the help of a human expert who is knowledgeable about the semantics of the two ontologies.

I have designed and implemented an articulation generator, SKAT (Semantic Knowledge Articulation rule generation Tool), that uses articulation-generation functions that are semi-automatic. That is, they have a manual as well as an automatic component. Though, in an ideal world, I would prefer to take an entirely automated approach to solving the problem of semantic interoperation, the precision achieved by using current technology mandates a semi-automatic approach especially when the tolerance for errors is low.

Ontology Formats

The ontologies associated with information sources are based on some existing, known vocabularies and data formats. To keep SKAT simple, I designed SKAT such that it does not handle input ontologies in any data format, but only understands the ONION ONTOLOGY FORMAT that I describe in Chapter 4. Native drivers and wrappers that translate an ontology from a native format to ONION's ontology format provide access to the source ontologies.

A Semi-Automated Approach

To establish articulation rules and constructing new ontologies for applications, the expert ontology composer uses two tools. First, the expert uses SKAT to match the terms in the ontologies and generate articulation rules automatically. Then, the human expert who knows the semantics of the two ontologies being articulated, verifies whether the articulation rules are correct and relevant to an application. The expert can accept, delete or modify the suggested articulation rules. SKAT logs the responses of the expert. In the future, when the articulation generator has to articulate similar ontologies, it can use the logs to generate articulation rules that are more accurate than those generated during the previous runs. The articulation rules that the expert has ratified are stored in ONTOSTORE. An ontology composer — an agent who wants to compose portions of existing ontologies and create new ones can use the articulation rules in the future to *compose* ontologies.

1.3.6 Ontology Composition: An Algebraic Approach

The composer of an ontology constructs the new ontology with a specific application (or a group of applications having the same semantics) in mind. The composer must be knowledgeable about the semantics of the source ontologies and the application in order to compose and create an ontology that is most suitable for the application. First, the composer selects concepts from ontologies of interest. He can use the ontology-composition algebra to specify the portions of existing source ontologies that he wants to compose. Then, he constructs an expression that expresses the composition task that needs to be performed using the selected portions of the source ontologies. The composition task is expressed using an ontology-composition algebra.

The ontology-composition algebra (see Chapter 6 for specification) comprises of unary and binary operators that operate on ontologies and produce new ontologies. The algebra provides a way to declaratively specify ontology composition tasks. The output of one operation is a *bona fide* ontology that the composer can compose with any other ontology to produce a new ontology.

The Maintenance Problem :

Let us assume that an expert expends the labor needed to construct a new ontology from existing ontologies manually or by writing specific programs and scripts that create the new ontology. Very soon this process has to be repeated when the source ontologies, on which the new ontology is based, is updated. The cost of recreating the new ontology using the individual primary sources, becomes prohibitive if there are frequent changes to the primary ontologies. Since it is costly to achieve and maintain global semantic information the approach of periodically recreating the same ontology is not scalable.

Advantages of an Algebraic Approach versus a Programmatic Approach

The primary advantage of having an algebraic framework is that it allows us to automatically update ontologies created from source ontologies when the sources change. As a secondary by-product, the algebraic characterization also allows us to optimize the composition tasks. The properties of the algebraic operators determine the types of optimization that can be enabled while composing the ontologies.

A declarative specification of the derivation process by which an ontology is created

allows easy replay of the derivation after the source ontologies change and the change needs to be propagated to the derived ontology. Besides, the properties of a declaratively specified composition task can be characterized more easily than those of a programatically specified one.

In our example shown in Figure 1.2, there are two choices. Either the ual.com and aa.com ontologies are composed into one airline ontology and then the hertz.com ontology is articulated with the airline ontology, or the hertz.com ontology can be articulated with each individual ontology. The former composition is more scalable because it performs one articulation with the hertz.com ontology instead of two. Furthermore, in the presence of multiple car-rental companies, the former strategy works even better since the number of articulation it avoids is even greater than if an ontology-composition tool employed the latter strategy.

In Chapter 6, I discuss the issues involved with optimization of the ontology-composition task and show how on the basis of an *ontology-composition algebra*, OntoComp optimizes tasks requiring ontology composition.

When an application seeks to compose information from multiple sources, it uses the new application ontology and the articulation rules to identify correspondences between concepts in the application ontology and similar concepts in other information sources. Using the articulation rules, the tool can compose required information from multiple sources to answer queries that the end-users had posed to the system.

Articulation rules generated by the articulation generation function form the basis for the composition of ontologies. The binary operators of the algebra depend upon the articulation rules generated by the articulation generation function. For example, an ontology composer seeking to intersect two ontologies must know which concept in one ontology is similar to which concepts in the other. Not surprisingly, therefore, the properties of the algebraic operators are determined by the articulation generation functions.

Though significant progress has been made in designing ontology matching and alignment techniques [MGMR02], [DDH01], [MBR01], [NM00], [MFRW00], to the best of my knowledge, no prior work has been done in identifying desirable properties of such functions, especially in settings where these algorithms are used for ontology composition. In this work, I identify and state desirable properties of such articulation generation functions.

1.3.7 The Query-Processing Engine

As discussed in Section 1.2.1, a query-processing engine uses ontology articulation rules and existing ontologies to translate queries. The translated queries use the vocabularies of the individual information sources and can be answered by them. The question of answering queries using articulation rules and ontologies is beyond the scope of the thesis. The interested reader is referred to my work in [Mit01, ALM02, ALM04].

1.4 The Hypotheses and the Organization of the Dissertation

In this section, I specify the hypotheses that I verify in the dissertation and describe the organization of the dissertation.

1.4.1 The Hypotheses

In this dissertation, I verify the following hypotheses:

Hx. 1 An automated articulation generator is able to partially match concepts, based on the structure and terminology used in the ontologies they appear in, thereby resolving semantic heterogeneity among the ontologies.

Hx. 2 In comparison to a totally manual approach of resolving semantic heterogeneity, automated articulation generators reduce the amount of time that needs to be put in by a human expert articulator.

Hx. 3 If ontologies are composed using an algebraic framework, the number of articulations that must be performed are less than that when all ontologies are articulated on a pair-wise basis, thereby enabling scalability.

Hx. 4 An algebra can be designed to describe the composition of ontologies.

Hx. 5 The properties of the operators in an ontology-composition algebra determine whether a task of composing ontologies can be rewritten as equivalent tasks. The most optimal equivalent task with respect to a cost function can then be chosen for execution.

Hx. 6 The properties of articulation generation functions determine the properties of the operators in an ontology-composition algebra.

I argue that each hypothesis holds in the following chapters. Together, by proving the six hypothesis, I form a foundation based on which applications can enable interoperation among information sources by effeciently composing ontologies after resolving the semantic heterogeneity among them.

To prove the hypotheses, in this dissertation, I describe ONION, a toolkit, which enables interoperation among autonomously maintained information sources. I provide algorithms for its most crucial components and demonstrate their successes. Specifically, I show that SKAT can establish articulation rules among multiple ontologies semi-automatically. I also demonstrate that information from multiple sources can be composed using an ontology-composition algebra. Furthermore I have characterized the properties of the operators of the algebra, and shown that the properties are determined by those of the articulation rules, which form the basis of the composition of ontologies.

1.4.2 Organization of the Dissertation

The organization of the rest of the dissertation is as follows. In Chapter 2, I discuss the motivation behind building an interoperation system. In Chapter 3.2.2, I lay out the conceptual architecture of an ontology management system. In Chapter 4, I specify a common ontology format that input ontologies must be wrapped to before the different components of the ONION system can process them. In Chapter 5, I describe SKAT, ONION's articulation-generator, and the several heuristic algorithms that it uses. I also show the experimental results obtained using SKAT. In this chapter, I verify the first two hypotheses. Chapter 6 contains the definitions of the ontology composition algebra and its operators. In this chapter, I validate the third and fourth hypotheses. I also discusses the effect of the articulation generation function on the algebraic operators. I discuss the properties of the operators and show how these properties determine the level to which ONION can compose ontologies optimally — with respect to suitable cost functions — thereby validating the last two hypotheses. In Chapter 7, I discuss related work and compare and contrast the ONION approach with these works. I conclude the dissertation in Chapter 8, and lay the ground for some future work by me and others by raising questions that arise out of the present work and are yet unanswered.

Chapter 2

Motivation

In the following chapter, I describe the ONION system. However, before I outline the system architecture in the next chapter, in this chapter, I disucss the motivation behind the design of the system and substantiate the reasons for the choices I made in this work. To substantiate my choice, where necessary, I briefly compare and contrast my approach with alternative approaches.

2.1 Global Consistency

To avoid the problem of information sources having different semantics, some researchers have tried to build a standard ontology or global schema and then build information sources that conform to that ontology or schema [GKD97], [KLSS95].

Even though this approach has worked for small communities, coming up with an acceptable standard for vocabulary in larger domains is impossible, especially among groups that have different applications in mind. There continue to be extensive debates about establishing a single top-level ontological taxonomy underpinning all terms in existence. The expectation is that a comprehensive ontology is reusable and will also impose consistency onto the world.

2.1.1 Updates to Ontologies

Most of the focus of ontologists has been on developing ever larger, static ontologies [Gru93]. Without an explicit contextual constraint they will differ in scope and structure, even though the developmental efforts were typically initiated in a specific application domain. When

new, related applications arise, existing ontologies must be broadened. For instance, to serve a set of intelligence queries [HPK] hundreds of new definitions had to be added to the already very broad Cyc ontology [Tek97]. When that ontology had to serve chemistry students and take a challenge examination for an Advanced Placement Test in chemistry, it required many more updates and revisions [all04].

2.1.2 Assumptions of Global Consistency

Global consistency is often assumed by applications that want to exploit the wealth of information that is available today. However, as soon as domain boundries are crossed, there will be a need to compose information from multiple Web sources. The ontologies used to answer questions posed in a high-school advanced-placement test in chemistry is of little use to military planners dealing with chemical weapons. When queries are answered using composed information whose semantics do not fully match, the quality of the answer suffers since major mismatches can occur.

2.1.3 Problems with a Globally Consistent Ontology

While global consistency might be wonderful for the users of information systems, it also has significant costs for the providers of information, and in many cases the costs will be greater than the benefits for specific information providers. A globally consistent ontology has the following problems:

1. Unnecessarily large size: The ontology contains a large number of concepts that are useless for the purposes of several end-users and information providers.

2. Imprecise definitions: The definition of the concepts in a non-specific context are a result of compromises. These definitions do not precisely reflect the semantics of the concepts used in the information sources or in the applications.

Problem 1 results in degradation of performance.

Problem 2 results in inaccuracies. Publishers use ontologies associated with information sources because they promise to provide the exact semantics of the concepts used in the information sources. If committees are needed to achieve global consistency by defining and standardizing terms, the committees are likely to make compromises since members often have conflicting and irreconcilable interests. As a result, the precision of the ontology

suffers and the promise of providing the exact semantics of concepts is broken. Inaccuracies abound due to the use of imprecise definitions of the terms used in the information sources.

2.1.4 Errors due to Differences in Scope

Consider the case where the creator of an information source wants to publish some information about a particular concept. Unfortunately, the global ontology contains no term whose defined semantics is exactly the same as the semantics of the desired concept. Instead, the global ontology has a similar concept. In such a scenario, the creator of an information source often makes a compromise. She chooses the term representing the concept that is nearest to the one she is seeking. Thus, in essence, she has used the same concept name with a slightly different semantics. However, this difference of semantics is not captured anywhere and results in misinterpretations by subscribers of the information source. Misinterpretations often result in missed business opportunities or even catastrophic losses.

2.1.5 Trying to Achieve Global Consistency

I describe below the process of achieving global consistency, outline the costs of being consistent especially when the objectives of the agents that use the information differ, and then discuss the most serious issue: the cost of maintaining consistency over time.

Errors due to Inconsistencies:

There have been a number of efforts to achieve consistency across multiple sources, typically by merging their terminologies with as much care as is possible. For large terminologies the efforts are large [MRB+96]. Problems arise in combining ontologies when the objectives of the sources differ [RL02]. If the exact semantics of terms are lacking, a term can be unified with another term that is at a higher level than the ideal match. Information retrieved on the basis of such unification can be misleading and excessive information will be retrieved.

Manual Disambiguation:

When the application's objective is to study and understand, the end-user can reject misleading information. Often, the volume of information retrieved is larger than necessary, and the searcher is assured of missing little. The end-user is overloaded and has to remove the chaff to retrieve the required information. Ranking of retrieved material can help in

dealing with excessive volume. However ranking based on popularity, as provided by Google [BP98] is not always the best for a service-provider or a business and does not eliminate the need to filter information manually.

Merging Ontologies:

Merging of large ontologies is a major effort [RL02]. Automating the integration task is desirable. Most merging tools are based on linguistic similarity. To get more matches thesauri might be used [MW02], [GMJ]. Terms that are spelled similarly may differ in meaning, sometimes because they have a common etymology, as 'vehicle' for transport, and 'vehicle' in chemical engineering. However, even though terms in a specific domain tend to be unambiguous, the confusion increases when the coverage of the ontology increases.

2.1.6 Cost due to Lack of Precision

The cost of missing relevant information (*type 1 errors*) can greater than the benefits of avoiding excess information (*type 2 errors*). This scenario occurs when the information being searched is relatively rare and finding it is crucial.

When the information that the end-user is searching for is abundantly available, the opposite scenario exists. The benefits of getting comprehensive information is less than the cost of weeding out the type 2 errors. For routine purchases, missing a few suppliers, say 2 out of 10, can mean missing one with a slightly better price than that of the chosen supplier. However, the purchaser saves the cost of locating those two suppliers, since to locate them he might have to look through the information corresponding to 50 potential suppliers. In the limit, to be absolutely assured of finding all suppliers, thousands of unlikely potential suppliers will have to be inspected. Usually, if we try to reduce type 1 errors, the number of type 2 errors increases rapidly, and vice-versa. The right balance between the cost and benefits of additional processing needs to be struck. Using a broad-based engine, say Google, to achieve that goal is not likely to be successful in reducing the cost [BP98].

2.1.7 The Cost of Precision

To avoid such 'information overload' business groups that interact are developing precise, controlled ontologies. Since the groups design the ontologies carefully and focus on a specific domain, as petroleum trading or short-haul trucking, they tend to be modest in size, and of

high quality for their limited application. Information in sources with well-defined ontologies can be profitably combined and used for complex applications, say in an emergency-management application, where petroleum products have to be shipped to supply a disaster site. However, before the ontologies can be combined, they need to be matched.

2.1.8 The cost of being consistent

Terms used in a domain are intended to allow people to communicate effectively. People use short, specific terms and abbreviations when they communicate with their peers. Often, members of the audience find themselves confused while listening in on a meeting of specialists. It is a major achievement for a scientist to become fluent in multiple domains. In addition to having to learn new words, and acquire new meanings for old terms, there will also be differences in abstraction and structure. A homeowner will use the term 'nail' for pointed things to be hit with a hammer, while a carpenter talks about brads, sinkers, in addition to modifiers as 8-penny nails, etc. For the homeowner to acquire the carpenter's terminology will be costly. To require the carpenter to just use the term nail and then specify length, diameter, head size and material engenders much inefficiency.

Structural differences create further mismatches. The traditional task of builders of ontologies has been to define a single ordered structure for the domain terminology, in the form of a hierarchical tree or lattice that is effective for the sponsoring audience. Medical experts think of the human body as composed of parts that are identified according to their function, while a tailor is content with a decomposition according to external shape. A cannibal may have yet a different model. Even a surgeon has a different mental model of the parts of the human body than that of a general practicioner.

When people use information that was not designed specifically for their purpose, we often encounter arrogance: 'I can't see why they classified x in category y', forestalling attempts at cooperation and integration. When data is used for unintended purposes, the error rate is rather high, say when billing data is used for medical research, since differences at low levels in clinical hierarchy would not affect billing.

2.1.9 The cost of maintaining consistency

Global agreement is hard to achieve, and even if it is achieved at an instant, our knowledge, and hence the meaning attached to the terms, changes over time. Everyday new discoveries

expand our knowledge and change our views of the universe. The intellectual process of splitting and lumping concepts, essential to the progress of science requires updating of source ontologies. Derived ontologies have to be updated then as well, a substantial effort when all possible related linkages have to be re-verified. Cyc addresses that problem through its use of micro-theories, but has not solved the issue in a formal manner [Guh91].

The most serious issue in dealing with large, integrated ontologies is their maintenance. For extremely large ontologies, the maintenance required to keep the ontology up-to-date will require all available resources. Development and maintenance costs increase more rapidly than the size of the sources, due to the exponential increase in possible interactions between designers of the standard ontologies.

I have no quantitative information on the cost of ontology maintenance (preliminary evidence of its enormity is available in [Jan00]). The reintegration of changes in ICD-9 codes into UMLS induced a delay of several years. A comprehensive model of maintaining medical ontologies has been proposed, but not yet assessed in a large scale [OliverSSM:98]. Given that ontology maintenance is likely to be similar to software maintenance, I presume that over the life-time of an ontology maintenance costs will range from 60% to 90% [Bro95], with increasing values for large collections [Jon98].

Updating Global Ontologies:

Even sources that have the same semantics at the beginning can diverge after maintenance updates. For the sources to have the same semantics even after the updates, the following must occur:

- If a new term is added, all parties must agree on the exact semantics of the new term.
- If an existing term is updated, all parties must agree to the change in semantics of the updated term. Furthermore, the parties involved must change their information sources such that each use of the term, including pre-existing uses, in the information sources is now consistent with the updated semantics.

The maintainers of the information sources that use a standard ontology must agree on the updates being proposed to that ontology. In fast changing fields like medicine, updates to standard ontologies are proposed based on newly discovered theories. Some of these theories result in controversies. Owners of information sources who target entirely

different applications than the proposers of the ontology update, or have differences in philosophy with the newly discovered theories do not want to accept the proposed updates. Furthermore, some participants might see the changes required to support the proposed updates as an unnecessary imposition since restructuring the information source will require substantial effort on their part without substantial benefits.

In quickly changing fields, arriving at a consensus on proposed updates to the standard ontology within a short period of time is not even feasible. Besides, the cost of such negotiations are often unacceptable to the ontology-owners. Therefore, sources are maintained autonomously resulting in heterogeneity among them.

Soon, we see multiple standard ontologies emerging and being used by information sources in the same domain, proving the goal of one standard ontology a myth. From the world of one integrated ontology, we return to a world having several ontologies requiring interoperation.

Besides, even if the owners, publishers, subscribers and users of information all decided upon a standard ontology and put in the resources to maintain it, it will be prohibitively expensive to restructure existing legacy information systems so that they conform to the standard ontology. The need for resolution of semantic heterogeneity and interoperation will therefore remain.

2.2 Interoperation versus Integration

ONION allows the sources to be updated and maintained independent of each other and enables the composition of information via interoperation. In order to answer queries posed by end-users, information is retrieved from individual sources and then composed as needed. Obviously, a scalable system needs to make intelligent choices as to which information is essential and needs to be transferred from the sources. Besides reasons of autonomy, interoperation provides advantages associated with distributed systems like having no single point of failure resulting in high availability.

An information interoperation system differs from an information integration system. While the former maintains the autonomy of the individual information sources, the latter integrates all the information available from multiple sources into one new integrated source. A typical example of an information integration approach is a data warehouse. Large corporations, like Walmart, maintain huge data warehouses where they have integrated and

stored information that is available to the organization from all its various sources, say for example, its suppliers or its various stores from all over the country. Data warehouses can work in a one-company setting, under tight controls from a parent company. However, the technique is infeasible if the intent is to enable access to data and information from a large number of information-producing organizations that must remain autonomous.

For an interoperation system to work, only the articulation of the individual sources must be established accurately. The interoperation system requires only the subset — the intersection of the sources relevant to the application at hand — of the information to be correctly matched, while an integration system requires matching and integrating entire information sources.

2.2.1 The Need for Interoperation

Most information sources are built to satisfy specific requirements of the individuals or the businesses that created them. The booksellers "amazon.com" and "Barnes and Noble" have created websites that provide access to their inventory of books. The purpose of the website is to sell books directly over the Internet to customers. The end-users would like an integrated datawarehouse that lists books, their sellers, their prices, and reviews of each seller side by side. Big booksellers like "amazon.com" and "bn.com" may not want their prices provided side by side with those of a deep-discount bookseller. Furthermore, generally booksellers want to retain control over their information. Among other things, retaining control over their information source allows them to make updates as and when required without having to inform the integrated data-warehouse and hence their competition of their changes. The booksellers want to keep their information sources autonomous and integrating the information in them to create a single source is not a viable option. Besides, if the integrated data-warehouse seeks to have a comprehensive listing of books and their prices as provided by all booksellers, the warehouse becomes unwieldy due to the huge amount of data and the frequent updates it has to entertain. Querying in such huge data warehouses often becomes slow unless a large amount of money and effort is spent to structure and index the data extremely efficiently.

2.2.2 The Synchronization Problem for Updates to Information Sources

In the rest of the work, I will refer to information sources whose information is obtained directly from the source of the information as *primary information sources* and information

sources that are constructed using information obtained from primary information sources as *secondary information sources* e.g., data warehouses or sources with aggregated data.

Researchers have created secondary information sources by integrating data and information from various primary sources [CIA00], [RKS+94]. However, secondary information sources often result in the presentation of stale data. For example, when information in the OPEC website [ope] was updated, the CIA Factbook, which contained information about OPEC, was not updated to reflect the changes in the OPEC. Consequently, the information available from the CIA Factbook was stale until the next year's edition of the CIA factbook was released. To avoid staleness of data, as soon as any information in any primary source changes, the secondary information sources must also be updated [OSSM98]. Oliver [OSSM98] cites cases where such updates took several years. If the primary sources change frequently, updating the secondary source to keep it consistent with the primary source becomes a costly affair. Oliver also found that when the structure of the primary information sources change, manual work is necessary to update the secondary sources of the information [OSSM98].

2.2.3 Cost of Merging Information Sources

When the expert resolving the inconsistency does not know the purpose and scope of use of the information and the number of sources is large, it becomes nearly impossible for the expert to achieve useful resolution of the inconsistency. When the inconsistencies between information sources cannot be resolved, a complete integration of such sources is infeasible. I have already discussed the issues involved in obtaining a global resolution of inconsistencies between information sources in sub-section 2.1 above.

Composition of ontologies for an application by merging has been performed for many instances [JSV+98]. It serves immediate needs effectively, and avoids the long-term maintenance problem. If an organization has to maintain one integrated ontology, it has to spend more time and effort than when it has to maintain multiple smaller ontologies. If an organization can use ontologies maintained by others, the savings are even greater.

2.2.4 The Maintenance Problem

Even if the sources can be merged to an integrated source, maintaining the large integrated source is prohibitively costly due to the sheer amount of information present and due to

the cost of synchronizing the source with the individual primary sources. Since it is costly to achieve and maintain global semantic information — information about the standardized semantics of concepts used across information sources — the merging approach is not scalable.

Therefore, we have desiged ONION to be an interoperation system based on an ontology-composition algebra. The declaratively-specified algebra helps maintain the formal relationships between the source ontologies and the derived ontologies such that they can be automatically replayed when the source ontologies change and the derived ontologies have to be synchronized with the sources.

Chapter 3

Architecture

In this chapter, I describe the conceptual architecture of the ontology management system that I built. In the next chapter, I describe their design and implementation details. The ontology management system can be conceptually divided into the following components:

1. Ontology Pre-processor
2. Articulation Rule Generator
3. Ontology Composer
4. Ontology Repository
5. Ontology Viewer
6. Query Processor

Figure 3.1 shows the different components of the interoperation system. I describe the functions of the different components in the rest of the chapter.

3.1 Ontology Pre-processor

The Ontology Pre-procesor is the module that pre-processes the source ontologies. I provide a brief description of its specification below and then elaborate on its function. The description lists the input and output of the module. The function of the pre-processor is dependent upon the needs of the individual ontology management system that it is a part

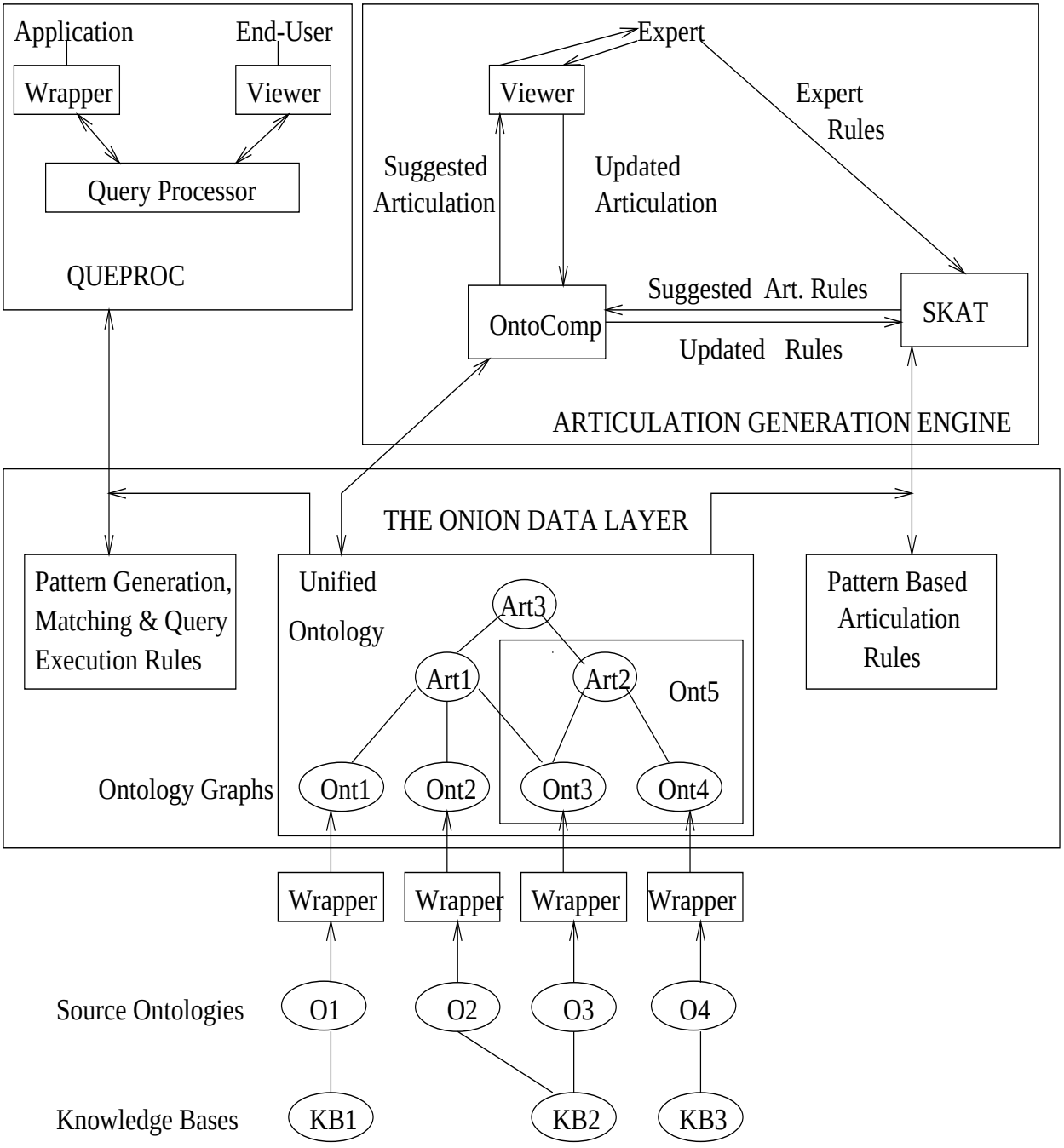


Figure 3.1: The ONION Interoperation System

of. Generally, I foresee this module being responsible for converting the source ontology into a format and rule language the ontology management system can accept. The component can be totally automated or semi-automatic where the pre-processing is facilitated by using interactive assistance from a human expert.

An example of a pre-processor is the set of interdataworking wrappers made available by Sergey Melnik [Mel].

- Ontology Preprocessor: Module to pre-process ontologies.

  Input: Source Ontology.

  Output: Pre-processed Source Ontology.

  Function: To wrap source ontologies into formats and rule languages supported by the Ontology Management system.

  Facilitator: Optional expert input depending upon implementation.

  Example: Interdataworking wrappers [Mel].

In Figure 3.1, the source ontologies *O1*, *O2*, *O3*, and *O4* are associated with the knowledge bases *KB1*, *KB2*, and *KB3*. The *wrappers* shown in the figure pre-process the source ontologies to make them compatible with the input format for the ONION System. The pre-processed ontologies are referred to as *Ont1*, *Ont2*, *Ont3*, and *Ont4*.

3.1.1 The Articulation-Rule Generator

An articulation rule generator takes two pre-processed source ontologies as input and outputs a set of articulation rules. As indicated before, the articulation rules express the corresponences between the concepts in the two pre-processed source ontologies.

The specification of the articulation rule generator is as follows:

- Articulation Rule Generator: Module to match ontologies and generate semantic articulation rules.

  Input: Two pre-processed source ontologies

  Output: A set of articulation rules

  Facilitator: Human expert

Function: To generate rules that establishing the semantic correspondences among concepts in the source ontologies iteratively using input from the human expert.

Example: SKAT

A domain expert deploys the articulation-rule generator by providing it with the names or references of the two ontologies that must be matched. The articulation-rule generator matches the two ontologies by identifying articulation rules (semantic bridges) between concepts in the the ontologies. Associated with each rule is a confidence score indicating how close the matches are according to the module.

As indicated before, the domain experts verify and accept the correct rules. The expert has the final word on the articulation-rule generation and is responsible to correct inconsistencies in the suggested articulation rules.This module may perform several iterations of consulation with the domain expert and refining of articulation rules before the final set of articulation rules are accepted. This iterative process terminates when the expert is satisfied by the set of rules generated by the articulation-rule generator.

The expert has the option to indicate whether a rule is valid, invalid, or irrelevant for the purposes of the application. A valid rule is stored by the articulation-rule generator in a repository. An invalid or irrelevant rule is deleted from the set of articulation rules but logged along with the response of the domain expert for future use. The stored rules in the articulation repository and the over-ridden rules in the logs, can be used in future. These rules are used when (according to the expert) the ontologies that are being articulated are from the same domain as the ontologies whose articulation rules are available in the articulation repository and the logs. For reuse in later articulation-rule generation, the expert will assess if the two applications are similar. The articulation rules in the articulation repository positively reinforce the articulation rules generated between the new pair of ontologies. If the rule was over-ridden or rejected during a previous run, it negatively influences the confidence score for the rule generated. The amount by which the score is reduced can be provided by the expert before deploying the articulation-rule generator.

An example of the articulation-rule generator used in the ONION system is SKAT.

3.2 Ontology Composer

The ontology composer is a module that takes in two or more pre-processed ontologies, and an algebraic expression specifying the composition task. The algebraic expression is used to select and articulate portions of existing ontologies to create new ontologies. If the ontologies have not been matched previously or if the articulation rules among the ontologies are not available to the module, it calls the ontology-rule articulation engine to generate the articulation rules among the ontolgies. Then, based on these articulation rules the ontology composer constructs the new ontology specified by the algebraic expression.

A brief description of the Ontology Composer module is as follows:

- Ontology Composer: Module to compose existing source ontolgies and create new ontologies.

  Input: Two or more source ontologies, an algebraic expression describing the desired composition.

  Output: Composed new ontology.

  Facilitator: None, fully automated operation based on inputs.

  Function: Select portions of existing ontologies, and create a new ontology as specified by the algebraic expression. The composition of the selected portions of existing ontologies is based on the set of articulation rules generated by calling Module 2.

  Example: OntoComp

An example of an ontology composition module is *OntoComp* used in ONION.

3.2.1 The Ontology Repository

The ontology repository is a central store used to store ontologies and articulation rules. The items stored in the Ontology Repository are as follows:

- Large modified source ontologies

  Stored: by the Ontology Pre-processing module.

  Duration: are stored as long as they are needed by the other modules that operate on them, like the Articulation-Rule Generator or the Ontology Composer. After the composition task is over, they are deleted.

- Articulation Rules

  Stored: by the Articulation-Rule Generator.

  Duration: are stored until

  - they are invalidated by changes in the source ontologies or
  - they are no longer necessary because all applications using the articulation rules or ontologies derived from the articulation rules have completed and will no longer be run in the future.

- Log of expert input

  Stored: by the Articulation-Rule Generator.

  Duration: is stored as long as all the ontologies and applications that the input is related to are valid.

- Newly composed ontologies

  Stored: by the Ontology Composition module.

  Duration: are stored as long as they are needed or may be needed by any application or information source.

A brief description of the ontology repository is as follows:

- Ontology Repository: Module to store modified source ontologies, articulation rules, and newly created and composed ontologies

  Input: Ontology/Rule-Set to be Stored or Loaded.

  Output: Result of Store or Load operations.

  Facilitator: None, totally automated component.

  Function: To store ontologies and rule-sets that are too large to be retained in main memory or that need to be stored persistantly due to the requirements of the application.

  Example: RDFStore [Mel02], Jena [WSKR03]

The most rudimentary ontology repository may be a simple file system, however, for most practical systems a repository with a database system at the back-end will be used, as in RDFStore [Mel02], Jena [WSKR03], etc.

Ontologies can be can be stored and retrieved from the ontology repository using their names or references like URIs. As mentioned before, the responses of the expert during the articulation-rule generation phase are stored can be used as hints in the future when the same ontologies need to be re-articulated due to some updates or when ontologies similar to those source ontologies are being articulated. The repository may also be used for storing intermediate ontologies created during the composition step, and query execution. For example, in the figure, we see that a derived ontology *Ont5* is stored in the ontology repository. *Ont5* contains the ontologies *Ont3*, and *Ont4*, as well as the articulation ontology *Art2* between them.

The Ontology Viewer

The Ontology Viewer module provides a graphical user interface so that a human expert can visualize source ontologies and the articulation rules among concepts in the source ontologies, select portions of the ontologies, and interact with the Articulation-Rule generator module. The module must also allow the expert to add, delete or modify the suggested articulation rules.

Typically, a domain expert initiates a session by calling into view the ontologies of interest by supplying the URLs (maybe local files) of the source ontologies). Ideally, the viewer should allow the expert to refine an existing ontology, select portions of ontologies that are of interest for the current application, import additional ontologies into the system, drop an ontology from further consideration and, most importantly, modify existing or suggested articulation rules, and specify new articulation rules.

A brief description of the Ontology Viewer is as follows:

- Ontology Viewer: Module to enable the expert interact with the ontology management system.

  Input: URIs of two modified source ontologies, or names of ontologies and their articulation rules as stored in the Ontology Repository.

  Output: Display of the two input ontologies and the articulation rules between them.

  Facilitator: Human Expert

  Function: Displays the source ontologies and the articulation rules generated between the ontologies. Human expert interacts with the system using this GUI. Input

of human expert logged for future use.

3.2.2 The Query Processor

The query processing module has not been implemented as part of this work. However, the algebraic framework that forms the basis of the implementation is complete and is discussed in Chapter 6. The algorithms for query rewriting and optimiztion are available in [Mit01, ALM02, ALM04]. The query processing module is responsible to processing queries posed to the system. There are two types of queries that can be posed to the system:

- Ontology-level Queries: These queries can be used to select and retrieve concepts in the ontologies and the relationships among them, and to create new ontologies or "ontology views" — a composition of concepts selected from various ontologies put together using articulation rules to form a new ontology. Not surprisingly, such a query can be translated to an equivalent expression using the ontology composition algebra.

- Instance-level Queries: These queries can be used to retrieve instances of concepts present in the ontologies and information sources. The correspondences between instances of concepts are also expressed using the articulation rules.

For a detailed discussion and examples, about the two types of queries that should be supported by an interoperation system, please see Section 5.2.

After the expert has articulated the ontolgies, the end-user can use the query system to query the information available across multiple source ontologies. The query processing module accepts queries from end-users, and experts, wraps or rewrites the queries using the vocabulary of the ontologies if needed, divides the query into subtasks, executes them using the available ontologies and information sources, and presents the results in a format comprehensible by the end-user. An end-user can also use the viewer can to interact with the system to run queries against a set of ontologies and their articulations. A user can select a set of pre-articulated ontologies and pose queries against them.

The query processor uses the articulation rules to reformulate the query, and executes it generate its solution. The query engine — the heart of the automated query-answering system — rewrites each query as a composition task. The composition is based on the established articulation rules. Optimization of the queries is enabled using the properties

of the ontology composition algebra, which I study in detail in Chapter 6. Queries are rewritten based on the articulation rules, and decomposed into subqueries. Each subquery is then redirected to a information source, evaluated there and the results are then composed to answer the original query.

A brief specification of the query processing module is as follows:

- Query Processor: module used to query the information in the ontology management system and the information sources.

  Input: A query

  Output: One of i) a newly composed ontology, ii) selected portions of existing ontologies, and iii) instances of concepts in the information sources, satisfying the query constraints.

  Facilitator: None, fully-automatic.

  Function: Retrieve the ontologies, concepts, relationships and instances of concepts that satisfy the requirements laid out in the query.

To conclude, in this chapter, I presented the architecture of the ontology composition system and briefly described its components. In the next chapter, I discuss the design and implementation of the components that were implementated as part of the ONION interoperation system.

Chapter 4

The ONION Ontology Format

In this chapter, I describe and discuss the ONION ontology format.

In this chapter, I specify the ontology format that ONION requires the input ontologies to be in — the ONION ontology format. Before I describe the ontology format, however, I provide a few definitions that I will use in the rest of this chapter and the dissertation. Finally, I describe the components of ONION and discuss how the system works.

4.1 The ONION Ontology Format

Heterogeneity among information sources needs to be resolved to enable meaningful information exchange or interoperation among them. The two major sources of heterogeneity among the sources are as follows: First, different sources use different data formats and associated modeling languages to represent their data and meta-data. In the rest of the dissertation, I will refer to the data format and the modeling language together as the ontology format. Second, sources using the same data format differ in their semantics. The ONION system uses a common ontology format, which I describe below. It expects all external ontologies, presumably in different formats, to be translated to this common format before it can resolve the semantic heterogeneity among the objects in the ontologies that it is articulating.

Melnik et al., [Mel00] have shown how to convert documents from one modeling format to another, e.g., from RDFS [rdf00] to DAML+OIL [dam01]. Wrappers convert the ontologies represented using their formats that I want to support within the ONION format.

4.2 Declarative versus Programmatic Specification of Format Converters

An expert who seeks to set-up an information integration or information interoperation system has to build individual wrappers to extract information from individual information sources and convert ontologies from one format to another. Another alternative to this approach is where the expert declaratively specifies a set of rules. These rules tell the system how to convert from a format native to the information source to the ONION format and then build an engine that takes in such specifications and performs the conversion. That is, I could first write rules that transform parts of one ontology from one format to another. Tools like OntoMorph [Cha00] make it easier to write translators to translate between data and knowledge formats declaratively.

However, often the transformations of objects from one format to another are quite complex and can be more naturally expressed procedurally. Expressing them declaratively requires a very expressive rule language. Deductions in more expressive rule languages are often not tractable. Also in ONION, I would have to create and manipulate articulation rules that not only have semantic information but also have information about transforming ontology formats. Besides, by converting to the ONION format, I eliminate the necessity of n^2 pairwise conversions among n ontologies and instead reduce it to n conversions (of all the ontologies to the common format). Similarly, formatted ontologies can share the same converter, so that the number of converters needed is considerably smaller than n.

4.3 Declarative Specification of Articulation Rules

While ONION can only process ontologies in the ONION ontology format, it refrains from requiring all ontologies to be converted to use a common vocabulary. It addresses the problem of establishing correspondences among ontology formats and the problem of establishing articulation rules among the concepts in the ontologies differently. This difference is because the number of modeling formats used to represent ontologies is distinctly smaller than the number of vocabularies existing in various fields where ontologies have been deployed. While I can write individual wrappers to convert the small number of ontology formats that I intend to support (currently XML, RDF, DAML+OIL, OWL) to one common format, the total number of different concepts and features used across all ontology formats are

rather large and creating a huge, integrated, common, global ontology format with all these different concepts and features is untenable. Furthermore, even if such an ontology could be created, it would not be maintainable.

Instead of coupling the two tasks, namely, resolving heterogeneity due to formatting, and resolving those arising from semantic heterogeneity, in ONION I take a layered approach. In the first, the heterogeneity due to different ontology formats are handled and in the second layer, I resolve semantic heterogeneity. Due to the widespread acceptance of standards like XML and RDF, the degree of heterogeneity with respect to ontology and data formats has reduced considerably. The problem of resolving heterogeneity of ontology formats is out of the scope of this work. For the handful of ontology formats in wide use today, I leverage the work done by Melnik [Mel] and others on the problem of converting between ontology formats. Instead, I focus on building a tool to resolve semantic heterogeneity and compose ontologies.

4.4 A Common Ontology Format

Information sources were, are, and will be modeled using different formats. I do not foresee the creation of a *de facto* standard data format that all information sources will use. The reasons for this stand are as follows:

1. Despite efforts, standardizing data and ontology formats have not succeeded in the past.

2. Even when standards arise, they are updated because either the initial design was not adequate or because the needs of the community have evolved. Updates result in versions of standard ontology formats. Once there are multiple versions of ontology formats, there is need to interoperate among them.

3. Different parties have different organizational needs. Standardization results in compromises. A standard ontology format will not meet the needs of certain organizations who need to move beyond the standard and proceed with its own ontology format.

Building an articulation-rule generator that resolves semantic heterogeneity among information sources and accepts inputs in all types of ontology formats is extremely difficult, complex and time-consuming. The ONION toolkit can only articulate ontologies represented using the ONION ontology format.

The ONION ontology format is a very simple format for representing ontologies. Unlike the recent formats DAML+OIL [dam01] or OWL [owl04], which has a number of classes defined, e.g., *Class*, *Datatype*, *Thing*, etc., the ONION ontology format only has nodes representing "concept"s. RDF [rdf99] is used to describe resources and their properties, and make statements about resources. Correspondingly, in the ONION ontology format, I can represent concepts and their attributes. The format has similarities with RDF-Schema [rdf00] in that it has relationships that have similar semantics to properties in RDF-Schema like "*SubClass*". I describe the format in detail and point out the similarities and differences with RDF and RDF-Schema below.

ONION matches ontologies that have been reformatted using the ONION ontology format to generate articulation rules. In order to allow for minimal-effort translations from various ontology formats, like DAML+OIL or OWL, I could have chosen the the common ontology format to be arbitrarily complex. I would then have inherited all the constructs in XML, XML-Schema [xml], RDF, RDF-Schema, DAML+OIL, OWL, UML [uml00], etc. into the ONION ontology format with some modifications done to handle the case where the same constructs have different semantic meanings in the different formats. However, processing such complex ontologies would require more complex software. To make the articulation generation simple, I take a different approach and strive to keep our format simple. In the next two subsections, first, I give an example of two ontologies expressed in the ONION ontology format, and then specify the ONION ontology format.

4.5 Motivating Example

To illustrate the ONION ontology format for ontologies, I use selected portions of three ontologies that I manually constructed. The portions of the ontologies *Transportation*, *Carrier*, and *Factory* have been selected (and greatly simplified) (Fig. 4.1). I have used SKAT to generate the articulation rules between the *Transportation* ontology with the *Carrier* and the *Factory* ontologies. These ontologies model the semantic relationships "*SubClass* ", "*Attribute* ", "*Equivalent*", and "*Instance* " that are represented as edge labels 'S','A', 'E', and 'I' respectively. For the sake of clarity, a few of the most obvious edges have been omitted. Apart from the above mentioned relationships, the individual ontologies also contain other binary relationships between terms that SKAT does not interpret. The ontologies are expected to have rules that define the properties of each realtionship, e.g., the ontologies

will have rules that indicate the transitive nature of the *SubClass* relationship. SKAT generates these links (without the relationship labels; those are supplied by the expert manually) between that the corresponding terms in the two ontologies. Typically the relationships are either *SubClass* or *Equivalent*, or *Instance*. However, they can be any arbitrary relationship, e.g., the relationship shown as *PSToEuroFn* between the price in *PoundSterling* and *Euro* supplied by the expert. The expert verifies the matches generated by SKAT, refines the matches it generates and supplies the exact relationships. The articulation generator uses these rules and they are also used while answering end-user queries.

I discuss the specifications of the graphical model and the semantics of the relationships that ONION can interpret in the next section.

4.6 A Graph-Oriented Conceptual Model

The formalization of the ONION ontology format — loosely speaking, a subset of RDF [rdf99] — is modeled on the work done by Gyssens et al., [GPVG90]. At its core, I represent an ontology as a graph. Formally, an ontology $O = (G, R)$ is represented as a directed labeled graph G and a set of rules R. The graph $G = (V, E)$ comprises a finite set of nodes V and a finite set of edges E. R is expressed as Horn clauses.

An edge e belonging to the set of edges E is written as (n_1, α, n_2), where n_1 and n_2 are two nodes belonging to the set of nodes V and α is the label of the edge between them. The label of a node n is given by a function $\lambda(n)$ that maps the node to a non-null string. In the context of ontologies, the label is often a noun-phrase that represents a concept. The label α of an edge $e = (n_1, \alpha, n_2)$ is a string given by $\alpha = \delta(e)$ where δ is a function that returns the label of the edge e. The label of an edge is the name of a semantic relationship among the concepts (that are represented as nodes) in the edge and it can be null. The domain of the functions λ and δ is the universal set of all nodes and edges, respectively (from all graphs), and their range is the set of strings (from all lexicons). I assume that the function λ maps a node to a unique label, that is, no two nodes in the same ontology share the same label. Thus, I will use the label of a node as a unique identifier of the node. As a short-hand, instead of writing out an edge with a node and providing its label separately, I will abuse the notation for the sake of clarity, and while representing an edge substitute the label of a node for the node and write an edge $e = (\lambda(n_1), \alpha, \lambda(n_2)$. All ontologies used in this work also satisfy the condition that between two concepts there exists not more than

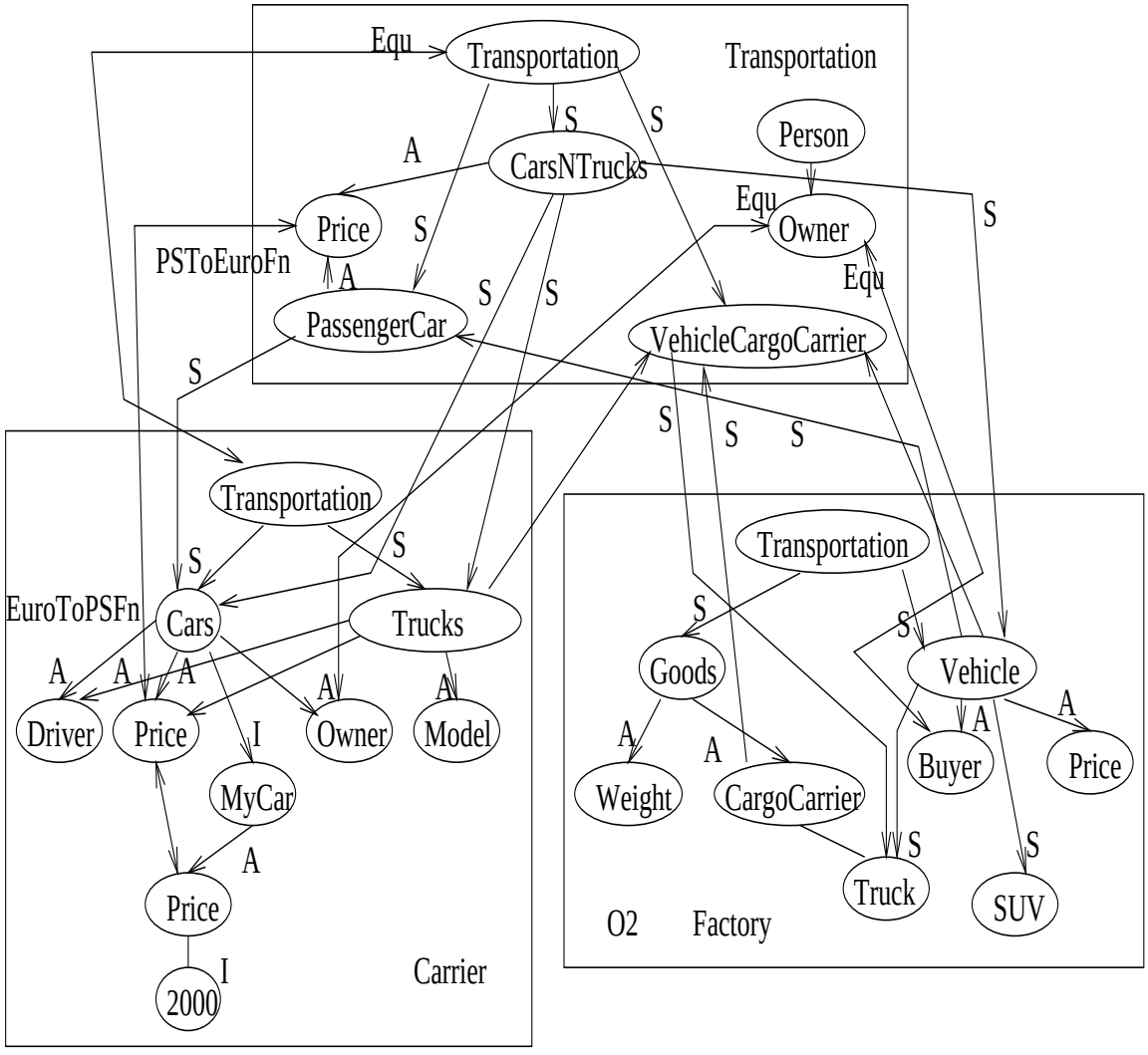


Figure 4.1: Two Ontologies and Their Articulations

one relationship. Further work needs to be done to eliminate this constraint and is beyond the scope of this dissertation.

The graph in the ONION ontology formalism can be expressed using RDF [rdf99]. Each edge in our graph is expressed using an RDF sentence. The two nodes in an edge are the subject and the object of the sentence and the relationship among the nodes is modeled as the property of the subject. In order to provide collection semantics RDF allows a construct called "containers", however, to keep our model simple, the ONION ontology format does not provide a construct to express containers in the graphical format. The onus is on the ontology designer to encode collection semantics or other desired semantics using rules associated with the ontology. Generally, the children of a node in the ontology graph are unordered. In order to express order among the children of a node, I use an "order" relationship explained below. As mentioned above, the ONION ontology format is expressed in RDF. By choosing RDF, I can leverage the various tools that are available for RDF and do not have to write parsers and other tools for our model.

Rules in an ontology are expressed in a logic-based language. The choice of the rule language, in principle, is left open to the ontology constructor. However, to keep the system simple, I have chosen the language of Horn Clauses as the rule language. Although, theoretically, it might make sense to use first-order logic or any other rule language with greater expressive power, in order to limit the computational complexity I use a simpler language like Horn Clauses. A typical rule $r \in R$ in an ontology is of the form :

$$CompoundStatement \implies Statement$$

A *Statement* is of the form (*concept Relationship concept*). A *concept* is either a *Node* or a variable that can be bound to one or more nodes in the ontology graph. Like an edge in the ontology graph, a *Relationship* also expresses a relation between the two *concept*s.

4.7 Semantic Relationships in ONION

SKAT can generate better articulation rules (Chapter 5 for a detailed description of the articulation rule generation techniques) if it has some semantic information about the relationships used in the ONION ontology model.

Certain data formats allow only strictly typed relationships with pre-defined semantics. For instance, relationships like *SubClass*, *Attribute*, etc., have very clearly defined semantics in most object-relational databases. A system that knows the exact semantics of the

relationships in a conceptual model can use the information, for instance, to find better matches between concepts in two ontologies or to perform type-checking and flag errors.

Other models allow any user-defined relationships, without any restriction. For instance, relationships like *OwnerOf* tend to be interpreted according to the semantics associated with them by the local application. Such relationships need not be strictly typed. A general system that imports a model allowing relationships that are not strictly typed does not know the application-specific semantic interpretation of these relationships. Such an approach provides enormous flexibility and can accommodate a large number of relationships. However, since the semantics of these relationships are not exactly known by the system, it cannot use them for matching related concepts or for type-checking.

Before I describe the ONION ontology format, I will define a few terms that we use below. A *Class* represents a concept. It is usually an abstraction for a group of objects with a common set of properties. An *Object* is an instantiation or member of a class. This usage is consistent with the usage of the term in object-oriented databases. The *Instance* relationship (described below) expresses the relationship between an *Object* and a *Class*. Note that I use the term "Object" to represent both an instantiation of a class and also to represent the last part of a sentence. However, the sense in which the term is used is clear from the context in which it is used. The ONION ontology format encourages the use of a set of strictly-typed relationships with precisely defined semantics. SKAT supports the following pre-defined relationships, whose semantics it uses while generating articulation rules:

- SubClass
- Property
- Instance
- Value

The semantics of these relationships are explained below.

In the following description, I use the term *Literal* to denote a string and the terms *Class* and *Object* to denote classes and objects in the sense it is used in object-oriented databases.

4.7.1 SubClass

This relationship relates two concepts each of type *Class*. The relationship indicates that one concept is a subclass of another. For example, the statement *(Vehicle SubClass Car)* denotes that the concept *Car* is a subclass of concept *Vehicle*. Any instance of the class *Car* is also an instance of the class *Vehicle* and all the attributes of the class *Vehicle* are also attributes of the class *Car*. The relationship *SubClass* is transitive, and in the absence of an explicit rule in an ontology that states that the *SubClass* relationship is transitive, ONION will add such a rule to the ontology before reasoning or rewriting the queries using the rules. The inverse of this relationship is *SubClassOf* or *SuperClass*. That is, *(A SubClass B)* is equivalent to *(B SubClassOf A)* and *(B SuperClass A)* under the interpretation of the ONION system.

4.7.2 Property

This relationship indicates that a concept is a property of another concept, e.g., an edge *(conceptA Property conceptB)* indicates that *conceptA* has a property named *conceptB*. This relationship, also referred to as *Attribute* in some information models, has typically the same semantics as attributes in (object-)relational databases. The relationship has two forms:

1. Property: Class ⇒ Class
2. Property: Object ⇒ Object

In the first form, the subject and the object of the sentence with the relationship *Property* are both of type *Class*. For example, in Figure 1.2, we have an edge labeled 'A' between *Car* and *Price*. That edge is written in the ONION ontology format as a sentence: *(Carrier.Car Property Carrier.Price)*. In the second form, subject and the object of the sentence are both of type *Object*. For example, in the sentences *(Car Instance MyCar), (Price Instance MyPrice), (MyCar Property MyPrice)* we see that the objects *MyCar* and *MyPrice* are related using the relation *Property*.

The following rule holds:

$$((B\ Property\ A)\ \wedge\ (C\ SubClass\ A))\implies (B\ Property\ C)$$

The inverse of the *Property* relation is the relation *isPropertyOf* . That is, *(A Property B)* is equivalent to *(B isPropertyOf A)* under the interpretation of the ONION system.

4.7.3 Instance

This relationship indicates that an object is an instance of a class. Therefore, the first concept in the relationship is of type *Object* and the second of type *Class*. For example, an edge (*Car Instance MyCar*) indicates that *MyCar* is an instance of the *Class Car*. The following rule holds:

$$((B\ Instance\ A)\ \wedge\ (C\ SubClassB))\implies(C\ Instance\ A))$$

The inverse of this relationship is the relationship *InverseOf*. That is, *(A Instance B)* is equivalent to *(B InstanceOf A)* under the interpretation of the ONION system.

4.7.4 Value

This relationship is used in the following scenarios:

1. *Value*: *Class* $\Rightarrow$ *Class*
2. *Value*" *Object* $\Rightarrow$ *Object* — *Literal*

In the first sense, it expresses the relationship between two classes. Typically, the first class is a class that is a "property" of another class and the second class is the value of this property. For example, we use *(Steel$_D$oor Property Material), (Material Value Steel)* to express that the value taken on by the *Class Material* is the *Class Steel*. In the second sense, the relationship represents the value of an attribute of an object, e.g., *(Car Instance MyCar), (MyCar Property CarAge), (Age Instance CarAge), (CarAge Value "5")*. Thus the object of the relationship *Value*, "5" is of type *Literal* and the subject *CarAge* is of type *Object*.

4.8 Sequences

XML is becoming a popular format for expressing data and meta-data on the web. Like SGML and other markup languages primarily designed to express documents, XML imposes order among its elements. By itself, the graphical ONION model, described above, does not impose order among the children of a node. To express order, I introduce a special relationship, namely *Sequence*, which is very similar to the container *Sequence* in RDF as well as in some SQL extensions and object-oriented models. For example, a list ranking cars can be described using the edges

$(MoneyLineRanking\ Sequence\ CarRankingList)$,
$(CarRankingList\ :1\ HondaAccord)$, and
$(CarRankingList\ :2\ FordTaurus)$.

The intermediate node *CarRankingList* represents the list object and its elements form an ordered sequence. In an edge of the form (*conceptA Sequence conceptB*) the first concept can be a *Class* or an *Object* and the second concept is an *Object* representing the list. The individual elements of the list can be objects or classes and are related to the list-object via the relationships $:1, :2, \ldots, :N$, where the list has N elements.

4.9 The ONION Ontology Format and SKAT

In the ONION format, I do not require that every relationship must belong to the small set of relationships whose semantics are predefined. The model is flexible enough to allow any other user-defined relationship. SKAT will not be able to use the relationships, whose semantics it is not aware of, unless the creator of the source ontology explicitly specified the semantics of the relationships using rules. For example, if the source ontology uses a relationship *Is-A* and has a rule expressed in the ONION rule language (i.e., as Horn Clauses) that says that *Is-A* is transitive, SKAT can use that information to generate matches. The articulation rules that SKAT generates uses only the relationships whose semantics are predefined to establish correspondences among nodes in the source ontologies.

4.9.1 Interpretation of Relationships

An expert invokes SKAT with two source ontologies. SKAT only matches concepts in the two source ontologies and does not attempt to match the relationships among ontologies. SKAT uses only those relationships whose semantics it clearly understands (by virtue of them belonging to the set of relationships defined in the ONION ontology format) to derive meaningful matches among the nodes in the ontologies that are being articulated and ignores the relationships whose semantics it does not know.

Let us suppose that two different ontologies have the relationships *Buyer* and *Owner* expressing the obvious relationship between a car and the person who buys it and owns it respectively. In most cases, the two terms refer to the same person, but different terms are used because of the differences in context. The Ford dealer that sells a car uses the term *Buyer* to refer to the person who buys the car. The Department of Motor vehicle registers

the car in the name of the same person but lists the person as *Owner*. Such relationships that are similar are not detected by SKAT since it does not try to match relationships.

While matching the nodes, SKAT checks to see if the same relationship holds between a pair of nodes and make deductions based on that, but unless two relationships in the source ontologies are indicated to be equivalent using explicit rules, it does not try to determine the similarity among relationships in the source ontologies. Therefore, it treats the relationships *Buyer* and *Owner* as distinct. Although it is perhaps not difficult to use the same techniques that I use to match nodes to match relationships, again I have taken a design decision to match only nodes and not relationships, so as to keep SKAT simple.

A workaround to the problem caused by SKAT not matching relationships is to *reify* the two ontologies, that is, to model the relationships as nodes themselves. To allow such modeling, sc Onion allows two more special relationships with pre-defined semantics: *rel:domain* and *rel:range*. For example, an edge, $e = (A\ Buyer\ B)$ can be modelled by two edges $e_1 = (A$ *rel:domain Buyer)* and $e_2 = ($*Buyer rel:range B)*. Once the relationships are modeled as nodes, as shown above, the articulation-rule generator can match them and establish semantic correspondences between two relationships from distinct source ontologies.

4.10 Reference and Subsumption

4.10.1 Object Definition in Semi-Structured Models

Several data formats now in common use for databases and ontolgies were designed predominantly to model documents, e.g., XML [xml04], SGML [sgm], OEM [ACM01]. In such data formats, where there are nested objects and entities, an object is modeled as a subtree in a graph. The entire subtree rooted at a node comprises the object that the node represents. When a query asks for the object, the entire subtree is returned. Such models assume that an object subsumes all objects that are in its subtree.

4.10.2 Objects in ONION

In our model, I intend to keep the concept of an object as simple as possible. Faced with the question of defining the scope of an object in the ONION ontology format, I take a minimalistic approach. In the ONION ontology format, a single node represents a concept and is either a class, or an object, or a value. All edges are referential in nature. Thus,

when a query asks to select an object, only the node representing the object is returned and not the entire sub-graph reachable from that concept-node. This minimal definition of an object helps me keep the articulation rules and the resulting ontology intersections as small as possible. The larger the intersection, the greater the cost of managing the articulation and the greater the cost of any operation that requires manipulating the articulation e.g., while using the articulation to answer queries. Therefore, I decided to keep the definition of an object as simple as possible.

4.11 Specifying the Semantics of Relationships Using Rules

In the ONION ontology format, apart from the graph model, an ontology can also contain a set of rules specified declaratively. I provide this feature to facilitate the translation of ontologies that have features that do not have an equivalent in the ONION ontology format. For example, an ontology might use a relationship that does not have an equivalent relationship in ONION. The semantics of such a relationship can be specified as a rule associated with the formatted ontology. Currently, these relationships and rules are not interpreted by ONION. Of course, despite having the rule language, I do not expect that all features taken from all possible ontology formats and modeling languages can be expressed using the Onion model. However, in order to maintain a simple ontology model, I forsake the ability to be more inclusive with respect to models.

4.12 Translation of Ontologies and Articulation-Rule Generation

If the articulation-rule generator understands only one ontology format, ontologies that that generator will match must be translated to the common ontology format. So if a feature in an ontology format cannot be translated into our common ontology format, it will not be matched with similar features carrying similar semantic messages in other ontologies by the articulation-rule generator. Therefore, it is important for the translation process to be as complete as possible. Of course, such information can still be accessed directly from the individual ontology and the engine associated with the individual sources, but the ontology articulations will be ignorant of such information even if that information is semantically related to information in other ontologies that are being used by the application. Ontologies

expressed in different ontology models and modeling languages are converted into the ONION ontology format by using wrappers.

In this section, I have discussed how I avoided the problem of heterogeneity among ontology formats by designing a common ontology format to which ontologies need to be converted before ONION can use them. I have also discussed the features of the ONION ontology format that the articulation-rule generator expects all input ontologies to be conformant with.

However, the second important problem of semantic heterogeneity among the concepts used in the source models still remains. In the next section, I will summarize various methods that I use to automatically suggest ontology articulations.

To conclude, in this chapter, I presented the specification of the ONION ontology format. In the next chapter, I discuss in detail how SKAT works and the heuristic algorithms it employs.

Chapter 5

Articulations and their Generation

As indicated in the previous chapters, an articulation between two ontologies matches the two ontologies and established the correspondence rules among concepts across the two ontologies. The linked concepts have semantic similarities that are useful for the application for which the articulation is constructed. In this chapter, I first motivate the need for an articulation, show the use of articulation rules to answer queries, and then discuss how SKAT generates articulations semi-automatically.

5.1 The Need for Articulation

The creators of information sources use different vocabularies to express the information contained in the sources, even when the information is from the same domain. To satisfy the needs of end-users, applications require information from multiple information sources. An application can utilize the information from two sources completely and accurately only after establishing the precise semantic correspondences among the objects in the two sources. Missing correspondences - both matches and mismatches - result in errors of omission (cited as type 1 error in Chapter 1). The conclusions drawn using wrongly matched information are erroneous (type 2 error due to semantic mismatches). Therefore, for information sources using diverse vocabularies, an application that interoperates between the sources needs precise and complete articulations of the ontologies that contain the specification of those vocabularies.

Articulations play a crucial role when two software applications, representing two different parties - individuals or institutions - have to cooperate to achieve a common goal. In

order to achieve their goal, the applications must communicate and exchange information. The information sources associated with the two different applications often use different vocabularies. In the last chapter, I stated why getting multiple applications and information sources to agree to one common vocabulary is futile. In the absence of a common vocabulary, the applications can communicate meaningfully only if information from the source corresponding to the applications that is sending a message is translated accurately using a vocabulary that the applications who is receiving the message can interpret. Again, ontology alignment and articulation is the answer [Hov98].

Even when the aim is not to interoperate but to integrate multiple sources of information, articulation of the various ontologies corresponding to the sources is essential. While constructing the Unified Medical Language System [MRB+96], the medical terminology used in the different machine- readable medical information sources had to be first matched and aligned before they could be merged. The work done by Hovy [Hov98] is another example where ontologies had to be articulated. In his effort, to create a single top-level ontology of world knowledge, Hovy had to match and reconcile terms used in the ontologies Sensus [KL94], Cyc [cyc], and MIKRO [Mah96] that purported to represent common sense knowledge. All three ontologies were rather large. Hovy articulated 6000 concepts from Sensus [KL94] with 5000 concepts of MIKRO [Mah96] and 3000 concepts of CYC[cyc]. To match large ontologies, one needs to consider a large number of potential node-pairs although most of them eventually do not match. Matching such large ontologies without automated tools involves an unacceptably huge manual cost. Besides, experts who manually match very large ontologies, often miss potential semantic links between objects across sources and sometimes misinterpret the semantics of objects in the sources to establish wrong matching rules because no expert can envisage all the possible consequences of a match. Such omissions and errors result in losses from imprecision and lost opportunities.

The constructor of a new ontology, who wants to reuse existing ontologies to create the new one, must first articulate selected portions of the existing ontologies before they can be composed to create the new ontology. Often, designers of new applications also need to create new ontologies. Even though the application is new, it might have similarities with existing applications. The concepts and relationships used in the new application is often not exactly the same as those in any one existing ontology but is similar to concepts and relationships used in several available existing ontologies. The designer of the application

can create the new ontology specifying the vocabulary used in the new application by selecting and composing concepts and relationships from existing ontologies. The sub-ontologies that are composed to create the new ontology contain concepts that are semantically related. If the correspondences between concepts across the sub-ontologies are not identified, the sub-ontologies remain disjoint in the new ontology. To identify the correspondences between concepts across the sub-ontologies, the designer of the application or a domain expert employed by the designer, needs to articulate the sub-ontologies before composing them.

In the rest of the chapter, I discuss algorithms that SKAT uses to generate articulations semi-automatically.

5.2 The Use of Articulation Rules for Query Answering

An important use for articulation rules is to answer queries those require information from multiple information sources. These queries can be posed either on the ontologies or on the instances of the concepts in the ontologies.

5.2.1 Ontology-level Query Answering

As indicated earlier, a large number of ontologies are available today. Typically, such ontologies are stored in a repository and retrieved from a server when necessary [Ont]. A key component of an ontology management and retrieval system is an articulation generator.

An ontology management system treats ontologies as first-class objects. That is, the system allows end-users to specify operations to manipulate ontologies and execute queries to select portions of ontologies stored in a repository. In the remainder of the dissertation, I will refer to queries that manipulate ontologies as ontology-level queries and refer to queries that are posed to information sources and select the information or instance data stored in the sources as instance-level queries.

A query-execution engine that executes an ontology-level query uses the articulation rules to match the ontologies before composing selected portions of multiple ontologies.

EXAMPLE 5.2.1 Ontology-level Query Answering: Consider the scenario where a buyer wants to buy several items from multiple suppliers. The buyer and each supplier have published his or her own ontologies. Each ontology lists a set of items, their attributes and

specifications, and the relationships between items in the ontology. In Figure 5.1, I show selected portions of the three ontologies:

The ontology named *Buyer_Ontology* contains a class *Door*, which has subclasses *Aluminum Door*, *Steel-frame Door*, *Glass Door*. There are two suppliers shown above: *Supplier1* and *Supplier2*. The first supplier *Supplier1* sells among other items two types of doors *Door1* and *Door2*. The frames of the two doors are made of *Wood* and *Steel* respectively. The second supplier *Supplier2* can supply doors as well. It stocks two types of doors: (i) *Metalic_Door* - doors made of *Aluminum* and (ii) *Wooden_Door* - doors made of wood.

Now, the buyer wants to answer the following query:
What are the items that the buyer wants to buy that can be supplied by the suppliers?

One possible plan to answering the query is as follows. First, obtain the intersections of the buyer ontology with each supplier ontology. Then, take the union of those intersections and from the union select the items that the buyer wants to buy. Articulation rules relating items that the buyer wants to buy (and are in the buyer ontology) to items that a supplier can supply (items in a supplier ontology) must be first established before the intersection of the buyer ontology with the supplier ontology can be computed.

For the ontologies shown above, the following describes how a buyer wanting to buy aluminum doors, steel-frame doors and glass doors for various parts of a new building being constructed can find out which of these the available suppliers can supply, namely *Supplier1* and *Supplier2*. The domain expert, using SKAT, identifies that *Supplier1.Door1* is a subclass of *Buyer_Ontology.Door* and *Supplier1.Door2* is a steel-door that is semantically matches *'Buyer_Ontology.Door.Steel-frame Door'* and that *Supplier2.Door.Metallic_Door* matches *'Buyer_Ontology.Door.Aluminum Door'*.

Using the above-mentioned rules, we get the intersections of the *Buyer_Ontology*, and *Supplier1* and *Supplier2* respectively as shown in Figure 5.2. The union of these two intersection ontologies is shown in Figure 5.3.

If the buyer is interested in buying steel-frame doors, aluminum doors and glass doors, a simple selection of those concepts, from the union of the intersections, indicates that only the steel-frame doors and aluminum doors can be supplied by *Supplier1* and *Supplier2* together and they cannot supply glass doors.

Thus, we see how articulation rules can be used to answer ontology-level queries. □

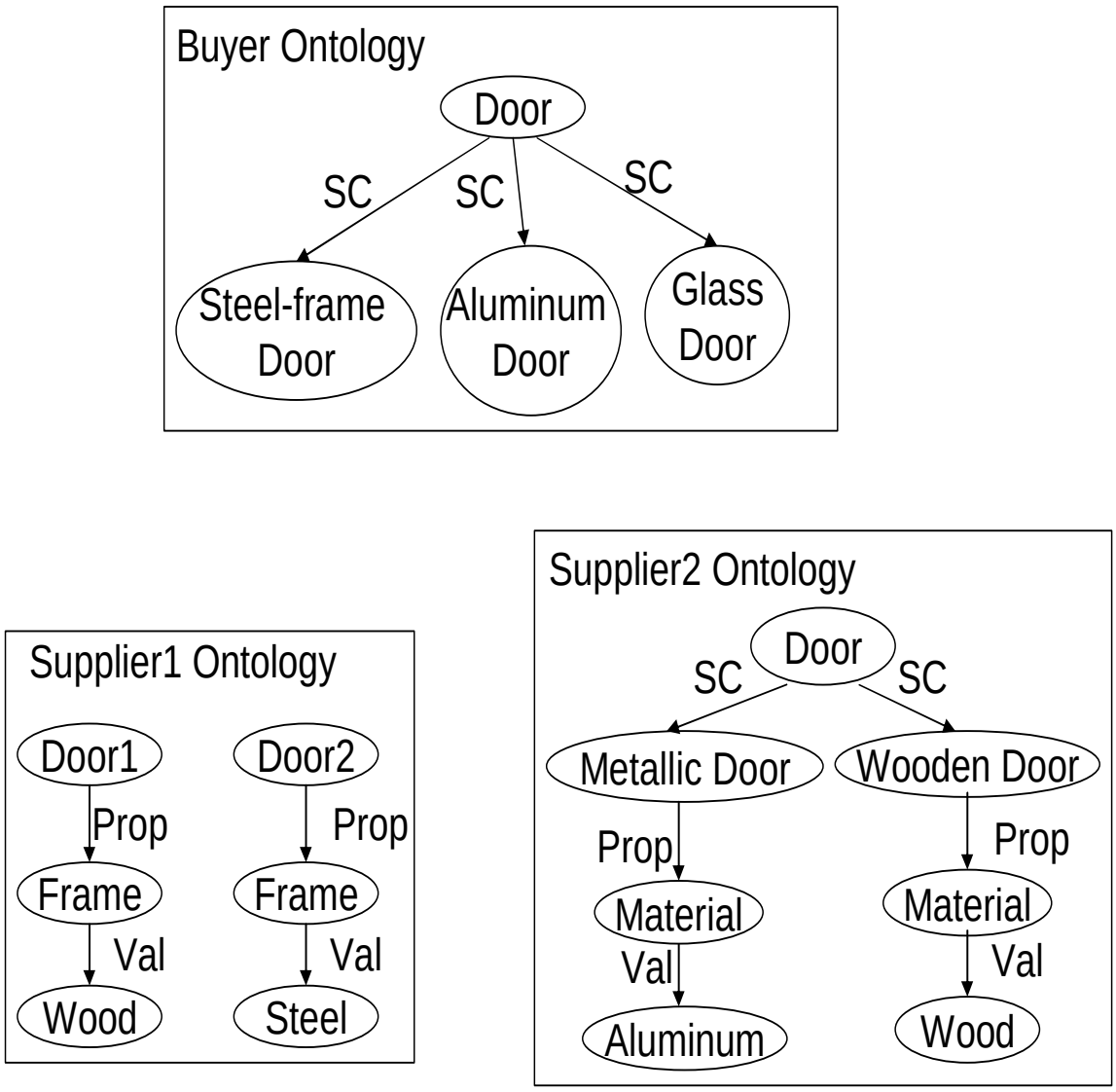


Figure 5.1: Three source ontologies: *Buyer*, *Supplier1*, and *Supplier2*

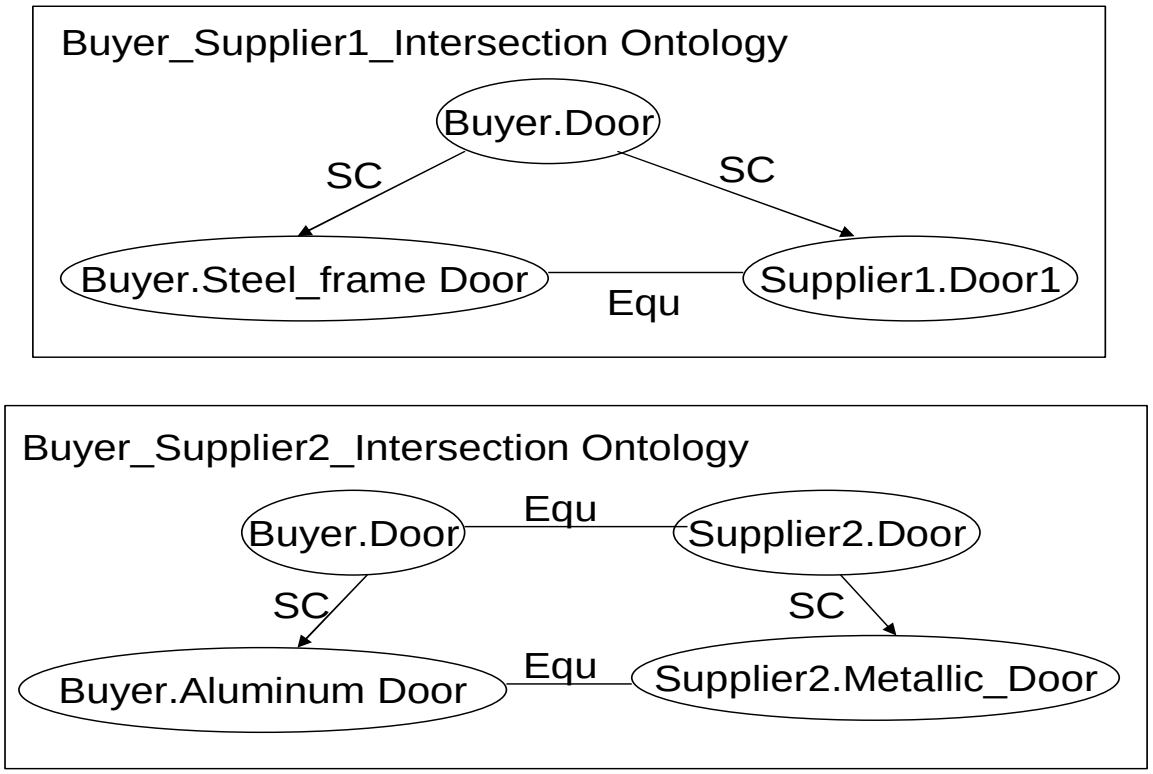


Figure 5.2: Pair-wise intersection of the Buyer ontology with the Supplier1 and Supplier2 ontologies

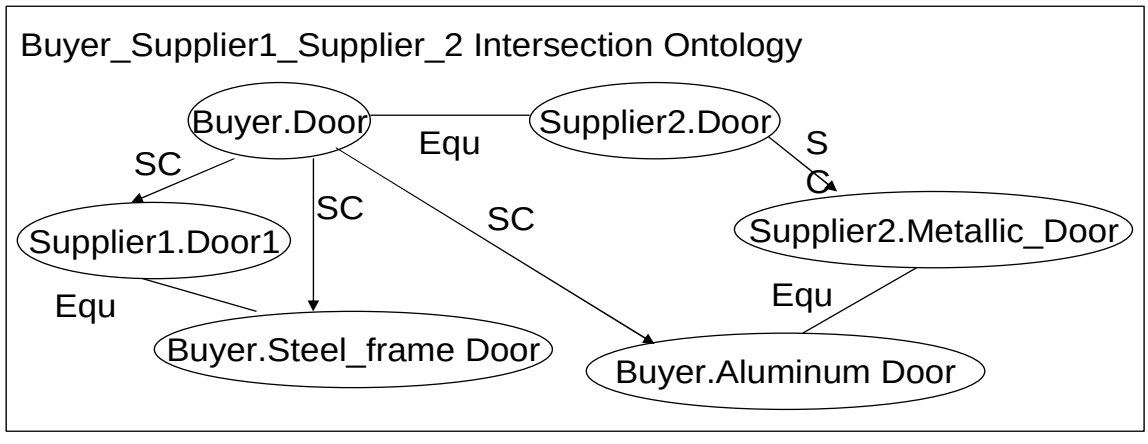


Figure 5.3: Intersection of the three ontologies Buyer, Supplier1, and Supplier2

5.2.2 Instance-level Query Answering

Articulation rules are also useful to answer instance-level queries. Typically, the end-users pose these queries. The results of the queries are individual instances (or their attributes) of the classes of objects present in the information sources. Whereas an ontology-level query can identify the suppliers those can supply the sports cars a buyer wants to buy, an instance-level query retrieves all individual instances of sports cars that all suppliers can supply and that meet the specifications of the buyer.

To answer queries using information sources in the presence of articulation rules that are first-order-logic rules I require an inference engine. However, if I use conjunctive queries as the query language and also the articulation-rule language, I can reduce the problem to that of answering queries using views used in information integration systems [LRO96] and in description logics [BLR97], [CGL99a]. There are two variations of the problem of answering queries using views that apply to answering queries using our ontology-based information systems: *global-as-view* (GAV) and *local-as-view* (LAV). I describe the two variations and provide examples of both below.

In the GAV approach, the concepts used in queries are obtained from some global ontology. The global ontology contains concepts from the source ontologies. In the LAV approach, terms used in the global ontology are different from those used in the source ontologies. The terms used in the source ontologies are defined based on terms in the global ontology. In this work, I have assumed that there is no stand-alone global ontology. Therefore, I utilize the union of the concepts in the source ontologies as a global ontology. Thus, the ONION framework conforms to the global-as-view model. I discuss the two models and give an example of how I use the global-as-view model below.

The Global-as-View Model

In the global-as-view model, the end-user poses a query using terms in the global ontology. An articulation rule relates a concept - the consequent of the rule - in one ontology to one or more concepts - appearing in the definition (antecedent) of the rule - obtained from other ontologies. If a concept appears in the query, and also in the consequent of an articulation rule, the system can rewrite the query. In a rewritten query, one or more concepts that appear as a consequent in an articulation rule is replaced with the antecedent of the articulation rule. Such a replacement creates another query equivalent to the original

query. All possible equivalent (yet distinct) queries are obtained by systematically replacing the concepts by their definitions. The answer to the original query is the union of the answers to all the equivalent rewritings of the query [Ull97]. ONION takes each query that needs evaluation and divides it into sub-queries. A sub-query contains all concepts in the original query pertaining to one ontology. The query processor then sends the each sub-query to individual sources for evaluation. The results to the sub-queries returned by the individual sources are then used to answer the main query.

EXAMPLE 5.2.2 For example, let $O1$ and $O2$ be two ontologies from the automotive domain that defines different types of cars. Now, consider an instance-level query

```
((?X InstanceOf O2.BlueCar) AND
 (?X InstanceOf O2.NewCar))
```

The query asks for all instances of new, blue cars. More formally, the query asks for all X such that X is an instance of the class *BlueCar* as defined in ontology $O2$ and the class *NewCar* as defined in the same ontology $O2$. Now, assume that a domain expert has supplied the following articulation rules that establishes the correspondences between instances of cars whose information is available in information sources conforming to ontology $O1(O2)$ and instances of cars whose information is available in information sources conforming to ontology $O2(O3)$:

```
( ((X InstanceOf O1.Car) AND
   (X O1.ColorOf "Blue"))
 => (X InstanceOf O2.BlueCar))
```

and

```
( ((X InstanceOf O3.Car) AND
   (X O3.ManufacturedIn Y) AND
   (Y Equals "2003"))
 => (X InstanceOf O2.NewCar) )
```

where X represents instances of cars and the variable Y represents the years the cars were manufactured in.

QueProc, ONION's query processor, will rewrite the query as:

```
((?X InstanceOf O1.Car) AND
 (?X O1.ColorOf "Blue") AND
 (?X InstanceOf O3.Car) AND
 (?X O3.ManufacturedIn Y) AND
 (Y Equals "2003")))
```

Since the query is a conjunction of several conjuncts, the query is divided into sub-queries each containing a subset of all the conjuncts as shown below. The first sub-query contains all the conjuncts that contain concepts from ontology $O1$, and the second sub-query contains all the conjuncts that contain concepts from ontology $O3$.

```
((?X InstanceOf O1.Car) AND
 (?X O1.ColorOf "Blue"))

and

((?X InstanceOf O3.Car) AND
 (?X O3.ManufacturedIn Y) AND
 (Y Equals "2003")))
```

The first sub-query is posed to the source corresponding to ontology *O1* and the second sub-query is posed to the source corresponding to ontology *O3*. The intersection of instances bound to the variable X in the results returned by the first source and those instances bound to X in the results in the second source are returned as the answers to the original query. □

The Local-as-View Model

The local-as-view variation requires a global ontology based on which the concepts of each source ontology are defined. As argued before, the ONION framework does not assume the presence of a global ontology and therefore does not use this model. However, for the sake of completeness, I describe the paradigm briefly below.

EXAMPLE 5.2.3 Consider the following example:

```
((?X InstanceOf OG.Car) AND
  (?X OG.SoldIn "CA"))
```

Now, assume the following concepts definitions (analogous to database views) from the ontologies are available:

```
((X InstanceOf OG.Car) AND
  (X OG.ColorOf "Blue")
=> (X InstanceOf O1.BlueCar))
```

and

```
((X OG.SoldIn "CA")
=> (X InstanceOf O2.CASales))
```

It is easy to see that the query can be answered using the concept definitions as follows:

```
((?X InstanceOf O1.BlueCar) AND
 (?X InstanceOf O2.CASales))
```

□

This problem has been studied in the context of information integration, in knowledge-bases [AYL95], databases[LMSS95], and description logics [CGL99b], and is commonly referred to as answering queries using views. Several existing optimal algorithms can perform the rewriting and evaluation of queries in such a setting. However, since ONION does not assume the existence of a global ontology and thus does not support the *local-as-view* model, I will not elaborate on these algorithms any further.

5.3 Storage of Ontologies and Their Articulations

ONION does not store the individual ontologies corresponding to the sources it seeks to interoperate but assumes that the ontologies are available to it and can be fetched when required. This choice of storage of ontologies ensures that the ontologies are stored and maintained at their sources and provides greater autonomy to the publishers of the ontologies. Because ONION fetches ontologies only when needed, the fetched ontologies contain the latest versions made available by the individual sources. Fetching the latest version of an ontology helps ONION ensure that it avoids the problems that can occur when decisions are made based on stale information.

As mentioned in Chapter 3.2.2, the articulation rules are stored in OntoStore for future use. Since ONION does not have the permissions to store the articulation rules at the locations where the ontologies themselves are stored, the articulation rules are stored at a central repository accessible to ONION.

The articulation rules may be stored in a central repository or in a distributed one — the choice is orthogonal to the current work. For the current work, I assume that all articulation rules are stored in a central repository. When an end-user poses a query to the system, the query-rewriting engine translates the query using the articulation rules such that each translated query or its sub-queries uses the same vocabulary as that used in the individual information sources.

In the next section, I discuss how articulation rules are generated semi-automatically.

5.4 Semi-automatic Generation of Articulation Rules

As indicated in the previous chapters, often different sources use different terminologies to describe the objects in the sources. The same term, used in different sources, often has overlapping or somewhat different semantics, e.g., the term "nail" has entirely different semantics in a "cosmetics" ontology and the "carpentry" ontology. Similarly, different sources, often, use different terms to refer to semantically similar objects, e.g., the terms "truck" and "lorry" in two transportation ontologies might refer to the same class of objects. Even though ontologies expose some of the semantics of the terms and their relationships, they often remain incomplete or inadequate if we consider the needs of the various applications that use them.

In this section, I will discuss a semi-automated algorithm for resolving the terminological heterogeneity among the ontologies and establishing the articulation rules necessary for meaningful interoperation. This algorithm forms the basis for the implementation of SKAT. The algorithm uses multiple heuristics to generate articulation rules. Such heuristics are based on the terms and relationships used in the ontology, the structure of the ontology graphs, and the instance data available for each ontology concept. No single heuristic works best across all domains. Empirical evidence shows that combining the information obtained by using multiple heuristics provides a better match between semantically related terms in the ontologies than using a single heuristic.

5.4.1 Application-specific Articulations

As mentioned above, ONION builds application-specific articulations using SKAT and OntoComp. Application-specific articulations are focused and retain only information that is necessary for the application. For example, ontologies on two travel sites might list the diferent products it has to offer that will help a user make air, car and hotel reservations. However, if a business application only wants to rent cars locally for its business purposes, it is not interested in the air and hotel portions of the ontologies. If the source ontologies are large, then creating and maintaining articulation rules for the air and hotel portions of the source ontologies would incur additional cost and unnecessary overload. To avoid such unnecessary costs, I build ontologies that are cognizant of the needs of the application and, in our example, retain only articulation rules regarding car rentals.

Application-specific articulations also result in more accurate results, since the articulations are based on the specific needs of the application. Consider the following example. There are sources of information about flights that take place between two destinations. A civilian application that needs to book tickets for passengers on a flight ignores a sortie that a military unit will conduct and only considers a flight to be undertaken by a commercial airline, for the civilian can not travel in military aircraft. Thus an articulation-rule generator that is articulating two ontologies for such a civilian application should not try to match the two types of flights that take place. However, when the military wants to move personnel from one location to another, it can avail itself of either option depending upon space and cost considerations. An articulation-rule generator that is matching two ontologies for a military application will match a "sortie" with "commercial flights" and may establish a rule that says that, for the purposes of this application, a "flight" and a "sortie" can be used interchangably.

Even two experts creating applications using the same source ontology for similar clients in the same domain may not agree on the desirable semantics for the applications. The primary objective for one client of a commercial travel agency is minimizing the cost, while for another it is a combination of cost and frequent flyer miles. Similarly, two organizations using the same commercial airline for their transportation needs differ significantly when their objectives are different, – say one is interested in moving goods and freight while the other is primarily interested in moving personnel. While building an interoperation system, it is important to cater to the differences among different clients, organizations and experts. In our framework, since ONION builds articulations only with respect to

a particular application, the differences of opinion between two experts can be respected. Each expert has the freedom to build articulations that cater to the needs of the semantics of the application. It is no surprise, then, that two ontologies can have multiple articulations, each tailored towards a particular application.

Thus, in this sub-section, I have argued for the creation of multiple articulations between two ontologies depending upon the applications the articulations will be used for.

5.4.2 Primitives Used in Articulation Rules

The preliminary articulation rules generated by SKAT are of two types - ones that are simple statements of the form *(Match 'Department of Defence' 'Defense Ministry')* expressing matches between equivalent concepts and the more complex rules expressed as conjunctive queries that the domain expert supplies.

SKAT uses the relation *(Match Concept1 Concept2)* to indicate that the two concepts *Concept1* and *Concept2* are related (above an acceptable threshold supplied by an expert and the threshold can vary from application to application). *Match* does not indicate the exact semantic relationship between the two concepts, for example, whether they have a class-subclass relationship, or are equivalent etc. Therefore, *Match* gives a coarse relatedness measure and must be refined to more precise semantic relationships, if such refinement is required by the application. Such refinement can be performed by more automatic functions or derived using the assistance of a human expert.

For example, the human expert might indicate that by default all concepts that *Match* are to be taken to be equivalent unless otherwise noted by the expert. In the current implementation, I depend upon the human expert to modify the *Match* relations to more precise relationships using the ONION viewer. The relationships used in articulation rules can be any of the special relationships listed in Chapter 2, i.e., any of *(SubClassOf, PartOf, AttributeOf, InstanceOf, ValueOf)* . Additionally, the articulation rules use the two primitives

- *EquivalentTo* and
- *Equals*.

The relationship *EquivalentTo* holds between two classes of objects and expresses that the classes are equivalent. Two classes are equivalent if the set of real-world entities they

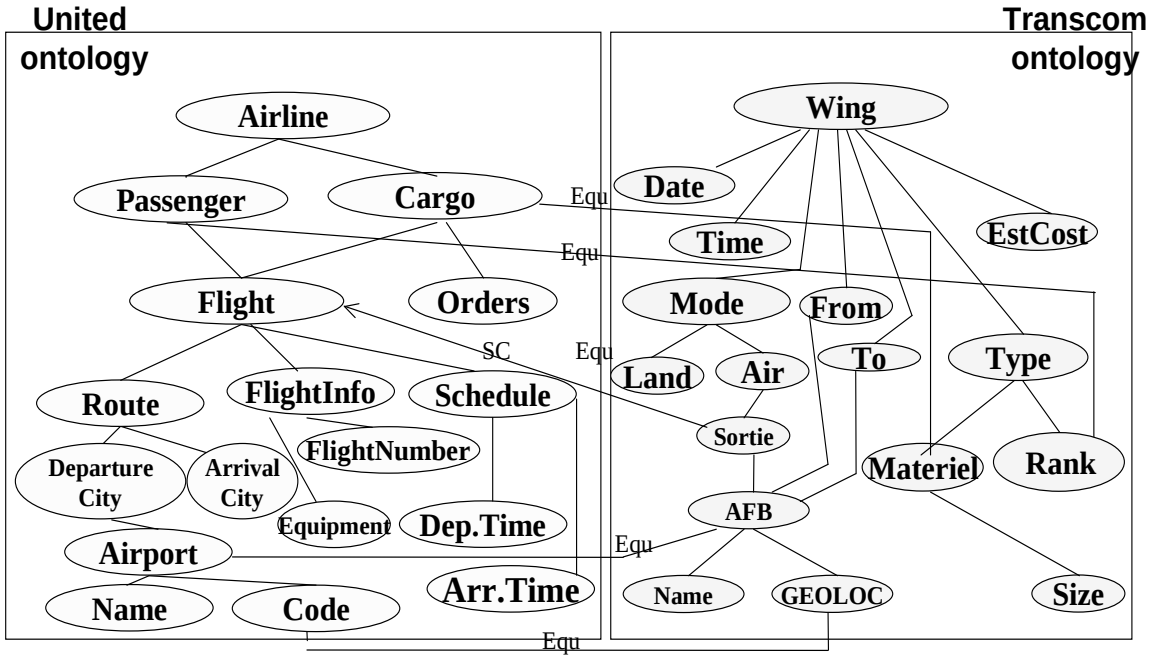


Figure 5.4: An articulation between the United Airlines Ontology and the TRANSCOM Ontology

represent are exactly the same. The relationship *Equals* holds between a class or a variable and a value, e.g. *(Y Equals "40,000")* indicates that the variable Y takes the value "40,000".

The articulation rules may also contain a relationship from the individual source ontologies provided a reference to the source ontology is included in the articulation rule. For example, the relationship *OG.SoldIn* is used in an articulation rule above. Note, that the ontology reference *OG* is used along with the name of the relationship *SoldIn* to unequivocally specify which relationship is being referred to.

5.4.3 An Example

In Figure 5.4, I show an example articulation. On the left hand side, is a portion of the United Airlines Ontology and on the right a portion of the TRANSCOM Ontology. These ontologies were constructed manually for experimentation. The objective of the application is to transport military men and materiel from Washington D.C. to Al Jabar Airbase in Kuwait. A combination of commercial flights and special purpose sorties is to be used to meet the transport objective.

The United Airlines source has *flight*, whose *DepCity* is *Washington D. C. (IAD)* and *ArrCity* is *Frankfurt(FRA)*. This corresponds to a flight from Washington D. C. to Frankfurt. There exists an articulation rule, supplied by the domain expert, that says that the *connection* relation is transitive.

The TRANSCOM source has a *sortie* that runs from *Rhein Main AFB* in Frankfurt, Germany to *Al Jabar* airbase in Kuwait.

SKAT establishes the articulation rules semi-automatically. The rules indicate that the *Frankfurt* airport is the co-located with the *Rhein Main AFB*. It tells us that if there is a *sortie* or a *flight* between two cities, then there is a *connection* between them. It also indicates that *DepCity* in the United ontology is the same as *From* in the TRANSCOM ontology. Due to lack of space in the figure, the rule that states that *United.ArrCity* is equivalent to *TRANSCOM.To* is not shown.

Using these rules, an inference engine can easily establish that there is a connection between *Washington D. C.* and *Al Jabar Airbase, Kuwait.*

The automated tool generates and suggests the simpler articulation rules to indicate the terms in the two ontologies that are related. The expert then validates these suggestions and the final set of articulation rules are stored to be used for composing ontologies and answering queries posed to the information sources.

5.5 Generation of Ontology Articulations

As indicated in the last chapter, ONION employs SKAT to generate a set of articulation rules between two source ontologies. SKAT uses several matching algorithms and provides the human expert with a confidence score associated with each matching rule it suggests. The scores from the different matchers are combined to determine the overall matching score between terms across ontologies.

OntoComp or QueProc recalls the articulation rules from the repository when the task involves composing portions of the two ontologies.

Since SKAT is modular in nature, one can plug any application-specific matching algorithm in to the driver that generates the articulations. I believe that a set of basic matching algorithms will be useful in a wide variety of applications. In the rest of the chapter, I first describe a windowing algorithm that is used in order to perform the matching scalably and then show a set of heuristic matching algorithms and empirical evidence of their efficacy.

The heuristic matchers used by SKAT are of broad types - iterative and non-iterative.

5.5.1 The Windowing Algorithm

The algorithms described in the rest of the chapter are based on computing the similarity score between all pairs of terms, such that the two terms in a pair are from different ontologies. For very large ontologies, this quadratic computational complexity is unacceptable since it gets very time-consuming and expensive. To avoid such cases, and make the algorithms more scalable, SKAT deploys a "windowing" algorithm as described below.

A *window* consists of a connected subgraph of an ontology graph and the set of rules in the ontology such that the concepts and relationships used in the rules all appear in the subgraph. SKAT partitions each ontology into several windows, and constructs pairs of windows by selecting one window from each ontology such that a window does not appear in two such pairs. The concept names in a window is matched to the concept names that appear in the paired window. The computational complexity of the algorithm is reduced by not computing the similarity between all pair of terms but by computing the similarity only between pairs of terms where two terms in a pair are from "corresponding" paired windows.

The expert provides SKAT with an estimate of the number of terms permissible in a window, say N. The expert is guided by the fact that the algorithm will compute $O(N^2)$ similarity scores between pairs of terms for each pair of window. I assume that the expert knows the sizes of the ontologies and the maximum amount of computational resources available (maximum computational complexity permissible) and makes a judicious choice of N.

SKAT performs the following computation. Let an ontology O_1 have $|O_1|$ number of terms, and is being articulated with another ontology O_2 having $|O_2|$ terms. Without loss of

generality assume, $|O_1| \geq |O_2|$. O_1 is divided into P windows of approximately equal number of terms T. SKAT constructs the windows by traversing the ontology graph in a breadth-first manner. The first T terms visited are put into first window, the next T terms in the second window and so on. SKAT selects P by choosing the minimum P that satisfies:

1. $P * T \leq |O_1|$, and
2. $T \leq N$.

O_2 is also divided into the same number of windows as O_1. The windows in each ontology are numbered sequentially according to the order of their creation from 1 to P.

By default, in the next stage of the algorithm, SKAT tries to match the i^th window of O_1 to the windows numbered $i-k, i-k+1, \ldots, i+k$. The domain expert provides the value for k. Clearly, a larger value of k increases the amount of computation. The assumption in this matching is that the two ontologies have the same granularity and are similar in their structure. That is why, the top window of one ontology is matched to the top windows of the other and the bottom window with the bottom windows of the other. If this assumption is not true, the expert can override this default behavior by creating windows in the source ontologies using a GUI and indicating the pairs of windows that he or she wants SKAT to match. SKAT finds the articulation rules between terms taken from the indicated windows.

Currently, if windowing is done, SKAT cannot match large ontologies with different granularities. In the future, the work can be extended to support matching ontologies with different granularities, by enabling the expert or the automated component to omit subgraphs of concepts from the more detailed ontologies.

5.5.2 Non-iterative Algorithms

Non-iterative algorithms are ones that identify the matching concepts in the two ontologies in one pass. Our linguistic matcher employs only non-iterative algorithms since the techniques SKAT uses do not generate additional matches if more than one iterations are allowed.

Linguistic Matching

The linguistic matcher looks at all possible pairs of terms from the two ontologies it is matching and assigns a similarity score to each pair. If the similarity score is above a threshold, then the linguistic matcher generates an articulation rule and forwards it to the expert for verification. Increasing the threshold decreases the number of articulation rules suggested, and decreasing the threshold reduces the number of articulation rules suggested. The expert is in charge of increasing or decreasing the threshold.

The linguistic matcher expects a concept name to be represented as a string of words. The matcher constructs all pairs of words where the two words in a pair come from strings obtained from different ontologies. The matcher uses a word-similarity table generated by a *word relater*, which I describe below. The matcher looks up a word-similarity table to

determine the similarity between all such pairs of words. Finally, the matcher computes the similarity of the strings based on the similarity of the pairs of words.

The lexical matcher computes the similarity between pairs of strings *s1* and *s2* as follows. First, it strips the strings of all stop-words: prepositions, conjunctions, pronouns, and interjections. A word is a continuous run of alphanumeric characters in a string that may be hyphenated. Spaces or underscores demarcate words in a string. Stop-words are identified by looking up a table of stop-words. I constructed a table of stop-words containing common prepositions, conjunctions, pronouns, and interjections. This table can be extended in the future by adding more words.

The rest of the algorithm is run on these stripped strings. After the stripping, let *s1* be the string with fewer number of words between the two strings that are being matched . After obtaining the similarity between pairs of words, SKAT obtains the best "matching" between the words. The best matching is found by successively selecting the pair with the highest similarity score such that the words in the pair have not occurred in any pair selected before. Finally, SKAT sums the existing matching scores and obtains the average by dividing the sum by the number of words in *s1*. This average gives us the matching score between *s1* and *s2*.

The detailed algorithm is as follows:

- match(String s1, String s2, WordSimilarityTable wst)
  - List similarityList;
  - for each word w1 in s1:
    - for each word w2 in s2:
      similarityScore ← wst.lookup(w1, w2);
      Add (w1, w2, similarityScore) to similarityList;
  - Sort similarityList on the similarity score of the tuples;
  - Set matchedWords ← null;
  - floatingPointNumber matchingScore ← 0.0;
  - for each tuple (w1, w2, ss) in similarityList:
    - if both w1 or w2 is in matchedWords continue;
    - else
      matchingScore ← matchingScore + ss;

add w1, and w2 to matchedWords;

- similarityScore ← similarityScore / min(size(s1), size(s2));
- return similarityScore;

For example, say, given the strings *Department of Defense* and *Defence Ministry*, the algorithm generates a matching score between the words: $match(Defence, Defense) = 1.0$. Furthermore, it generates $match(Department, Ministry) = 0.74$. I will describe algorithms below to compute the similarity between words below. The heuristic algorithms, used by SKAT, generate these numerical similarity scores. They indicate the degree of similarity as computed by the algorithms between the two words. Given the similarity between the words as computed by the algorithm, SKAT calculates the similarity between the two strings as follows:

```
match("Department of Defense",
      "Defence Ministry") = (1 + 0.74)/2 = 0.87.
```

The denominator is the number of words in the string with fewer words between the two strings.

If the generated similarity score is above a pre-supplied threshold, then the two concepts1 are said to match, and SKAT generates an articulation rule,

$$((Match\ \text{“}Department\ of\ Defence\text{”}\ \text{“}Defense\ Ministry\text{”}), 0.87),$$

where 0.87 is the confidence measure with which SKAT generated this match. The confidence measure varies betwen 0 and 1.

Constructing the Word-Similarity Table:
I have experimented with several ways to generate the table containing the similarity between all pairs of words. First, SKAT checks if the words are spelled similarly, and then derives word similarity using methods that can be differented into two main classes: a) thesaurus-based, and b) corpus-based.

Thesaurus-Based Word-Relator: I have devised matching algorithms based on dictionaries or semantic networks, like Nexus [Jan00] and WordNet [wor]. WordNet gives us a list of synonyms for each word. If the two words are found to be synonyms, then SKAT returns a similarity score of 1.0. If the words are found to be synonyms from WordNet, which is a manually generated thesaurus, during our experiments, we took the matching score of

the words to be 1. Now, whether doing so makes sense is arguable. In our experimental data, the we did not see any accidental matches between different word senses of the same word. The value of the matching score for the synonym lookup is a configurable parameter and can be modified to any other number by the expert.

If the two words are not synonyms, it looks at the the number of words that are "similar" in the defnitions of each word. The matcher repeats this process of looking into the definitions of words to find their similarity recursively until a specified recursion depth is reached at which point SKAT requires the word-similarity to be supplied to the algorithm. The definitions of words are treated as strings and we use the string matching algorithm described above. To generate the word-similarity used in the leaf-level of the recursion tree, SKAT first checks whether the words in the pair are stated as synonyms in WordNet [wor]. If WordNet indicates that the words are synonyms, their similarity score is 1.0. If the word pair does not occur in WordNet, the tool generates a word-similarity score for the pair of words using the Corpus-Based Word-Relator, that gives a score for the degree of similarity of words between 0 and 1. Any other dictionary that provides synonyms can be used to replace WordNet. We initially experimented with the Nexus system [Jan00] but later on replaced it with WordNet.

The detailed algorithm is as follows:

- GenerateSimilarity(word w1,word w2,dictionary dict,depth dep)
  - if w1 and w2 are synonyms in dict, return 1;
  - if (dep == 0) return similarity-score using corpus-based method;
  - else
    - def1 ← dict.lookup(w1);
    - def2 ← dict.lookup(w2);
    - List similarityList ← new List;
    - for each word wd1 in def1:
      - for each word wd2 in def2:
        Add (w1, w2, GenerateSimilarity(wd1, wd2, dep-1)) to similarityList;
    - Sort similarityList on the similarity score of the tuples;
    - Set matchedWords ← null;

* floatingPointNumber matchingScore ← 0.0;
* for each tuple (w1, w2, ss) in similarityList:
  - if either w1 or w2 is in matchedWords continue;
  - else
    matchingScore ← matchingScore + ss;
    add w1, and w2 to matchedWords;
* similarityScore ← similarityScore / min(size(def1), size(def2));
* return similarityScore;

EXAMPLE 5.5.1 We saw above how the similarity between the terms *"Department of Defense'* and "Defence Ministry" was determined using WordNet. We now show another example where there are no synonyms like "Department" and "Ministry" and the words have some overlap but are not very closely related for the purposes of the application. While generating articulation rules for an application in the transportation domain, SKAT encountered the terms "truck" and "boat". The definitions of *truck* and *boat* are "an automotive vehicle suitable for hauling", and "a vessel for water transportation" respectively in WordNet. If the specified depth is 1, SKAT does not look into the definitions of *vehicle* and *vessel* to determine their similarity. Since they are not exactly the same, SKAT sets their similarity to 0 (or gets their similarity scores using a corpus-based method if one is available). If however, the depth were set to 2 (or more),

$$match(truck, boat) = match(\quad \text{“}automotive\ suitable\ hauling\text{”}, \\ \text{“}vessel\ water\ transportation\text{”}\ )$$

Note that the stop-words have not been shown since, as mentiioned above, SKAT strips them before matching the two definitions. To match the two definitions SKAT invokes the algorithm as indicated above where SKAT treats each definition as a string and identifies the matching score between the strings by breaking down the definitions into its component words. SKAT looks up the definitions of all pairs of words such that each word in a pair is taken from a different string. For example, SKAT tries to match *vehicle* and *vessel*, discovers their definitions both have *transportation* in common, and generates a similarity score between *vehicle* and *vessel* while no such similarity is found between *water* and any word in the definition of *truck*. The algorithm then combines the similarity of the words in the definitions to find the similarity score between the two definitions, and propagates that similarity up to generate a non-zero similarity score between *truck* and *boat*.

□

Corpus-Based Word Relater: The linguistic matcher also generates word similarities using a corpus-based matching algorithm. The word relater uses a corpus of documents belonging to the domain of the ontologies that are being matched. SKAT generates similarities among terms in the corpus based on the context in which the terms occur [Sch92].

I identify the *context* in which a word, w, appears by looking at words that appear in a 1000-character neighborhood of all occurrences of w in documents in the corpus. The 1000-character neighbourhood of a word w contains all complete words that appear in the 500 characters preceeding w and also all complete words that appear int he 500 characters succeeding w. For example, consider the word "looking" in the first sentence of this paragraph. We now show the 50-character neighborhood of the word "looking". The words "a", "word", "w", "appears", and "by" appears in the 25-character neighborhood preceeding the word "looking" and the words "at", "words", "that", "appear", and "in" appear in the 25-character neighborhood following the word "looking". Together, they constitute the 50-character neighborhood of "looking". Note that the word "which" appears partially in Together, they constitute the 50-character neighborhood of "looking". Note that the word "which" occurs partially the 25-character neighborhood preceeding the word "looking" and therefore is not included in the 50-character neighborhood of "looking". Words that appear partially in a neighborhood are not considered to be in the context of the word whose neighborhood SKAT is constructing. In a pre-processing step, the word relator looks at all words that appear in all the documents in the corpus. For each occurrence of a word w, the relator identifies the words in context of w. The number of rows in the context vector, V_w, of a word w is equal the number of words in the corpus. Let $V_w[i] = c$. This implies that the i^{th} word in the corpus occurs with a frequency c in the 1000-character neighbourhood of the word w. Let the word w occur n times in the set of documents out of which in m contexts of w the i-th word appears. $c = m/n$ and is a fraction between 0 and 1. The cosine of the context vectors of two words gives a measure of the similarity of contexts in which the two words appear [RW86]. SKAT's word relator uses this similarity measure to generate a matrix of word similarities that gives the similarities between all pairs of words. The linguistic matcher then uses this matrix. The generation of the word similarity matrix is done as a pre-processing step to avoid incurring the cost at runtime.

Ideally, along with each ontology, SKAT would have access to a corpus of documents. The terms used in the documents appear in the ontology. The semantics of the terms are

specified in the ontology. In such a case, SKAT can derive the context vectors of the terms in the ontology from its associated corpus of documents. However, for the experiments that I ran, the ontologies did not have an associated corpus of documents.

If a corpus is not supplied with an ontology, SKAT's word relater generated a corpus associated with an ontology by searching the web using the search-engine Google. SKAT performed the search using 10 keywords and retrieves 200 documents. SKAT uses 10 keywords because Google allows using a maximum of 10 keywords for any particular search. An expert assisting the word-relater chose the 10 keywords carefully so that the searching the searching the web with those keywords would return a set of documents that use the terms that occur in the in the sense in which they are used in an ontology. Empirical evidence shows that a corpus of 200 pages proved adequate to produce good matches.

The linguistic algorithms match the concepts based on their names. Therefore, it is imperative that the constructor of the ontologies chooses very accurate concept names. That is, the concept name should accurately reflect the semantics associated with the concept for the linguistic matcher to generate accurate matches. Often, a concept name comprised of few words is inadequate to capture the exact semantics of a concept. In this scenario, if each concept in an ontology is described using natural text, the matching of the descriptions will result in more accurate matches assuming that the descriptions themselves are precise.

Performance of the Linguistic Matching Algorithm: SKAT uses linguistic matching algorithms that do not have a theoretical limit on the number of words a concept name can have. The runtime of the algorithm is quadratic in the number of words in the ontologies. Since the articulation rules are usually generated "offline" much prior to their actual use, a quadratic runtime is acceptable if the number of words is reasonable. The time taken to determine the matches, when the number of words is large, is unacceptable to applications that need the matches to be determined in real time, but such uses are rare in our case. Besides, if the concept names have a large number of words, and are in essence descriptions of concepts, a better and faster matching can be produced by using techniques used for determining document similarity in information retrieval.

Analysis of the Linguistic Matching Algorithm: The linguistic algorithm discussed above break up each concept name into a list of words and then determine the similarity scores of all possible pairs of words such that two words in a pair belongs to two different ontologies.

To avoid computing the similarity between a pair of words multiple times, SKAT resorts

to dynamic programming. SKAT stores the similarity between a pair of words upon generating it and reuses that similarity value instead of re-computing it if the similarity between the same pair is required in the future.

Another optimization that the linguistic matcher uses is as follows. It pre-processes the similarity scores between all pairs of words in the corpora once and stores these similarity scores in the OntoStore repository. The number of such similarity scores is large and the pre-processing step saves the computation of these similarity scores during run-time that is, when the expert invokes SKAT, which in turn invokes the linguistic matcher. In this scenario, while computing the similarity between concept names, SKAT simply looks up the similarity between individual pairs of words instead of computing them. Since looking up the similarity between a pair of words is cheaper than computing the similarity on the fly, the algorithm runs faster.

SKAT performs the above-mentioned optimization whenever possible. Of course, SKAT performs the optimization only if the following conditions are satisfied:

- The corpus of documents is available before the articulation is generated, and
- The cost of pre-computing the similarity scores of all word pairs is acceptable to the application.

The cost for pre-computing the similarity scores for all pairs of words is quadratic with respect to the number of words. If the number of words in the corpora are not too large in comparison to that of the two ontologies, the application would be better off by pre-computing the similarity scores instead of computing them on demand. The tradeoff between how much extra processing - introduced by the pre-computing - is acceptable is dependent upon the needs of the application and the amount of resources available to the system. An application that caters to clients for whom the ontologies must be matched in real-time would pay the extra cost to pre-process the similarity scores.

5.5.3 Iterative Algorithms

Iterative algorithms are algorithms that perform multiple iterations over the two source ontologies in order to generate articulation rules between them. Articulation rules generated in one iteration are used to generate additional articulation rules in subsequent iterations. Since the non-iterative algorithms alone cannot generate all desired matches without generating false matches, SKAT uses a combination of strategies to reinforce correct matches,

generate missed matches, and eliminate false matches.

Structure-based Heuristics

These algorithms look for structural isomorphism between subgraphs of the ontologies to find matching concepts. For the ontologies I have experimented with, I observed that a purely structure-based matcher - one that simply looks for isomorphism between subgraphs in the ontologies without considering concept names - performs very poorly and is inadequate.

Therefore, SKAT uses a structure-based matcher that it calls after some preliminary matches generated by other matchers, like the linguistic matcher are available. The algorithm used by SKAT to match concepts in two ontologies *O1* and *O2* based on their structure are as follows: Let *O1.A* be a node in ontology *O1*. Let A_1 be a set of nodes in ontology *O1*, related using the relationship R to a concept represented by *O1.A*. That is, in the ontology graph, each element of $O1.A_1$ is related to node *O1.A* using the relationship R. Similarly, let $O2.B_1$ be the set of nodes related using the same relationship R to a concept represented using a node *O2.B*. Let T be a threshold percentage supplied by the expert. Let $O1.A_1(O2.B_1)$ have fewer elements than $O2.B_1(O1.A1)$. SKAT computes the percentage p of the number of attributes in the set $O1.A_1(O2.B_1)$ that have a matching attribute in the set $O2.B_1(O1.A1)$. If $p > T$, then generate a match between *O1.A* and *O2.B*.

For example, consider the following selections from two ontologies *O1* and *O2* respectively:

```
(CreditCard ON.offeredBy Bank)
(Visa SubClassOf CreditCard)
(MasterCard SubClassOf CreditCard)

(CreditAccounts ON.offeredBy CreditUnion)
(VisaAccount SubClassOf CreditAccounts)
(MasterCardAccount SubClassOf CreditAccounts)
```

ON.offeredBy is the relationship *offeredBy* whose semantics is defined in a third ontology *ON*. Both *O1* and *O2* uses this relationship. Now assume SKAT has generated the following rules:

```
(Match Bank CreditUnion)
```

```
(Match Visa VisaAccount)
(Match MasterCard MasterCardAccount)
```

The structural matcher considers the two concepts "CreditCard" and "CreditAccounts" and sees that their parents match (*Bank* and *CreditUnion*). All their children match on a one-to-one basis (*Visa* with *VisaAccount*, and *MasterCard* with *MasterCardAccount*). The matches between the parents and the children of the nodes causes the structural matcher to generate the match
(Match CreditCard CreditAccounts).

Currently, SKAT uses the algorithm indicated above where the relationship R is one of {*AttributeOf, SubClassOf*} or where the corresponding relationships match exactly (in the example above, *ON.offeredBy*). Although the articulation model does not interpret a relationship called *superClassOf*, SKAT runs the algorithm above in both "directions" for the *subClassOf* relationship. That is, if the subclasses of nodes *O1.A* and *O2.B* match, SKAT increases the similarity score between *O1.A* and *O2.B*. Similarly, if the superclasses of *O1.A* and *O2.B* match, SKAT increases the score between *O1.A* and *O2.B*. For example, if the linguistic matcher has matched nodes, *O1.A* and *O2.B* in the ontology-graphs, the structural matcher looks to match their children (also parents), *O1.C*, and *O2.D*, if the linguistic matcher has not already matched them. If a substantial percentage (above a threshold percentage supplied by the expert) of the parents of *O1.C* have been matched with those of *O2.D*, and the children of *O1.C* have been matched with those of *O2.D*, then an articulation rule matching *O1.C* and *O2.D* is generated.

Expert-supplied Articulation Rules

SKAT is a semi-automatic articulation-rule generator and depends on expert-supplied feedback. SKAT accepts expert-supplied articulation rules and uses them as the starting point to generate further articulation rules using the iterative algorithms described above.

Pre-Processing: The expert can help the tool preprocess the ontologies before the automated articulation generation algorithms try to match them. In the pre-processing step, the tool removes stop-words and stems words to their root words. The expert can choose and edit a list of suggested stop-words and either provide a stemming procedure or specify a table (or individual rules) of root words.

Seed-Rules: The expert can supply an initial set of seed rules that contain information specific to the needs of the application that will use the articulation. For example, a rule

```
(Match US.President FRG.Chancellor)
```

indicates that the expert wants the matcher to match the node representing the US President with that representing the German Chancellor. Similarly, a rule like

```
(Mismatch Human.nail Factory.nail)
```

indicates that the expert does not want the term nail from the Human ontology match with the term nail in the Factory ontology. A expert supplies a *Mismatch* statement to instruct SKAT to override matches generated by the SKAT's automated component. In case a *Mismatch* statement conflicts with an articulation rule provided by a human expert, the conflict is logged and SKAT halts. A human expert has to resolve the conflicting rules and correct them before SKAT can proceed any further.

Context Identifier Removal Rules : In the experiments described below, an application sought to match ontology graphs obtained from the government websites of NATO countries to identify government departments and entities that are similar in their function in the different countries. The expert wanted SKAT to generate a match between the parliaments of two countries. For example, the expert wanted SKAT to match the node labeled 'Finnish Parliamentary System' in the ontology graph of Finland with that labeled 'The UK Parliamentary System' in the ontology graph pertaining to the United Kingdom. The expert supplied a set of preprocessing rules that remove the "context-identifiers" from the labels of the nodes. Specifically, SKAT reduced the label such as 'Finnish Parliamentary System' to 'Parliamentary System' and the 'The UK Parliamentary System' to 'The Parliamentary System', thereby paving the path for the matcher to generate a match.

5.6 Experiments & Results

SKAT uses an implementation of the linguistic methods and the structural methods (using Java as the programming language). I experimented with three sets of ontologies represented in RDF[rdf99] on a Linux PC platform (650 MHz Pentium III, 256 MB primary memory) :

1. Transportation Source: Ontologies constructed manually to represent a domestic airline (terminology used on United Airlines website) and a Air Force ontology (terminology used by TRANSCOM).

2. Government Source: Ontologies constructed manually from the NATO government web sites representing each web-page associated with an department of the government

as a node. The edges in the ontology graph were derived from the links between the pages.

3. Construction Source: Ontologies constructed from civil engineering catalogues. One ontology contains terms referring to supplies that a supplier carries. The second ontology contains terms referring to objects that a purchaser wants to procure.

The United Airlines ontology was manually constructed using information from the website http://www.ual.com . A human ontology constructor picked phrases from the airlines web pages that represents a concept name and manually input the relationships among them. The TRANSCOM ontology contains concept names as used in the Air Force Office of Scientific Research (AFOSR) and was kindly made available by Mr. Mark Gorniak.

The NATO government ontologies were also constructed manually by using information in the websites of NATO countries. The NATO website [nat] has a link to the government websites of its member countries. From the government websites of each member country, an ontology was constructed. Each concept name in the ontology is obtained from either the anchor tag of a link in a webpage or the title of the webpage (chosen manually). The relationships between the concept names are manually provided.

The third set of experimental data was obtained from catalogs used in the construction industry. This set of data was provided to us by our colleagues in the civil engineering department [Law]. Each catalog is converted to an ontology by minimally supplying the relationships among objects that are missing.

There was no expert-provided input information supplied to our algorithms since we wanted to compute the accuracy of the automated components of SKAT. I measured the accuracy of the generated matches by comparing the results generated by the automated matcher with those expected by a human expert. I counted any match deleted by the expert as a false positive and lowered the precision figures, and lowered the recall value when a matching rule that SKAT failed to find was added by the expert.

5.6.1 Results of Experiments

The results of the experiments are as follows:

Source	*Number of Nodes*	*Run Time*
NATO (UK vs. Finland)	25 x 30	195 sec.
Transportation	27 x 30	215 sec.
Construction	72 x 178	121 min.

Table 5.1: Sizes of Ontologies and Run Time for Matching

Threshold	*Expert-Indicated Matches*	*Automatically-Generated Matches*	*Correct Matches*	*Recall*	*Precision*
.75	10	8	5	0.5	0.625
.7	10	11	8	0.8	0.73
.65	10	13	8	0.8	0.62
.6	10	16	9	0.9	0.56

Table 5.2: Precision and Recall of Ontology Matching for Transportation Ontologies

Time taken to match the ontologies :

I ran SKAT to to generate matches between each pair of ontologies. The recursion depth was set to 1 and the word similarities using the corpus-based method was pre-computed. The average time taken for 10 runs on each pair of ontologies is reported below:

Apart from the two ontologies that I reported on above, the construction source also comprised two very large ontologies (over 500 nodes in each). The experiment with the large ontologies was aborted because SKAT could not complete matching the large ontologies in more than several hours.

Precision and Recall

Table 5.2 shows the precision and recall values of the matching performed by SKAT while articulating the transportation ontologies. Table 5.3 contains the results for the Construction Ontology, and

Table 5.4 shows the results for the NATO Ontology.

Out of the matches generated, even after turning off the structural matcher, all matches in the Construction and Transportation ontologies were generated, whereas all except 2 correct matches in the NATO ontologies were generated.

Threshold	*Expert Indicated Matches*	*Automatically Generated Matches*	*Correct Matches*	*Recall*	*Precision*
.75	26	18	15	0.58	0.83
.7	26	26	19	0.73	0.73
.65	26	30	20	0.77	0.67
.6	26	40	21	0.81	0.53

Table 5.3: Precision and Recall of Ontology Matching for Construction Ontologies

Threshold	*Expert Indicated Matches*	*Automatically Generated Matches*	*Correct Matches*	*Recall*	*Precision*
.75	12	8	7	0.58	0.88
.7	12	9	8	0.67	0.8
.65	12	11	8	0.67	0.73
.6	12	15	9	0.75	0.6

Table 5.4: Precision and Recall of Ontology Matching for NATO Ontologies

False Positive Results

Since the algorithm is heuristic in nature, we encountered some false positives like *Arrival Time* with *To*. If we lowered the threshold to 0.60 *(Match Airline Destination 0.61)*. Further lowering the threshold to 0.50 introduces more false matches *(Match FlightNumber Sortie 0.52)*, *(Match Equipment Materiel 0.54)*.

I investigated the results to determine the reason why false positives were generated. I discuss the causes of obtaining false positives in decreasing order of their importance below.

1. Poorly chosen concept names: The target ontology had two concepts "From" and "To". Unfortunately, these terms are frequently used in various contexts in documents. Since they were the only words in the concept names, they were not pruned out with other stop-words. More representative concept-names would increase the accuracy of our algorithms. Another way to avoid such false matches because of unrepresentative concept-names is to augment the concept-names by introducing some representation of the "context" in which it appears in the ontology. For example, a simple augmentation can be to tag the name of a concept with the label of its parent node.

2. Low choice of threshold: Since the same threshold does not work for all domains, the threshold may be lower than what should be optimally set for a particular test-case. Such a choice results in a large number of false positives. The threshold should be

increased in such cases.

3. Quality of the corpus: The suitability of the corpus to the matching task influences the matches generated. Most of the documents we had in our corpus were obtained from websites of commercial airlines and news reports or other articles about airlines. In a number of these articles, the term "destination" occurred in contexts that were very similar to the term "airline". Consequently, SKAT determined a match between the terms "Airline" and "Destination". The expert rejected this match because for the purposes of the application, airlines and their destinations are quite distinct concepts.

4. Ambiguity of word senses: We have not encountered this error in our small data sets and thus do not make any effort to disambiguate the senses in which words are used in the ontology concept names. While looking up the thesaurus, we are optimistic. That is, if any one sense of a word matches with any sense of the second word, we declare them as synonyms. The matching senses might not be the exact senses the words are used in the two ontologies and thus, we might be inadvertently introducing errors. However, in our experiments, we did not find such a case and have not remedied the situation in our tools. In the future, in order to improve the precision of the matching process, we can take into account the context in which the concept appears in the ontology and determine the accurate senses accordingly before matching them.

Other Observations

I summarize some of the other the results of the several experiments below:

- A purely structural method that requires exact concept-name match, as has been used in existing tools [MFRW00], fails to generate even 50% of the matches expected by the expert. This result is not surprising since despite having useful information, the structure of the ontologies used hardly encodes sufficient semantics to use them solely for ontology matching.

- Adding linguistic heuristics gave significantly better results, especially, the corpus-based heuristic provided the expert supplied the matcher with a good representative set corpus of documents from the applicable domain.

- However, a multi-strategy approach works best. On the average about 75% of the matches were generated, with less than 5% false positives that the expert indicated

was not correct. The linguistic method generates on the average about 60-70% of the matches (recall with 95% precision). Adding the structural matcher boosts the matches by 5-10%. The human expert provided us with the other 30% of the rules that were not generated automatically.

I believe that the human task is simpler once the automatic articulation rule generation algorithms have matched the ontologies. The human expert now has to use the Viewer and click on an existing articulation rule to remove it or use the text-box on the Viewer to modify the articulation rule. As shown above, since the automated algorithms are reasonably precise, the human has significantly less work than when the human expert has to match the ontologies from scratch. However, a large-scale, formal user-study has not been performed and has been designated as important future work.

Parameters Influencing the Performance of the Algorithms

The performance of the algorithm depends upon several parameters:

- Thesaurus-based Method: A general-purpose thesaurus results in very poor results with substantial numbers of both type 1 and type 2 errors occurring. The precision and recall numbers were lower than 75%. Domain-specific thesauri produce better results but are harder to obtain.

- Corpus-based Method: A corpus-based method produced better results than the thesaurus-based method. In the aircraft example, solely employing the thesaurus-based method produced a 30% recall (at 90% precision). A corpus-based method, where we obtained a corpus by searching the web with a few keywords from the domains, boosted the match to 60%. Combining the two, we obtained a recall of 70%.

- Scalability: Initially, we tried the corpus-based method with a preprocessing step of collecting the corpus and building up the word-context vectors. The linguistic matcher, while matching the ontologies, constructed the word similarities as needed. However, for a test case with 300 nodes in each ontology took an hour to run on a Pentium III machine (650 MHz.) with 256M memory. It becomes clear that for larger ontologies, the algorithm does not scale well if we compute the word similarities while matching the ontologies. For the algorithm to scale, SKAT had to build the corpus and construct the word-context vectors a priori, and pre-compute the similarity of

all pairs of words in the corpus. With this pre-processing, the corpus-based method can be thought of as equivalent to a lookup based method, where the word-similarity matrix is constructed from the words in the corpus. This variation of the corpus-based method scaled well and for our ontologies finished within a few minutes at worst.

- Quality: The quality of the matches were very dependent on the quality of the corpus available. I experimented with corpora of size 50 pages, 100 pages, 200 pages and 1000 pages. Corpora of size 50-100 pages resulted in low recall figures for the matches. A size of 200 web pages often proved adequate to generate a recall of 70%, although in most cases having a corpus of 1000 pages increased the recall less than a few percentage points.

  In an alternate experiment, I compared the results obtained by running the experiments where I had obtained the corpus using the search engine Alta Vista [alt] instead of obtaining them using the search engine Google. The recall figures decreased an average of about 8% points over the 3 sets of experiments I ran but the precision figures were similar. If we accept the premise that Google returns more relevant documents than Alta Vista, then my results can be explained to indicate that the quality of the ontology matching obtained depends on the relevancy of the documents to the domain of the ontologies. Further investigation on the correlation between the relevance of the documents of the corpus with the precision and recall of ontology matches needs to be performed (see discussions on future work in Chapter 6).

Result of Combining Multiple Heuristics

In Figure 2.1 in Chapter 2, we show two ontologies - the United Ontology and the TRANSCOM Ontology and the matches generated. For this experiment, I used a hybrid method that uses WordNet as a thesaurus, and a corpus generated by searching Google. For example, the page "http://www.etrackcargo.com/Help/Agents/Fieldwas part of the corpus. The confidence scores of the matches are as follows when the threshold was set to 0.7:

A word-relater that consults WordNet generates only the first two matches. I ran the word-relater with a maximum recursion depth value of 1. That is, the relater looks into the definition of the two words for similar words but does not proceed any further recursively. The confidence score between *Cargo* and *Payload* was not higher than 0.7 using the corpus-based word-relater. Thus, the corpus-based word-relater would not generate this match.

Table 5.5: The matches between the United and TRANSCOM Ontologies

Term in United.ont	Term in TRANSCOM.ont	Confidence Score
Passenger	Passenger	1.0
Cargo	Payload	1.0
Departure Time	Time (in From subtree)	0.90
Arrival Time	Time (in To subtree)	0.88
Arrival City	Destination	0.79
Name	Location Name	0.75
Departure City	Origin	0.72
Airport	Airforce Base	0.71
Flight	Sortie	0.70

Thus, we see that a hybrid method gives us a better accuracy than any one method alone.

Choice of an Appropriate Threshold

Unfortunately, to the best of my knowledge, in the existing literature, there exists no analytical way of determining the correct threshold for ontology matching. The choice of the best threshold value depends upon the application, the source ontologies being matched, how the similarity scores are calculated, the accessories the matching algorithms use, e.g., the quality of the thesauri or the corpora used, and the tradeoff between precision and recall that is suitable for the application. Therefore, I believe for real-life situations, one should choose a threshold and then tune it by experimenting with progressively increasing and decreasing the threshold and observing its effect on the quality of the results.

In this example, we see that with a threshold value of 0.7, we generate all the desired matches and no false matches - the ideal solution. However, achieving a 100% recall and 100% precision is not achievable in all cases. From this and several other experiments, we see that setting a threshold of 0.7 gives the most number of matches with the a 95the matches are false positives. However, in a significant number of cases the value of the threshold varies depending upon both the corpus supplied and the ontologies being matched. Therefore, we suggest that for an unknown application or an unknown corpus, when running the first time, the matching threshold be set to 0.7. There is no one threshold value that will provide satisfactory for all applications. A threshold of 0.7 will not always produce the best results. However, in scenarios where the desired threshold was not known, in our experiments, a

threshold of 0.7 provided a good starting point. When experts are not satisfied with the quality of the results, they can increase or decrease the threshold to get better matches.

5.7 Summary

In this chapter, I have discussed several heuristic methods to produce simple matching rules between concepts in ontologies that are being aligned. I found that a multi-strategy method based on initial linguistic-similarity followed by structural matching generates matches between ontologies with reliable accuracy. The work of an expert then reduces to validating the suggested rules or supplying new rules instead of having to match entire source ontologies from scratch.

Chapter 6

An Algebra for Ontology Management

In this chapter, I develop an algebraic approach to managing ontologies. Often, creating an ontology from scratch is unnecessary and more expensive than constructing an ontology by composing selected parts of existing ontologies. In this chapter, an algebra is used to declaratively specify how to derive ontologies by composing existing ontologies. A declarative specification of the derivation process makes the composed ontology easy to maintain since such a specification allows easy replay of the composition task when source ontologies are updated and the update must be propagated to derived ontologies [Jan00].

Operations in the algebra are defined on the basis of an articulation-rule-generating function (defined below) that generates rules expressing semantic correspondence between concepts across ontologies. The properties of the operators depend upon those of the articulation-rule-generating function deployed. I have identified the necessary and sufficient conditions that the articulation-rule-generating function must satisfy in order for the algebraic operators to satisfy properties like commutativity, associativity and distributivity. Based on whether these properties are satisfied, a task of composing multiple ontologies can be expressed as multiple equivalent algebraic expressions. Using a cost model, the most optimal algebraic expression can be chosen and executed to derive the composed ontology.

6.1 Composition of Ontologies

Applications that reuse the same ontology can easily communicate with each other because the terms they use come from the same ontology. In this chapter, I describe an algebra that is used to declaratively specify how to derive ontologies by reusing and composing existing ontologies. A declarative specification allows easy replay of the composition task when the source ontologies change and the change needs to be propagated to derived ontologies. Besides, the properties of a declarative specified composition task can be characterized and validated more easily than those of a programmatically specified one.

The ontologies being reused and composed are often independently created and maintained. Thus, they differ in the semantics of the terminologies they use. Before composing two ontologies with different semantics, one needs to resolve their semantic heterogeneity. The composers of the ontologies need to know the correspondence between terms used in the two ontologies before they can perform the composition.

To resolve semantic heterogeneity among ontologies, I assume that there exists functions that take in two ontologies and generate *articulation rules* between them. I refer to functions that generate articulation rules as *articulation-rule-generating functions*. I have discussed in Chapter 5, how I have implemented sc Skat that contains an implementation of such articulation-rule-generating functions.

Articulation rules generated by articulation-rule-generating functions form the basis for the composition of ontologies. Assuming the existence of an articulation-rule-generating function that identifies the semantic matches between terms across ontologies implies that the algebra no longer has the onus of identifying, understanding and reasoning about the semantics of the source ontologies. In this way, I have decoupled the semantic component of ontology composition from the algebraic machinery in order to make it simple to construct, maintain, and reason about each component.

Composers of ontologies can select parts of the ontology that is of interest to the application, for which a new ontology is being created, using an unary operator *Select*. They can compose two ontologies or their selected parts using the binary operators *Union*, *Intersection*, and *Difference*. The binary operators of the algebra depend upon the articulation rules generated by the articulation-rule-generating function. For example, to compute the intersection of two ontologies I need to know which concept in one ontology is similar to which concepts in the other. Not surprisingly, therefore, the properties of the binary operators

are determined by those of the articulation-rule-generating functions.

In this chapter, I identify the necessary and sufficient conditions that must be satisfied by the articulation-rule-generating function in order for the algebraic operations to satisfy properties like commutativity, associativity and distributivity. An expression, corresponding to an ontology-composition task, can be rewritten by rearranging the operators and operands based on the properties satisfied by the operation. A composer can rewrite a given expression to generate all minimal expressions - expressions that do not have redundant sub-expressions - equivalent to the given expression. Using a given cost model, the composer can optimize the expression by choosing the equivalent rewriting of the expression with the least cost. The chosen rewritten expression is then executed to derive the composed ontology.

Even though a lot of work has been done on suggesting heuristics for mapping schemas and aligning ontologies [MGMR02], [DDH01], [MBR01], [NM00], [MFRW00], to the best of my knowledge, no prior work has shown how the properties of heuristic composition algorithms affect the task of composition of ontologies (or the information sources they describe).

Camara, et al., [CFM02], describe a partial algebra for integrating ontologies. However, their work only focuses on obtaining an *Union* of ontologies. Unlike the ONION approach, where the ontologies and articulation rules can contain any relationships, they restrict themselves to only three relationships (equivalent, is-a, and part-of). In addition, they do not consider the effects of the articulation-rule-generating function on the properties of the composition operation.

The rest of the chapter is organized as follows. In the next section, I describe and define some preliminary concepts that I use in the sequel. In Section 4.3, I describe the algebra and its operators — Select, Intersection, Union, and Difference. In Section 4.4, I discuss the properties of the operators and then conclude.

6.2 Preliminaries

For ease of description of the algebra, I will introduce the following terminology: For a Sentence *s=(Subject R Object)*, *Nodes(s)* contains *Subject*(or *Object*) provided *Subject*(correspondingly *Object*) is not a variable (that is, it is a node in some ontology graph).

For an ontology *O1*, *Nodes(O1)* represents the set of nodes consisting of each node that belongs to the ontology graph for *O1*.
For a set of rules R, *Nodes(R)* represents the *Union* of *Nodes(s)* for all s, such that s is a Sentence in any rule $r \in R$.

EXAMPLE 6.2.1 In example in Figure 1.2, for the set of rules

$R = \{(O2.Car\ SubClassOf\ O1.Vehicle), (O2.HouseBoat\ SubClassOf\ O1.Vehicle)\}$,
$Nodes(R) = \{O2.Car, O1.Vehicle, O2.HouseBoat\}$.

□

I use *Edges(E,n)*, where E is a set of edges and n is a node in an ontology graph, to represent all edges in E incident upon or incident from the node n. Formally, $Edges(E, n) = \{s \in E |\ \exists n', l : s = (n, l, n')\ or\ s = (n', l, n)\}$. I use *Edges(E,N)*, where N and E are a set of nodes and edges respectively in an ontology graph, ro represent a set of edges $S \in E$. Both nodes (the node from which an edge is incident from and the node to which it is incident upon) of each edge in the set S must belong to the set of nodes N. Formally, $Edges(E, N) = \{s = (n_1, l, n_2) \in E | n_1, n_2 \in N\}$.

6.2.1 Articulation Rules and Articulation-Rule-Generating Functions

As discussed in the previous chapter, SKAT uses one or more procedures to generate the set of articulation rules between pairs of ontologies. I will refer to procedures that generate articulation rules between ontologies as *articulation-rule-generating function*s (see Chapter 5 for implementation information). An articulation-rule-generating function takes as input two ontologies (domain: the set of all possible ontologies O) and outputs a subset of the set of all possible rules (range: the set of all possible rules R) between them

$$f : O \times O \rightarrow 2^R$$

The articulation generation must be a complete function. That is, given any two ontologies, the function always:

1. terminates and

2. outputs a set of articulation rules.

An articulation rule r belonging to a set of articulation rules generated between two ontologies *O1* and *O2* is such that $\exists O1.n \in Nodes(r)$ and $\exists O2.n' \in Nodes(r)$ for some nodes n and n' in *O1* and *O2* respectively.

EXAMPLE 6.2.2 In the running example in Figure 1.2, I show a few of the articulation rules generated by an articulation-rule-generating function. For lack of space, all articulation rules are not shown in Figure 1.2 , but I show two rules graphically, and two textually at the lower part of the figure. The two graphical rules are shown by dotted arrows spanning different ontologies in contrast to the edges in an ontology indicated by solid arrows. Specifically, I see that $O2.Car$ is related via the relationship *SubClassOf* to $O1.Vehicle$. Similarly $O3.Boat$ is related via the relationship *SubClassOf* to $O1.Vehicle$. I show the rule expressing the first relationship both graphically and textually, and the second only graphically. The second articulation rule indicated textually at the bottom of the figure gives a Horn Clause that indicates the relationship between $O2.Car$ and $O3.LuxuryCar$. Any instance of $O2.Car$ that has a $O2.MSRP$ that has a $O2.Value$ that is greater than $40,000$ is a $O3.LuxuryCar$. Of course, such a rule should also consider the $O2.Denomination$ of the $O2.MSRP$ but for the sake of simplicity I have omitted the denomination from the rule.

□

For the sake of describing the algebra below, I now define the terminology related to articulation-rule-generating functions.

Definition 6.2.1 (**Direct Relation**) Let *f* be an articulation-rule-generating function articulating ontologies $O1$ and $O2$. *f*, is said to *directly relate* two nodes $n_1 \in O1$, and $n_2 \in O2$, iff there exists a relationship R and *f* generates an edge $(n_1\ R\ n_2)$ or an edge $(n_2\ R\ n_1)$. □

EXAMPLE 6.2.3 In the running example, the articulation function that generated the articulation rules directly relates the nodes $O2.Car$ and $O1.Vehicle$, and the nodes $O3.Boat$ and $O1.Vehicle$. □

If an articulation-rule-generating function *f* directly relates two nodes n_1 and n_2, I will denote that hereafter as $(n_1$ *f:drel* $n_2)$.

I now define a property of an articulation-rule-generating function called *transitive connectivity*. I show below in Section 6.3.3 that this property is required for the *Intersection* operator (defined in Section 6.3.2) to be associative (defined in Section 6.3.3).

Definition 6.2.2 (**Transitive Connectivity**) An articulation generator function f is *transitively connective* iff it satisfies the following condition:

$\forall O1, O2, O3 \in O, \forall O1.A \in O1, \forall O2.B \in O2, \forall O3.C \in O3:$
$If \quad \exists r_1 \in f(O1, O2), \exists r_2 \in f(O2, O3) \mid$
$(O1.A, O2.B \in Nodes(r_1)), (O2.B, O3.C \in Nodes(r_2)),$
$Then \quad \exists (r_3, r_4) \in f(O1, O3),$
$\exists O1.D \in O1, \exists O3.E \in O3 \mid$
$(O1.A, O3.E \in Nodes(r_3)), (O1.D, O3.C \in Nodes(r_4))$

□

In other words, if the articulation-rule-generating function discovers that *O1.A* is related to *O2.B* and *O2.B* is related to *O3.C*, then a transitively connective articulation generation function will discover that *O1.A* is related to some node in *O3*, and *O3.C* is related to some node in *O1*.

EXAMPLE 6.2.4 In the running example, if *f* discovers *(O2.Car SubClassOf O1.Vehicle)* and *(O3.Boat SubClassOf O3.Vehicle)*, a transitively connective articulation generator will find some node in *O2* that is related to *O3.Boat* (it might generate the edge *(O2.HouseBoat SubClassOf O3.Boat)*) and some node in *O3* (presumably either *O3.Automobile* or *O3.LuxuryCar*) that is related to *O2.Car*. □

A desirable property of an articulation-rule-generating function is *concept-level monotonicity*.

Definition 6.2.3 (**Concept-level Monotonicity**) An articulation-rule-generating function *f* is said to be *concept-level monotonic* only if it satisfies the following condition:
Let C_i be a concept represented by a node N_i in ontology O_i.
$\forall O_1, O_2, O_3, O_4 \in O, \forall C_i \in O_i, \exists Rel \mid$
$If \quad C_1 \equiv C_3, C_2 \equiv C_4, \ and$
$(N_1 \ Rel \ N_2) \in f(O1, O2)$
$Then \quad (N_3 \ Rel \ N_4) \in f(O3, O4).$

□

Between any pair of concepts, a concept-level monotonic articulation-rule-generating function always generates the same articulation rules irrespective of the ontologies or ontology contexts (neighborhoods) in which the nodes representing the concepts occur. To see how concept-level-monotonic articulation-rule-generating functions work, construct two ontologies *O1* and *O2* each with a single concept *c1* and *c2*. Let *f* be a concept-level monotonic articulation-rule-generating function. Let *f* generate all the set of articulation rules *R*

between the ontologies *O1* and *O2*. While articulating any pair of ontologies containing the concepts *c1* and *c2*, f must generate R for it to be concept-level-monotonic. Thus, whenever the concepts *c1* and *c2* occur across ontologies being matched, f always generates the set of rules R between *c1* and *c2*.

For example, consider an ontology matcher that implements an articulation-rule-generating function based only on the labels of nodes, say, using natural language processing systems that generate similarities of noun phrases. Such a function is concept-level monotonic since it only checks the node-labels, which remain the same in the source ontologies as well as in the intermediate ontologies.

On the other hand, an ontology matcher that articulates the ontologies based on the structure of the ontology graphs is not guaranteed to be node-level consistent. Articulation rules generated by a structure-based ontology matcher depend upon the neighborhood in which a node appears - that is the nodes in close proximity to it. During the process of ontology composition selected portions of ontologies can be grafted into *intermediate ontologies*. When performing operations like select, intersection, or difference, not all nodes in the source ontologies are selected into the intermediate ontologies. Therefore, the context in which nodes appear in source ontologies and that in which nodes appear in intermediate ontologies differ. Thus, when matching two intermediate ontologies, a structure-based matcher may generate different artiulation rules while articulating a pair of nodes in the source ontologies and the same pair in the intermediate ontologies.

EXAMPLE 6.2.5 Concept-level Monotonic Function: I now illustrate a concept-level monotonic function by giving an example. In the running example, let us suppose that an articulation-rule-generating function f generates a rule *(O3.LuxuryCar SubClassOf O2.Car)* based on its price. However, let us assume that in two intermediate ontologies *O4* and *O5* that contain *O3.LuxuryCar* and *O2.Car* respectively, only the concepts useful for describing the appearance and functionality of the cars have been chosen but not their prices. Now if f cannot generate the rule between *O3.LuxuryCar* and *O2.Car* - which might be perfectly reasonable - f is not concept-level monotonic. Of course, f can be made concept-level monotonic if it can follow a link from $O4.LuxuryCar$ to the source ontology *O3*, and from *O5.Car* to the source ontology *O2.Car* and look up the price information and then generate the rule between *O4.LuxuryCar* and *O5.Car* based on their prices. □

Requiring articulation-rule-generating functions to be concept-level monotonic places a strong restriction on such functions, however, as I will show later, the property is necessary for a sequence of two intersection operations to be associative. Associative operations allow flexibility in the order of execution of a composition task and this property is used to optimize the composition process.

In this work, I do not consider articulation rules that might introduce new nodes.

6.3 The Ontology-Composition Algebra

In order to formalize the task of composing ontologies, I propose an algebra for the composition of ontologies. If I use the algebraic framework to compose ontologies systematically, I can enable optimizations depending upon the properties of the operators. In this section, I describe the ontology-composition algebra and in the next section I discuss properties of its operators.

The algebra has one unary operator: *Select*, and three binary operations:
Intersection, *Union*, and *Difference*.

6.3.1 Unary Operator

Select:
Using the *Select* operator, an ontology composer can select portions of an ontology that might be of interest. For example, a car-dealer is interested about cars does not care about houseboats. The car-dealer will select only portions of ontology *O2* that contain terminology about cars and delete the portions that are not related to cars.

The *Select* operation has four forms. One form of the *Select* operation selects all nodes that are reachable from a particular node.

1. **Reachability-based Select**:
   This form of the *Select* operator selects all nodes that can be reached from on a start node and is defined as follows:

   Definition 6.3.1 (**Reachability-based Select**) Given an ontology *O=(G,R)*, *G=(V,E)*, and a node $n \in N$, $Select(O, n) = (G_n, R_n)$ where $G_n = (N_n, E_n)$ is a subgraph of G such that for all nodes $n' \in N_n$, there exists a path from n to n' in G. The set

$E_n = \{e = (n_1\ R\ n_2) \in E | n_1, n_2 \in N_n\}$ is such that each edge in E_n expresses a relationship between nodes n_1 and n_2 where both nodes n_1 and n_2 are in N_n Similarly, $R_n = \{r \in R | Nodes(r) \subseteq N_n\}$ is the set of rules obtained from R containing all rules in O whose concepts are all in N_n. □

The next three forms of the *Select* operation are based on a set of nodes, a set of edges, or a set of rules.

2. **Node-based Select**:
   This form of the *Select* operation is based on a given set of nodes and is defined as follows:

   Definition 6.3.2 (**Node-based Select**) Given an ontology $O = ((N, E), R)$, and a set of nodes V,
   $Select(O, V) = (G, R_v)$ where $G = (V, E_v)$.
   The set $E_v = \{e = (n_1\ R\ n_2) \in E | n_1, n_2 \in V\}$ and the set
   $R_v = \{r \in R | Nodes(r) \subseteq V\}$. □

3. **Edge-based Select**:
   This form of the *Select* operation is based on a given set of edges and is defined as follows:

   Definition 6.3.3 (**Edge-based Select**) Given an ontology $O = ((V, E), R)$, and a set of edges E',
   $Select(O, E') = (G, R_e)$ where $G = (V_e, E')$.
   The set $V_e = Nodes(E')$ and the set
   $R_e = \{r \in R | Nodes(r) \subseteq V\}$. □

4. **Rule-based Select**:

   Similarly, this form of the *Select* operation is based on a given set of rules and is defined as follows:

   Definition 6.3.4 (**Rule-based Select**) Given an ontology $O = ((V, E), R)$, and a set of rules R',

$Select(O, R') = (G, R')$ where $G = (V_r, E_r)$.
The set $V_r = Nodes(R')$ and the set
$E_r = \{e \in E | Nodes(e) \subseteq V\}$. □

EXAMPLE 6.3.1 In the running example, the ontology *O3* contain the edges *(O3.LuxuryCar SubClassOf O3.Automobile)*, and *(O3.LuxuryCar hasA O3.RetailPrice)*. $Select(O3, Automobile)$ would contain all nodes reachable from the node *O3.Automobile*, that is, the nodes (*LuxuryCar, RetailPrice, Denomination, Value*, and *Dollar*), and the edges between them.
Select(O3, {Automobile, LuxuryCar}) would on the other hand only select the nodes $\{O3.LuxuryCar, O3.Automobile\}$ and the edge ($O3.LuxuryCar$ *SubClassOf* $O3.Automobile$). □

Note that a rule r in R that does not involve any node in O has $Nodes(r) = \phi$. The rule r is included in the selected ontology since the empty set is a subset of N. For example, a rule expressing the transitivity of the relationship *SubClassOf*
(X SubClassOf Y), (Y SubClassOf Z) ⇒ (X SubClassOf Z)
contains only variables X, Y, and Z and no concepts from any ontology. The rule r is included in any selected ontology S since potentially, r can be used to reason about the relationships and concepts in S.

There are edges (and rules) that are not present in the source ontology but can be derived using the edges and rules available in the source ontology. There is a case for including such edges and rules in the results of the select operation. For example, suppose the ontology O had edges *(LuxuryCar SubClassOf Car)*, and *(Car SubClassOf Vehicle)* *Select(O, { Vehicle, Car })* contains the last edge. On the other hand *Select(O, Vehicle, LuxuryCar)* includes the nodes *Vehicle*, and *LuxuryCar* but includes no edges since there are no edges between *Vehicle* and *LuxuryCar* in the source ontology. *Select* could be defined to add an edge *(LuxuryCar SubclassOf Vehicle)* if such a relationship could be derived from the ontology O, for example, using a rule that said that the relationship *SubClassOf* is transitive.

Similarly, it is easy to see that *Select* could be defined to introduce additional rules over and above the ones that the current *Select* operation includes in the selected ontology. However, in order to generate these derived edges and rules, the ontology composition engine would need to interpret the rules of the ontology. In order to allow the ONION framework

to be applicable to different ontologies with different interpretation semantics for rules and because potentially, the expert could derive an infinite number of facts in certain scenarios (say with recursive rules), the ontology composition engine does not interpret the rules so as to maintain its simplicity. Moreover, interpreting rules and deducing relationships and additional rules during a select operation would result in the select operation being extremely expensive to compute, especially in the presence of recursive rules where the recursion results in a potentially infinite number of instances (of classes). An alternative future work is to only derive relationships from a limited, well-understood set of special relationships that results in a finite number of edges in the selected ontology graph, e.g., derive *SubClassOf* relationships between all nodes while performing the select operation.

6.3.2 Binary Operators

Each binary operator takes as operands two ontologies that must be articulated, and generates an ontology as a result, using the articulation rules. The articulation rules are generated by an articulation-rule-generating function.

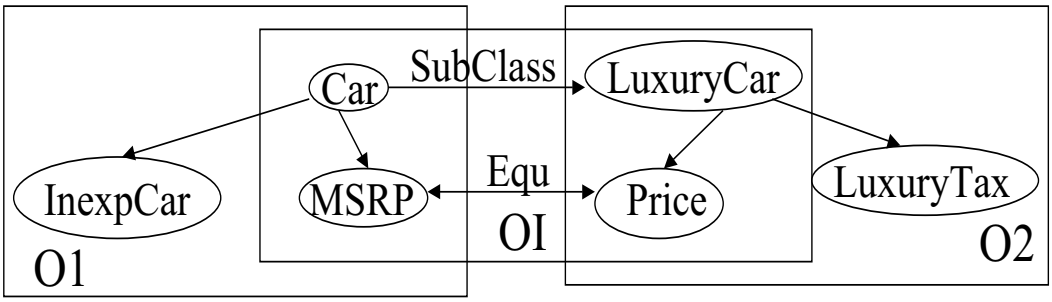
Intersection:

Intersection is the most important and interesting binary operation.

Definition 6.3.5 (**Intersection**) The intersection of two ontologies $O1 = ((N1, E1), R1)$, and $O2 = ((N2, E2), R2)$ with respect to an articulation rule generating function f is:
$OI_{1,2} = O1 \cap_f O2$, where $OI_{1,2} = (NI, EI, RI)$,
$NI = Nodes(f(O1, O2))$,
$EI = Edges(E1, NI \cap N1) + Edges(E2, NI \cap N2) + Edges(f(O1, O2))$, and
$RI = Rules(O1, NI \cap N1) + Rules(O2, NI \cap N2) + f(O1, O2)))$.
□

Nodes in an Intersection:

The nodes in the intersection ontology are those nodes that appear in the articulation rules. An edge in the intersection ontology are the edges among the nodes in the intersection ontology that were either present in the source ontologies or have been output by the articulation-rule-generating function as an articulation rule. The rules in the intersection ontology are the articulation rules that are present in the source ontology that use only concepts that occur in the intersection ontology.



- ArticulationRules = { (O2.LuxuryCar SubClass O1.Car),
  (O1.MSRP Equ O2.Price)}

Figure 6.1: Composition of Ontologies

Consider the example in Figure 6.1. The ontology labeled *OI* is the intersection ontology. Based on the articulation rules, the nodes *O1.MSRP*, *O1.Car*, *O2.Price*, and *O2.LuxuryCar* are in the intersection and the associated edges between these in the source ontologies *O1*, and *O2* respectively and the two edges introduced by the articulation rules *(O2.LuxuryCar SubClassOf O1.Car)*, and *(O1.MSRP Equals O2.Price)* are also in the intersection. The set of rules, *ArticulationRules*, shown below the ontology graphs in the figure are also included in the intersection ontology.

Several object-representation models or document-representation languages [PGMJ95], [xml04]), represent an object as a sub-tree (or sub-graph) rooted at(reachable from) a node corresponding to the object. However, in ONION's knowledge model, each node represents a concept. Therefore, the intersection ontology contains only the nodes that appear in an articulation rule and not the entire sub-graph reachable from those nodes. By including only the nodes and not the sub-graph, ONION keeps the intersection ontologies small and relevant, avoiding the inclusion of possibly irrelevant concepts in the intersection. The label of a node in an intersection ontology contains the URI of the source ontology from which the node was obtained. If the attributes of the object corresponding to the node are required, the onus is on the application to obtain them from the original source.

Note, unlike the set-theoretic Intersection, this definition of Intersection includes nodes from both *V1* and *V2* in the intersection ontology. If there are concepts that are common to both ontologies, both the nodes representing the same concept across the ontologies are

included in the intersection along with an articulation rule that indicates that both the nodes refer to the same concept. For example, let f be an articulation-rule-generating function articulating two ontologies *O1* and *O2*. They have two nodes referring to the same concept. The nodes have the same label, say, *O1.x* and *O2.x*. The function f indicates that they represent the same concept by generating the rule *O1.x Equals O2.x*. In the intersection ontology, I would retain both *O1.x* and *O2.x* and the edge generated by f between them instead of collapsing them into one node.

Union:

The second binary operator that is used to compose ontologies is the *Union*.

Definition 6.3.6 (**Union**) The *Union* OU between two ontologies $O1 = (V1, E1, R1)$ and $O2 = (V2, E2, R2)$ with respect to an articulation-rule-generating function f is expressed as $OU = O1 \cup_f O2 = (VU, EU, RU)$ where
$VU = V1 \cup V2 \cup VI_{1,2}$,
$EU = E1 \cup E2 \cup EI_{1,2}$, and
$RU = R1 \cup R2 \cup RI_{1,2}$, and
where $OI_{1,2} = O1 \cap_f O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies.
□

Nodes in an Union:

In the example in Figure 6.1, the *Union* of the ontologies $O1$ and $O2$ include all the nodes, edges and articulation rules shown in the figure.

The articulation rules indicate relationships between nodes in the two source ontologies and thus introduces new edges (the set $EI_{1,2}$) that were not there in the source ontologies.
S

Though queries are often posed over the *Union* of several information sources, I expect this operation to be rarely applied to entire source ontologies. The *Union* of two source ontologies is seldom materialized, since the objective is not to integrate source ontologies but to create minimal articulations and interoperate based on them. However, I expect that larger applications will often have to combine multiple articulations and here is where the *Union* operation is handy.

Difference

The third binary operator in the algebra is *Difference*.

Definition 6.3.7 (**Difference**) The *Difference* between two ontologies $O1$ and $O2$, using an articulation-rule-generating function *f*, between two ontologies $O1 = ((V1, E1), R1)$ and $O2 = ((V2, E2), R2)$, written as $OD = O1 - O2$, is expressed as $OD = ((VD, ED), RD)$, where $VD = V1 - VI_{1,2}$, $ED = E1 - EI_{1,2}$, and $RD = RI - RI_{1,2}$, and where $OI_{1,2} = O1 \cap_f O2 = (VI_{1,2}, EI_{1,2}, RI_{1,2})$ is the intersection of the two ontologies using *f*. □

That is, the difference ontology includes portions of the first ontology that are not common to the second ontology. The nodes, edges and rules that are not in the intersection ontology but are present in the first ontology comprise the difference.

In the example in Figure 6.1, the difference between the ontologies *O1* and *O2*, is given as: *O1-O2 = ((InexpCar,Null),Null)*. The difference between *O1* and *O2* contains the node *InexpCar* since that is the only node that is present in *O1* that does not have a related node (related by an articulation rule) in *O2*. Since there are no edges between the node *InexpCar* and itself and since the node *InexpCar* does not appear in any rule in the source ontology *O1*, the edges and rules in the difference ontology are *Null*.

The Necessity of the Difference Operator:

One of the objectives of computing the difference ontology is to optimize the maintenance of articulation rules. An articulation might need to be updated when one of the source ontologies that it articulates is changed. A change in the source ontology is to be forwarded to the articulation engine. The articulation engine then checks if the changes are confined to the difference between the ontology and the other ontologies that it has been articulated with. If the change happens to be in the difference, and it does not occur in the intersection, the change is not related to any of the articulation rules that establish semantic bridges between ontologies. Therefore, the articulation rules do not need to be changed. If the changes to a source ontology, instead, are not in the difference, the articulation in which it occurs needs to be updated to reflect the change in the source ontology.

The other objective of having the *Difference* operator is to introduce a form of limited negation for the sake of completeness of the algebra. For example, during the composition process a composer can exclude a set of concepts from the composed ontology using the *Difference* operator.

6.3.3 Properties of the Operators

I defined the operators in the algebra on the basis of the articulation rules produced by the articulation-rule-generating function. Not surprisingly, therefore, most of the properties of the binary operations are based on the properties of the articulation-rule-generating function.

In this sub-section, I examine the following properties of the binary operators:

1. Idempotence
2. Commutativity
3. Associativity
4. Distributivity
5. Existence of Identities
6. Existence of Inverses

Idempotence

Idempotence is a very important property that should hold for operations like *Union* and *Intersection*. Intuitively, the *Union* (and *Intersection*) of an ontology with itself should result in the same ontology. I now examine when the property holds for the algebraic operators.

Idempotence of the Union Operator

Theorem 6.3.1 *The Union operator is not idempotent in general.* □

Proof: The *Union* operation is idempotent if and only if $O1 \cup_f O1 = O1$. Let $OU = O1 \cup_f O1$ and let $AR = f(O1, O2)$. From the definition of the *Union* operation, it follows that $Nodes(OU) = Nodes(AR) \cup Nodes(O1) \cup Nodes(O1)$. For *Union* to be idempotent, I need $Nodes(OU) = Nodes(O1) = Nodes(O1) \cup Nodes(AR)$. For the last equation to hold, $Nodes(AR) \subseteq Nodes(O1)$. That is, the articulation-rule-generating function must not generate any new nodes.

Even if the articulation-rule-generating function does not generate any new nodes, and the nodes in *OU* and *O1* are the same, the former contains more edges and rules by virtue

of the definition of the *Union* operator. The extra edges are self-edges between a node and itself generated by f that are included in the *Union* ontology. Since the edges and rules of OU and $O1$ are not the same, $OU \neq O1$, and the *Union* operator is not idempotent. ∎

Semantic Idempotence

Although the strict definition of idempotence is not satisfied, I introduce the concept of *semantic idempotence*.

Definition 6.3.8 (**Semantic Idempotence**) An binary operator is semantically idempotent if and only if the result of the operation on two copies of the same ontology results in an ontology from which if all self-edges and self-rules are removed, the resulting ontology is the same as the source ontology. □

Recall that self-edges are edges between one node and itself and a self-rule is a rule that has only one node (concept) in its definition.

Consider an articulation-rule-generating function f that does not generate any new nodes $VU = V1$, $EU = E1 + Edges(f(O1, O1))$, $RU = R1 + f(O1, O1)$. Clearly, $Edges(f(O1, O1))$ and $f(O1, O1)$ both only contain self-edges and self-rules. Therefore, the *Union* operator using f is semantically idempotent.

Theorem 6.3.2 *The Union operator is semantically idempotent if and only if the corresponding articulation-rule-generating function generates only self-edges and self-rules.* □

A *duplicate eliminating Union* operator can be defined as an operator that is similar to *Union*, except whenever an articulation rule of the form *(O1.A Equals O2.B)* is generated by its associated articulation-rule-generating function, instead of both *O1.A* and *O2.B* in the union, it includes only one of them. Under such a definition, the *duplicate eliminating Union* operator does not have two copies of the same node provided f only generates rules equating a node with itself. In such a case the *duplicate-eliminating Union* is semantically idempotent.

Idempotence of the Intersection Operator

Now, consider the intersection operator. Let $OI = O1 \cap O1 = ((VI, EI), RI)$. For, intersection to be idempotent, I need $VI = V1$. That is, the articulation-rule-generating function generates articulation rules for all nodes in the source ontology. Similar to *Union*, it is easy to see that the intersection operator is semantically idempotent, if and only if,

the articulation-rule-generating function that generated the articulation rules between an ontology and itself generates only self-edges and self-rules for each node in the ontology. This requirement is obvious since the same ontology is matched to itself and one would expect a set of rules to be generated that matches each node to itself.

Theorem 6.3.3 *The intersection operator is semantically idempotent if and only if the corresponding articulation-rule-generating function generates only self-edges and self-rules and generates at least one self-edge or self-rule for each node in a source ontology.* □

Idempotence of the Difference Operator

The *Difference* operator should not be idempotent, nor should it be semantically idempotent. However, a difference operation using a "well-behaved" articulation-rule-generating function - one for which *Intersection* is semantically idempotent - should return the null ontology.

Definition 6.3.9 (**Null Ontology**) The *NullOntology* is an ontology whose sets of nodes, edges and rules are all null sets. □

Theorem 6.3.4 *The difference operator returns a Null Ontology, if and only if the corresponding articulation-rule-generating function generates only self-edges and self-rules and generates at least one self-edge and self-rule for each node in a source ontology.* □

Semantic Idempotence and Soundness of Articulation-Generation Functions

The semantic idempotence property serves as a sanity check to make sure that the articulation-rule-generating functions that are employed are semantically meaningful and intuitively correct. In order to guarantee that an articulation-rule-generating function is idempotent, ONION would have to check over the set of all (potentially infinite) ontologies. In practice, one can simply check to see that for the purposes of an application, the articulation-rule-generating function is idempotent for all ontologies used by the application. That is, although the functions are not proven to be idempotent, ONION can check to see that none of the ontologies that is useful for the application can be used to prove that the function is not idempotent. This checking has not been implemented currently and is left as important future work.

Definition 6.3.10 (**Unsound Articulation-Generation Functions**) An articulation-rule-generating function f is termed *unsound* if either the *Union* or the *Intersection* operator defined using f is not semantically idempotent. □

For the purposes of the compositions, OntoComp does not use any unsound articulation-rule-generating function.

Commutativity

Commutativity of operators is another important and desirable property. If an operator is commutative, the operands can be swapped. For certain cost models, the order of the operands influence the cost of the operation. An optimizer that is generating a plan for the composition of the ontologies can optimize the composition by swapping the operands if the operator is commutative.

By the symmetry of the definitions of *Intersection* and *Union*, it is easy to see that these operations are commutative if and only if the articulation-rule-generating function they are based on are commutative.

Theorem 6.3.5 *The Intersection (and Union) operator is commutative if and only if the associated articulation-rule-generating function is commutative.* □

Automatically proving an articulation-rule-generating function to be commutative is not always feasible or is prohibitively expensive. In the absence of an automated proof of commutativity, an ontology composition system requires input from the provider of the articulation-rule-generating function, or an human expert familiar with the function must indicate whether it is commutative. In the absence of such manual input, the query system conservatively assumes that the operation is not commutative if it cannot prove otherwise and eschews any optimization that results from assuming commutativity.

Associativity

Associativity enables optimization of a composition task by allowing the reordering of the executed operations. A composition engine can decide on the optimal ordering of the operations based on a cost model. Typical cost models potentially use available statistics about the size of the ontologies and their intersections. I now show an example that illustrates the optimization that can be enabled using the associativity of the *Intersection* operator.

EXAMPLE 6.3.2 For example, assume OntoComp needs to compute $Plan\ X : (O1 \cap_f O2) \cap_f O3$ where *O1* and *O2* are very large ontologies and *O3* is very small. The resulting

intersection will obviously be small. If the *Intersection* operator with the articulation-rule-generating function f is associative, I could rewrite the above composition as $Plan\ Y$: $O1 \cap_f (O2 \cap_f O3)$. If a query planner chose to execute $Plan\ X$, the query processor would spend a lot of time articulating the two large ontologies $O1$ and $O2$, and then articulate it with *O3*, the small ontology. On the other hand, if the *Intersection* operator was associative, an intelligent query planner that knows the sizes of the ontologies would opt for $Plan\ Y$. The plan, $Plan\ Y$ involves articulating a large ontology, $O2$, with the small ontology $O3$, and then articulating the resulting small ontology with the other large ontology $O1$. The presence of a small ontology in both articulations would make $Plan\ Y$ faster and require less storage for the intermediate steps. □

Intersection

I now give the necessary and sufficient conditions for the *Intersection* to be associative, i.e., for $(O1 \cap_{ARules} O2) \cap_{ARules} O3 = O1 \cap_{ARules} (O2 \cap_{ARules} O3)$ to hold.

Theorem 6.3.6 *The Intersection operator using an articulation-rule-generating function f is associative iff f is concept-level monotonic and transitively connective.* □

Concept-level monotonicity of an articulation-rule-generating function indicates that the function either always matches two concepts or never matches them, irrespective of the ontologies they are in. A stricter form of a transitively connective articulation-rule-generating function is one that is transitive. Although a transitive articulation-rule-generating function is sufficient, transitivity is not necessary for *Intersection* to be associative. I now show two examples that illustrate that transitive connectivity of the articulation-rule-generating function in Theorem 6.3.6, is necessary.

EXAMPLE 6.3.3 Consider the example where $O1 = (car, null, null)$, $O2 = (truck, null, null)$ and $O3 = (vehicle, null, null)$. That is, the three ontologies simply have one node each and no edges or rules. Let f be an articulation-rule-generating function that is not transitively connective, and the articulation rules between $O1$ and $O2$ be as follows:
$f(O1, O2) = null$. Articulating this null ontology with $O3$ returns null. Therefore, $OI_1 = ((O1 \cap_f O2) \cap_f O3) = null$. On the other hand, let the articulation rules $f(O2, O3) = ((O2.Truck\ SubclassOf\ O3.Vehicle))$. Therefore,

$(O2 \cap_f O3) = OI = ((O2.Truck, O3.Vehicle), ((O2.truck\ SubclassOf\ O3.Vehicle)), null)$.
Let $f(OI, O1) = (O1.Car\ SubclassOf\ O3.Vehicle)$.
Then, $(O1 \cap_f (O2 \cap_f O3)) = (OI \cap_f O1) =$
$OI_2 = ((O1.Car, O3.Vehicle), ((O1.Car\ SubclassOf\ O3.Vehicle)), null)$.
Since $OI_1 \neq OI_2$, the *Intersection* is not associative but depends upon the order of the operand ontologies. Thus, if the articulating generating function is not transitively connective, *Intersection* is not associative □

In a number of practical situations, the *Intersection* operator is not provably associative. Such a situation arises when the articulation-rule-generating function is programmatically specified and it is prohibitively expensive to prove its properties. In such a scenarion, reordering the order of execution of two intersections changes the resulting ontology since *Intersection* is no longer associative. However, the person wishing to compose the two ontologies does not necessarily have a preference in the order in which the ontologies are composed since the results will be semantically equivalent though not exactly the same. In such a scenario, the articulation-rule-generating function is not provably strictly transitively connective. That is, the intersections are not entirely independent of the order of the operands. Since the composer does not have a preference of the order, he might instruct the system to assume associativity. If the composition system is able to assume associativity, it is free to reorder the operands and enable optimization.

Union

The *Union* operation is associative if and only if

$$\forall O1, \forall O2, \forall O3 | (O1 \cup_f (O2 \cup_f O3)) = ((O1 \cup_f O2) \cup_f O3) \tag{6.1}$$

where f is the articulation-rule-generating function and *O1*, *O2*, and $O3$ are ontologies. I now identify the necessary and sufficient conditions under which *Union* is associative.

Theorem 6.3.7 *The Union operation is associative if and only if the associated articulation-rule-generating function f is concept-level monotonic.* □

Difference

I now show that the *Difference* operator is not associative (nor should it be) under any conditions.

Theorem 6.3.8 *Difference is not associative under any conditions.* □

Proof: I need to show that $O1 - (O2 - O3)$ is not equal to $(O1 - O2) - O3$. Let $O1, O2, O3$ contain only one node each: n_1, n_2, and n_3 (respectively). Let f generate rules $Rule(n_1, n_2), Rule(n_1, n_3), Rule(n_2, n_3)$ while articulating $O1$ and $O2$, $O1$ and $O2$, and $O2$ and $O3$ respectively. Using the definition of difference, $(O2 - O3)$ is a null ontology since the only nodes in $O2$ and $O3$, namely n_2 and n_3 are related by f. Thus $O1 - (O2 - O3) = O1$. Again $(O1 - O2)$ is a null ontology since the only nodes in $O1$ and $O2$, namely n_1 and n_2 are related by f. Therefore, $(O1 - O2) - O3$ is a null ontology. Thus, difference is not associative. ∎

Associativity and Articulation-Generation Functions

It follows from the necessary and sufficient conditions that the *Intersection* operation is not associative when the articulating generating function is not concept-level monotonic. When a function generates articulation rules based on the context of the nodes, the function is not guaranteed to be concept-level monotonic.

Several data integration systems employ structural matchers [MBR01], [NM00], [MWJ99], [MFRW00]. These matchers depend upon the structure of the two source ontologies, (that is they depend upon the presence of certain edges between nodes in the source ontologies) to generate articulation rules between them. If structural matchers are used to generate articulation rules, the corresponding articulation-rule-generating function is not concept-level monotonic. Therefore, the operators like *Intersection*, using structural matchers as the articulation-rule-generating function, are not guaranteed to be associative.

The reason why *Intersection* is not associative is explained below. Inherently, *Intersection* is a "lossy" operation. That is, all edges in the neighborhood of a node may be included in an intersection ontology. Thus, when the intersection ontology is composed with another ontology, the articulation-rule-generating function finds the node in a different context (that is, in a neighborhood that has a different set of edges than in the source ontology). Since the matching and generation of articulation rules are based on the context (e.g., in structural matchers), the articulation rules generated between a pair of nodes in derived ontologies (like intersection ontologies) will not be the same as those generated between the nodes in the source ontologies.

Therefore, if the articulation-rule-generating function depends upon structural matchers, optimizations enabled by associativity must be disabled unless the structural matcher is guaranteed to be concept-level monotonic (and transitively connective). Articulation generating functions that are based on linguistic matchers just look at concept-names and thus typically satisfy concept-level monotonicity conditions. Therefore, with respect to optimizing compositions of ontologies, linguistic matchers are preferred to structural matchers. However, if the structural matchers generate significant semantic matches that are necessary for the application, the composer should use the semantic matcher and not strive to achieve concept-level monotonic articulation-rule-generating functions or to optimize the composition process.

Distributivity

Recall that two operators σ and π are distributive with respect to a set of operands O, if and only if

$$\forall O_1, O_2, O_3 \in O : ((O_1 \ \sigma \ O_2) \ \pi \ O_3) = ((O_1 \ \pi \ O_3) \ \sigma \ (O_2 \ \pi \ O_3)) \tag{6.2}$$

Similar to associativity, I show that the *Union* and the *Intersection* operations, using a concept-level monotonic articulation-rule-generating function, are distributive. The distributivity property, again, enables the OntoComp to rewrite and reorder a composition task to optimize its performance.

Theorem 6.3.9 *:* $(O1 \cup_f O2) \cap_f O3) = (O1 \cap_f O3) \cup (O2 \cap_f O3)$ *iff the articulation-rule-generating function f is concept-level monotonic.* □

6.3.4 Identities and Inverses

In this subsection, I identify identity elements and define the notion of complements for the Ontology Composition Algebra. I want to define the notion of identity elements in order to define the complement or inverse operation.

Identity Elements

I introduce two identity elements: ϕ and U.

The first identity element ϕ is defined to be a *Null Ontology*. That is, $\Phi = ((\phi, \phi), \phi)$, where ϕ is the null set. The *Null Ontology* contains an empty set of nodes, an empty set

of edges and an empty set of rules. It is easy to verify that: $O1 \cup_f \Phi = O1$, $O1 \cap_f \Phi = \Phi$, $O1 - \Phi = O1$, and $\Phi - O1 = \Phi$ for any reasonable articulation-rule-generating function f that does not generate any articulation rules when one of the operators is the *Null Ontology*.

The second identity element is U - the *Universal Ontology*. The *Universal Ontology* is defined as the articulated union of all ontologies in the domain of discourse for the application where the algebra is being used. I now derive the necessary and sufficient conditions on the articulation-rule-generating function for the Ontology Composition Algebra to satisfy the properties given below. These conditions are similar to the ones for semantic idempotence.

1. $A \cup_f U$ is semantically equivalent to U if I ignore self-edges (rules of the form *(O1.n Equals O1.n)*) if and only if the articulation-rule-generating function generates only self-edges.

2. $A - U = \Phi$ if and only if the articulation-rule-generating function associated with the difference operator generates at least one self-edge for each node in A.

However, $A \cap_f U = A$ does not hold in general since the intersection, by definition, may include nodes in U that are not in A. $A \cap_f U$ contains A and the nodes in U that are related to nodes in A and their associated edges and rules.

Inverse Elements

Unfortunately, unlike a Boolean Algebra [Boo54], the Ontology-Composition Algebra does not have one unique inverse or complement element. I define two inverse elements, one based on the *Union* operator and the other on *Intersection*.

The *Union-complement* of an ontology *O1*, denoted by $O1'_U$, is the ontology obtained by taking the union of all ontologies in the domain of discourse except *O1*. It is easy to verify that $O1'_U \cup_f O1 = U$ if and only if f is concept-level monotonic. The ontology U is obtained by taking the union of all ontologies and $O1' \cup_f O1$ is also obtained in the same manner. If f is concept-level monotonic, it is consistent. That is, for any pair of concepts occurring in any of the source ontologies, f (i) either generates a rule in both cases (corresponding to the computation of the two unions) between two concepts, (ii) or it does not generate a rule in either case. As a result the two unions produce the same ontology.

The *Intersection-complement* of an ontology *O1*, denoted by $O1'_I$, is the ontology defined as $U - O1$. Again, from the defnition, it follows that $O1'_I \cap_f O1 = \Phi$ if and only if f is concept-level monotonic. Again, if f generates (or does not generate) the same set of rules between

pairs of concepts while constructing U and articulating U with $O1$ and while articulating $O1$ with $O1'_I$ consistently the equation above holds.

6.4 Summary

In this chapter, I highlighted an algebra for the composition of ontologies. The algebraic operators depend upon the properties of an articulation-rule-generating function that generates the rules on which the algebra is based. The primary advantage of the algebra enables efficient and systematic automated maintenance of the composed ontologies when source ontologies are updated. A secondary advantage is that optimizations can be enabled if the operations are commutative, associative and distributive. I identified the necessary and sufficient conditions that articulation-rule-generating functions must satisfy for the operators to be idempotent, commutative, associative, and distributive. As an important corollary, I find that linguistic matchers are better behaved than structural matchers in the sense that they allow more scope for optimizing the process of composing ontologies. The algebra provides a declarative framework for specifying how ontologies are composed from source ontologies. Such a specification can be readily replayed automatically when the ontologies change and the composition has to be reconstituted.

Chapter 7

Related Work

In this chapter, I discuss related work and compare my work with them.

7.1 Information Extraction

Before the processing of information from an information source can be automated, the information should be present in a form that a machine can interpret. Since ONION can only handle ontologies formatted using its common ontology format, described in Chapter 4, the source ontologies must be formatted to the common format. In order to do so, the ontologies must be "wrapped". In this section, I examine the various ontology formats in vogue today and briefly discuss the strides that have been made in constructing wrappers for them.

7.1.1 Wrappers

Researchers have made some progress and extracted such information using semi-automated wrappers [KMA+98]. However extracting information using wrappers is expensive.

Wrappers are programs that, given an unstructured webpage, extract information and structure it according to some pre-defined needs of the application invoking it. Writing wrappers requires a thorough knowledge of the information source that it will wrap. Wrapped information will be processed and composed using a machine. The extracted information is converted to a format, language, and vocabulary that the composing process understands. When the source changes or when the requirements of the target change, the wrappers have to be modified. Today, websites not only change their content but also their

layout quite frequently. Thus, maintenance of wrappers can be very costly for these fast changing websites.

Most wrappers are semi-automatic [KWD97, Kus00, CMM01, Sod99, MMK01, AK97, Hsu98] . That is, it involves a significant amount of human intervention to wrap an information source. To enable large-scale automated access to information sources, we need to interpret the semantics of the website using a machine.

Another approach to making the data on websites and other information sources machine processible is to use meta-data to describe the data that appears on the websites. Meta-data is additional data that describes the data available from the information sources. It aids a machine to understand data from websites and other sources. In the rest of the section, we examine the different choices for markup languages.

7.2 Markup Languages

In this section, I discuss the related work on data and object models and markup languages that are used to represent data in information sources primarily over the Internet.

7.2.1 OEM

The Object Exchange Model (OEM) [PGMJ95], has been used to represent meta-data. It contains atomic and complex objects. Each object has a label and a value. For complex objects, the value can be a sequence of multiple OEM objects. The value of an object can be inlined or it can be a reference. These models are tree-structured. For nodes with multiple ancestors, the models do not require the entire subtree rooted at the node to be duplicated under both its parents. Instead, the node (say n) exists as the child of one of its ancestors (the value field of one ancestor, say p) and a unique identifier is associated with that node (say, N). The value field of the other ancestor object (say p') contains a label that expresses the relationship (say l) of the ancestor object (p') and the node (n) and a value that is a reference($\&N$) to the node (n) using its unique identifier(N). The use of this reference to the node (n) helps preserve the tree structure of the models, since such models were designed to represent documents that require serialized entities.

Unlike OEM, ONION a concept is represented using only a node and not by a subtree. Thus, to maintain the minimality of the intersection of two source ontologies, the *Intersection* operation only contains an individual node and not the entire subtree rooted

at the node.

7.2.2 SGML and HTML

The problem of meta-data representation and sharing has been addressed in the research on information retrieval. For websites, a common place to make metadata available is to provide it along with the markup of a webpage. Documents have long been marked up using the Standard Generalized Markup Language, (SGML) [sgm]. The de facto standard for marking up web documents is the Hyper Text Markup Language (HTML) [htm]. The primary focus of HTML is on presenting information so that it can be browsed by a human being and not on enabling machine-processing of the information. The simplicity of HTML has ensured that it has been very well adopted. Jeffrey Veen [vee] describes the capabilities of HTML well by saying that it is "a very simple way to describe a limited set of information for transmission and display on the web." For example, the following creates a simple webpage displaying the name of the author:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<HTML>
 <CENTER> Prasenjit Mitra </CENTER>
</HTML>
```

Obviously, for richer semantic interpretation of data, a richer language is needed to express markup. Undoubtably, SGML fits such a description for a language, but does not provide readily sharable specification, since it is a meta-language itself. Different publishers have defined widely varying specifications with SGML. HTML started out as one such specification. Therefore, there is a need for languages beyond SGML and HTML, in order to represent information whose semantics is uniformly specified.

7.2.3 XML: Extended Markup Language

The Extended Markup Language (XML) provides a good compromise between retaining the simplicity of HTML and providing the flexibility (of expressing metadata) in SGML. XML, with its tagged elements, is a step forward towards providing metadata that describes the semantics of the data in a document.

Our very simple html webpage would now be written in XML as follows:

```
<?xml version="1.0" encoding="ISO-8859-1" ? >
<name> Prasenjit Mitra </name>
```

The XML document is interpreted by a browser for page-layout representation. Note that we have used a tag that is more meaningful than that in the HTML version and if the semantics of this tag is specified and accessible by a machine, the document becomes processible by a machine.

7.2.4 The Semantic Web: A Layered Approach

XML, on its own, however, can not express the semantics of the published information. Machines cannot interpret tags to derive sufficient semantics required for applications. To the computer, the XML tag <book> has as much meaning as the HTML tag <i>. We need precise definitions of the tags and their semantics, documentation of what object a tag represents, a description of relationships among the objects associated with the tags, and at least a precise specification of the constraints on these objects. Therefore, very soon, we realize the need for expressing relationships among objects in a document and providing richer descriptions.

A language that lets us capture the semantics of a tag would require features that would make the language unduly complicated and unwieldy. Interpretation of the language and deduction of the full semantics would be very slow. I believe that the most elegant and practical solution is to take a layered approach. That is, the solution is to employ a framework that has multiple layers - each layer is entrusted with capturing a clearly defined subset of the semantics of the world of discourse. For example, instead of designing a more complicated markup language that would provide a construct to express relationships among elements described in the document, it is easier to design a layered metadata description framework. In this framework, XML provides the basic metadata in the form of tagged elements in a document. Over and above this layer that tags objects, we seek to build other layers that expresses relationships amongst them, and provides other rules and constraints that can be used to derive other meaningful information useful to the application at hand.

7.2.5 RDF: Resource Description Framework

In order to express relationships among XML documents and elements present in XML documents, we need a framework to describe such objects. The Resource Description Framework

(RDF) was designed with such an objective in mind.

Once again, we could express meta-data about our simple web-page using RDF in XML serialization syntax. In the following document, we express the fact that the webpage at "http://www-db.stanford.edu/~ prasen9" has a "Creator" named "Prasenjit Mitra". The term "Creator" has a precisely defined semantics that is available at "http://mydescription.org/schema/".

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://mydescription.org/schema/">
  <rdf:Description about="http://www-db.stanford.edu/~prasen9">
    <s:Creator>Prasenjit Mitra</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

RDF as its name suggests provides a framework to describe resources. The RDF Model and Syntax Specification [rdf99] indicates that "All things being described by RDF expressions are called resources. A resource may be an entire Web page, ... A resource may be a part of a Web page, e.g., a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book."

Resources are identified by an Uniform Resource Identifiers (URIs also known as URLs) [URI]. URIs help refer to a resource declaratively instead of using a description of how to obtain the resource (specifying the machine, access protocol etc.) to identify objects.

7.2.6 RDF Schema

RDF allows us to model objects and their relationships using a simple model that can be expressed as a graph. The relationships among the resources are expressed in terms of properties and values. However, only the objects and their relationships might not be enough for the purposes of an application. Not only do we need to define properties of resources, but also describe classes of resources.

Consider the example in Pease[pea]. He shows how we can express that the property *motherOf* is a *subProperty* of the property *parentOf* using the DAML statement: *(motherOf*

subProperty parentOf) Once we have established such a rule, we can use it to derive the statement *(Mary parentOf Bill)* from the statement *(Mary motherOf Bill)*

RDF Schema [rdf00] provides a language to define classes of resources and to specify restrictions on combinations of classes and properties.

7.2.7 OWL: Web Ontology Language

It is easy to see that even RDF Schema provides very basic functionality and is not sufficient to express several constructs. For example, RDF-Schema does not have any construct to express that two classes are disjoint. A language that can be used to express even richer relationships among the data is required. The DARPA Agent Markup Language (DAML) was designed to be a layer on top of RDF in which one can express relationships between relationships and specify constraints on such relationships.

DAML also provides the ability to express constraints including cardinality constraints and qualified constraints. The former can be used to express minimum and maximum cardinality constraints on an object. For example, we can specify that *Mary* has between 1 and 3 children. The latter constraints can be used to specify restrictions on the type of objects. For example, we can state that "at most 2 of the children of Bill are of type Doctor" [dama]. DAML was the precursor to DAML+OIL, which finally got supplanted by OWL, the Web Ontology Language, as a standard. By and large, OWL and DAML+OIL are very similar except a few changes and improvements made in OWL.

In this section, I outlined the various formats that can be used to mark up documents and express meta-data about information sources. I have argued that HTML and XML are too primitive to satisfactorily model ontologies in, and even though DAML+OIL is more expressive than RDF, it is less tractable than RDF. RDF, thus, provides the most appropriate middle-ground for the ONION system.

In the next section, we examine the work related to the articulation-rule-generating algorithms implemented in SKAT.

7.3 Ontology Matching and Articulation

As in [CHG], [GBMS99], and [Kar96], I assume that information sources are independently created and maintained, remain autonomous and do not conform to a standard ontology.

7.3.1 Karp's Interoperation Framework

Karp [Kar96] proposes a strategy for database interoperation. His strategy is based on posing queries to multiple network-accessible databases. The system determines which databases contain relevant information to answer the query and consists of "translators that can interconvert among different schema languages". While building ONION, I extended Karp's approach to apply not only to databases, but also to knowledge bases and information sources.

In Karp's system, each database comes with a schema, which is saved in a Knowledge Base of Databases. Correspondingly, ONION requires that associated with each information source is an ontology, but it does not require all ontologies to be saved in a central repository. Karp proposes that the knowledge base contain descriptions of relationships among the conceptualizations used in different databases. These descriptions are similar to articulation rules used in ONION. I assume that the schema and ontologies associated with information sources have been pre-processed in a lower layer of the system into one common format such that they are no longer described in different languages. That is, the interoperation enabled by ONION is on the semantic level and not at the language level.

7.3.2 Ontology Alignment

The problem of ontology matching and alignment has been studied for some time. Tools like OntoMorph [Cha00], PROMPT [NM00], and Chimaera [MFRW00] help significantly automate the process. However, these tools do not contain a built-in linguistic matching component that deduce the similarity of concepts without any human input. Due to the extensive linguistic matching component, SKAT provides a greater degree of automation while keeping the option of a human expert to ratify the suggested articulation.

Noy, et al., have proposed PROMPT [NM00], an algorithm and tool available for automated ontology merging and alignment. Like ONION, PROMPT is also a semi-automatic tool. PROMPT does not attempt to determine linguistic similarities between concept names, but the tool can use matches generated by a linguistic matcher to generate further matches. Thus one can use ONION's ontology matching algorithm to generate matches between concepts across ontologies and then use these matches as a starting point and generate further matches using PROMPT.

The structural matching algorithm used by SKAT has similarities with

Anchor-PROMPT[NM01], an algorithm that was independently developed and used in Prote'ge'[GMF+02]. Anchor-PROMPT uses two anchors, concepts in an ontology with a path between them. Those two concepts have already been matched to corresponding concepts in another ontology, that too are connected by one or more paths. Anchor-PROMPT considers all possible pairings of paths where the two paths in a pair come from different ontologies but are between the chosen anchors in the two ontologies. For each path-pair, Anchor-PROMPT increases the similarity score between corresponding concepts in the path. That is, the similarity score between the n^{th} concept in one path with that of the n^{th} concept in the other path is increased, where n lies between 2 and the minimum length of the two paths. Consequently, when two concepts that appear at the same distance from their corresponding starting anchors in multiple paths, their similarity score gets consistently increased and has a greater chance of exceeding the threshold similarity score. Our algorithm can be thought of as a special case of Anchor-PROMPT since it considers only small paths of length 3 - the matching candidates, their parents, and children. Since the ontologies I used were of varying granularity, I observed that matching based on distance from anchors produced less than 50% accuracy when longer paths were considered. Therefore, SKAT considers only small paths while matching based on structural similarity.

McGuiness, et al., have proposed Chimaera [MFRW00], another tool that has been used to match ontologies. The focus of the Chimaera is on providing end-users with a graphical user interface using which they can align ontologies and merge them. It uses rudimentary name similarity techniques to suggest matches and looks at the limited structure of the ontologies. However their work too does not perform deep linguistic matching of the concept names and would be unable to decipher the linguistic matcher generated by ONION. For example the similarity between *flight* and *sortie*, as required by the transportation application, would not be deciphered by Chimaera.

Gal, et al., [GMJ] have improved web search by matching ontologies. They have suggested a set of algorithms that automatically match terminologies in two Web resources. They too look at similar words and consult a thesaurus. However, their algorithm aims at improving web search on the fly, and not at generating articulation rules. Besides, ONION's linguistic matcher is substantially more sophisticated than their ontology matching algorithm since it uses a novel recursive algorithm to derive matches depending upon the definition of words. ONION's lexical matcher also uses a corpus-based algorithm and couples it with the thesaurus-based algorithm to generate more and precise quality matches

than the matcher proposed by Gal, et al.

Doan, et al., [DMDH02], [Doa02], [DDH01] have proposed the GLUE system for ontology matching and the LSD system for schema matching. Like our approach, GLUE also uses multiple strategies and combines them to generate matches between ontologies. GLUE deploys a learning-based strategy to match ontologies. It leverages different types of information like data instances, taxonomic structures, like in our approach, to generate matches.

To the best of our knowledge, GLUE does not look into linguistic similarities between concept names. Our experiments show that at least in several domains, the concept names provide sufficient semantic clues about the nature of the concepts they represent. Thus, concept names are an important source of information that can be used for matching ontologies and should not be ignored. Second, a learning-based approach has increased starting costs since for the success of the approach a well-constructed training set is very important. In contrast, even though the ONION system can utilize seed rules, our approach can be minimally bootstrapped without providing any seed rules and can still generate matches between ontologies. ONION learns from the feedback provided by the expert.

Partitioning of Knowledge-Bases and the Windowing Algorithm

I have used a very simple ad-hoc windowing algorithm. MacCartney, et al., [MMAU03] have proposed an elaborate algorithm for automatic partitioning of large knowledge bases into smaller units in the context of theorem proving. They identify and exploit the implicit structure of the knowledge in the knowedge base to find effective partitions. They have studied the effects of the partitioning and proposed message passing algorithms to guarantee the soundness and completeness of their reasoning. Extending the windowing algorithm in this work using principled partitioning of the ontologies so that we can achieve maximal efficiency without sacrificing the accuracy of the ontology matching is left for future work.

Matching Relationships across Ontologies

SKAT does not match relationships across ontologies. However, just like a concept is represented using different terms in different ontologies, a relationship may be represented using different terms in different ontologies. For example, the Foundational Model of Anatomy (FMA) [RSB98] and GALEN's [RGG+94] common reference model have the relationships

PART_OF and *isBranchOf* that have similar semantics. SKAT bypasses the problem by expecting these relationships to be either translated into equivalent relationships in the ONION Ontology Format or reified into concepts. If the relationships are reified into concepts, the concept-matching algorithm matches them.

Zhang and Bodenreider [ZB04] have developed an algorithm to match associated relationships across ontologies by analysing paths between equivalent concepts across the ontologies. The algorithm expects concept matches have been already determined and uses matching concepts as anchors. Using two concepts as anchors, the algorithm finds all paths between these concepts and their matching concepts in both ontologies. They only consider paths of length six or less. In case multiple patterns are found to match one relationship, the pattern that occurs most often is chosen as the match.

7.3.3 Schema Matching and its Similarities with Ontology Matching

The problem of ontology matching has similarities with the problem of schema matching in databases. There exists several schema matching tools, e.g., [YMHF01],[MBR01], [DDH01], [MGMR02], [PGMU96], [CHG] etc. Most of these approaches are not adequate when the primary differences among sources are due to differences in terminology in sources with little structural similarity or when instance data is not available. These methods provide poor results due to the absence of good structural similarity or the absence of instance data in our applications.

The Clio project, designed at IBM Almaden Research Center, also attempts to solve the schema matching problem.Clio[YMHF01] bases its matches upon instances of classes and their attributes and cannot match the schemas in the absence of instance data. Our articulation generator uses multiple strategies and makes use of instance data. However, since we augment them with linguistic and structural analysis of the data, we can articulate ontologies that do not have instances available with them.

Cupid[MBR01] and the similarity-flooding algorithm[MGMR02] bank on the schema structures for generating its articulations and again does not make use of the linguistic information available in the information sources.

LSD[DDH01] uses a learning-based approach where the tool requires some manual work in identifying suitable test cases that are representative of the schema that will be matched in future. ONION also employs a semi-automatic articulation generator but the manual effort is required *after* the matcher has already generated a suggestion. For large ontologies,

constructing suitable test cases and identifying the matches and mismatches among the ontologies for LSD takes a large amount of manual effort. Whereas in ONION, the expert needs only to verify the matches generated and provide the rules only for cases where the articulation generator erred. The articulation generator for ONION logs the responses of the expert and uses this information to generate better articulations in future. Using a systematic approach to construct articulations and logging the input provided by an expert allows us to quickly recompute the articulation between the modified ontology and other ontologies.

The Tsimmis[PGMU96] and the Coin[CHG] projects entirely depend upon rules written by human beings to enable mediation.

As part of the linguistic algorithms, ONION leverages the work done at Stanford on the SKEIN system. Jannink [Jan00] proposed and implemented the SKEIN system that develops a Word Nexus. As indicated in the previous chapters, the Nexus is built from either of two dictionaries (Websters and the Oxford English Dictionary) and contains graphs of words connected to each other. A directional connection is generated when a word appears in the definition of another. The distance between words gives a measure of their similarities. The SKEIN system also provides other algorithms to determine the similarities between words. ONION has been coupled with the SKEIN system and utilizes the word similarity measures generated by SKEIN. This generated similarity is used as a heuristic among many other heuristics that ONION uses to determine the articulation rules. Jannink [Jan00] has shown that the manual and machine effort can be significantly reduced by using our methodology with respect to the construction of a Nexus (graphical thesaurus-like structures derived from the Webster's and the Oxford English Dictionary).

7.4 Ontology Management Algebras

The primary inspiration of developing an Ontology Management Algebra comes from the seminal work of Codd [Cod70] on the relational algebra for database management. Ontologies are analogous to schemas in relational algebra. Techniques used for ontology matching have also been used for schema matching and vice-versa. Our articulation of ontologies has similarities with joining multiple tables in a database system. Query optimization, more specifically join optimization, depends upon the properties of the operators in the relational algebra. Similarly, the composition of ontologies using our algebra depends upon

the properties of the ontology composition algebra.

The immediate inspiration for my work comes from the work of Wiederhold [Wie94], who proposed the necessity of an algebra for ontology composition. The current work brings the vision to reality in that it defines the algebra, and characterizes the properties of the algebraic operators. I have shown how the properties of the algebra depend upon articulation generation functions and thus argued for the careful design and implementation of such functions.

To the best of my knowledge, there exists no other comprehensive work on an ontology management algebra. After the initial publication of my ontology management algebra [MKW00], the following similar works have also been published in the research literature.

7.4.1 An Algebra for Composing Spatial Ontologies

Camara, et al., [CFM02], describe an algebra for composing spatial ontologies. However, their work only obtains an "Union" of ontologies. Unlike my approach where the ontologies and articulation rules can contain any relationships, they re-strict themselves to only three relationships (equivalent, is-a, and part-of). In addition, they do not consider the effects of the articulation generating function on the properties of the composition operation. However, since they are restricted to this basic set of connections, they can use the semantics of these relationships while obtaining the union of ontologies.

7.4.2 Model Management

Recently, Bernstein, et al. [Ber03], [BLP00] have proposed an an "algebra" for model management, which has similarities to our ontology algebra. The setting of their work is meta data management and they have used their algebra to build a model management platform [MRB03]. In their work, the operands are models and mappings between models. They describe models as abstract concepts that are similar to concepts in our ontologies. Models, like ontologies, have nodes.

Their work has identified a set of algebraic functions that can be used to manipulate models - *Match*, *Merge*, *Diff*, *Compose*, *Apply*, and *ModelGen*. *Match* takes two models as input and returns a mapping between them. This operation is similar to an articulation generation function in my framework. The elegance of ONION in my view lies in its layered

approach. On one hand, the layer responsible for establishing semantic correspondences depends upon semantic interpretations and is encapsulated within the articulation generation function. On the other hand, the algebraic machinery mechanically manipulates ontologies and does not attempt to interpret the semantics of the ontologies. The algebraic machinery depends upon the existence of the articulation generation function. In Bernstein's work, the two layers have been flattened into one. That is, some operators, like *Match*, require semantic interpretation of the ontolgies, while others, like *Apply*, do not.

From their preliminary descriptions, their "diff" operator is analogous to "difference" in my work, "merge" is similar to a duplicate-eliminating "union" that we have defined. Their algebra does not have an analog to *Intersection* but they have a "match" operator that is analogous to an articulation generation function. However, they require the root nodes of each model to match. Their mappings must be expressed using a "mapping" structure that primarily represents binary relationships, whereas we have a rule language to express my articulation rules. Their description of the algebra is at a rudimentary stage. Unlike in my work, they have not yet studied the properties of the algebraic operators and have not shown how the semantic matching functions influence them.

7.4.3 The Maintenance Problem

Oliver [Oli00] in her dissertation highlighted the problems that arise due to changes in ontologies. She observed that such changes are frequent in the medical informatics domain and proposed a methodology for change management and synchronization of local and shared versions of a controlled vocabulary. I believe that, though such change management and synchronization can resolve problems that arise due to heterogeneity of information can succeed in controlled domains like medical vocabularies, it has no significant chance for wide-spread success in large uncontrolled domains like publishing on the World Wide Web. Therefore, an interoperation approach is vital if information from multiple uncontrolled sources need to be composed. The ontology management algebra provides a description language for expressing composition tasks that derive new ontologies. Changes in source ontologies can be systematically and easily propogated to the derived ontologies automatically in this algebraic framework.

7.5 Composition

In this sub-section, I outline other knowledge composition efforts that are related to my work.

7.5.1 Composition of web services

Sheng et al. [SBDM02] have built and demonstrated a platform for the rapid composition of Web services in a peer-to-peer enironment. Their approach is based on using statecharts to declaratively specify the composition of composite Web serices from its components. A member identifies itself as part of a community and registers with the community. This approach though more streamlined than our interoperation scenario would involve the willingness of individual members converging to a basic agreement over to standards, protocols and semantics. The assumption of reaching an agreement does not hold in our scenario.

Rapid Knowledge Composition

Rapid composition for new applications has been demonstrated within the DAML and RKF projects. Participating knowledge engineers must comprehend an ontology in order to accurately gauge how changes to it will affect the performance of the associated information source. At a workshop, a presentation [Cha00] described preliminary efforts to align and merge two ontologies containing over 3000 concepts a piece. The original structure of the merged ontology was so fragile that no substantial transformation or enhancement was feasible. Many current approaches envisage much larger ontologies [cyc].

Chapter 8

Conclusions and Future Work

In this chapter, I summarize the contributions of this dissertation. I also outline extensions to the related research that needs to be performed in the future. To implement an ontology interoperation system that enables a user application to interoperate among heterogeneous information systems, there are many issues that are beyond the scope of my work and still need to be resolved.

8.1 Summary of the Results in the Thesis

In Chapters 2, 3, and 4, I have outlined the ONION system. ONION enables interoperation among heterogeneous information system. The primary roadblock towards building an automated interoperation system arises due to semantic heterogeneity among the information sources. This work lays out the foundation for building an interoperation system and provides a partial implementation. In Chapter 3, I have described how I constructed a semi-automated articulation-rule generator. The semi-automated articulation-rule generator interacts with a domain expert to resolve semantic heterogeneities and establish articulation rules. Based on the existence of a function that generates articulation rules between ontologies, I have designed an algebraic framework for ontology management. I summarize the contributions of my work below.

8.1.1 A Graph-Oriented Model for Ontologies

In order to enable interoperation among ontologies, several problems that arise due to heterogeneity of the information sources must be tackled. Problems due to heterogeneity of

platforms, operating systems and messaging interfaces have been addressed before. This dissertation looks at important problems that arise due to the heterogeneity of the information in the sources. One of the problems of heterogeneous information sources is that the semantics of the information differs from source to source. To automate the accurate processing of information from heterogeneous information sources, publishers of information are also publishing meta-data - data describing the semantics of the information - like ontologies.

However, there has been no agreement on the format and language used for representing ontologies. In the ONION system, I proposed and used a simple graph-oriented model to represent ontologies. The simplicity of the graph-oriented model gives us allows for fast computation. Concepts in ontologies are represented as nodes in the ontology graphs and the relationships among concepts as edges. Rules expressing constraints that the concepts must satisfy are encoded as Horn Clauses - simple logic-based rules. Simple wrappers have been built to convert ontologies from their native formats to the ONION Ontology Format. Converting all ontologies into the ONION Ontology Format, simplifies the construction of the articulation generator. The articulation generator now has to deal with only one simple ontology format. It can focus on only resolving the semantic heterogeneity in the information sources rather than simultaneously having to resolve heterogeneity due to differences in ontology formats.

8.1.2 SKAT: The Articulation-Rule Generator

SKAT deploys semi-automatic techniques to generate articulation rules between ontologies. SKAT generates articulation rules and presents them to a domain expert to ratify the rules. The expert can verify a rule and delete it if it is not accurate or correct it. The expert can also provide other new rules that the automated component failed to generate. The responses of the expert are logged and can be used in future articulation-rule generations between similar ontologies.

I have described several heuristic algorithms that comprise the automated component of the articulation-rule generator. While non-iterative algorithms make a single pass over the two ontologies, others (iterative algorithms) must make several passes over the ontologies until the set of generated articulation rules converges to a stable set. Iterative algorithms use the articulation rules generated after one pass to generate more articulation rules in subsequent iterations.

The algorithms discussed in the previous chapters match ontologies based on the names of their concepts, dictionaries like WordNet or the Nexus, corpus-based word-similarity methods, the structure of the ontology graphs, the instances of concepts (if available) and inference using expert-supplied seed rules. I show that no individual strategy works best in all cases. Instead, a hybrid strategy that combines the scores generated by several matching algorithms discovers more valid matches with high precision.

The matching algorithm that generated the most number of matches among the ontologies that I experimented with is the lexical matching algorithm. This algorithm depends upon the fact that concept names contain semantic information about the concept. It matches all pairs of concept names where the two names in a pair are obtained from the two distinct ontologies that are being matched. After generating the matches, articulation-rule generator selects the matches with a score above the expert-supplied threshold and shows them to the expert for validation. The algorithm uses dictionaries, semantic networks, and corpuses of documents to determine the matches between concept names.

I realize that automatic resolution of semantic heterogeneity automatically can never be perfect; human interaction is essential. Even then imprecision will remain. In the end, the assessment must be based on the benefits of integration, versus the cost due to errors that are a result of imperfect resolution of heterogeneity.

8.1.3 Ontology Management Algebra

As we have seen in the previous chapters, an ontology contains named concepts and their relationships, along with constraints that the concepts adhere to. Creating an ontology from scratch is an expensive endeavor. Since a large number of ontologies from various domains exist, it is a lot more pragmatic - less expensive - for publishers of information to create the ontology of their liking by reusing existing ontologies. Besides, existing ontologies that have been used, have been reviewed and well tested and tend to be more accurate than newly created ontologies. I have proposed and studied an ontology management algebra that can be used to select and compose portions of existing ontologies to create new ontologies.

The ontology management algebra has unary and binary operators. The unary operators provide several variations of the select operation. A publisher can use the select operation to select any relevant subset of concepts, relationships and rules from existing ontologies. The three binary operators are based on the availability of an articulation-rule-generating function. These operations can be used to compose selected portions of existing ontologies.

The *Union* operation creates a new ontology consisting of the concepts in the two operand ontologies and the relationships and rules introduced by the articulation-rule-generating function among them. The *Intersection* operation creates a new ontology that contains the shared concepts, that is, concepts in one of the operand ontologies that have a related concept in the other. The *Difference* operation creates a new ontology that contains concepts in the first ontology such that there exists no related concept in the second operand ontology.

Since the definition of the operations are based on the articulation-rule-generating function, the properties of the ontology management algebra are defined by those of the articulation-rule-generating function. The algebraic operators do not obey the laws of commutativity, associativity, and distributivity for all possible articulation-rule-generating functions. However, for certain articulation-rule-generating functions, these properties hold. In these cases, an ontology composition engine, can reorder the operands and operations in an ontology composition expression and optimize the composition task. The advantage of expressing a composition task declaratively using an ontology management algebra is that when the ontologies change, the composition task can be easily replayed to reconstruct the derived ontologies automatically and does not involve human intervention.

8.1.4 Modularity of Onion

The ontology algebra I have shown, however, is not restricted to being applicable to only the articulation generator used by ONION. The algebraic framework developed is modular in nature since it is decoupled from SKAT the articulation-rule generator. It assumes the presence of an articulation-rule-generating function. The articulation-rule-generating function can be an automated matcher like PROMPT or Chimaera or may be a manual effort. The algebra is based on the articulation rules - the output of the articulation-rule-generating function. Thus, the properties of the articulation-rule-generating functions influence the properties of the algebraic operators. To the best of my knowledge, the rules generated by PROMPT and Chimaera match concepts across ontologies and express them as rules. Thus, they have the same properties as Skat's articulation-rule-generating functions. Therefore, the algebraic framework developed in this dissertation is equally applicable to them.

8.2 Future Work

Since this dissertation outlines a new approach to integration of information, it is also quite incomplete and open to extension. The research outlined in this thesis raises a number of questions that need intensive investigation. In the rest of the chapter, I initially outline the ways in which the ONION system can be extended, and then I discuss broader issues that must be investigated in order to design better interoperation systems.

8.2.1 Extensions to Onion

Optimizing OntoStore

ONION stores its ontologies, their articulations and logs of expert feedback as RDF-triples. One option is to store these triples in databases. Though years of research has made database systems optimized and scalable, since they were not designed to store one large table consisting of triples (corresponding to the ontologies), substantial research and progress is necessary before an RDF-store can achieve the performance levels and scalability of traditional database systems. Problems of this nature have been encountered in the Protege project [pro04].

Presently, I implemented a central repository storing all the ontologies of interest. ONION is a system that attempts to solve problems in distributed systems. Therefore, it makes sense to design a distributed ontology store. However, if SKAT has to incur a large number of round-trip communication costs in order to articulate ontologies the system will become very slow. In the future, to avoid such numerous round-trips, one can build an ontology cache. Before each use, the version of the ontology in the cache is checked with the version of the ontology at its source. If the version has changed, and the application has indicated that the latest version of the ontology be used, the latest version of the ontology is fetched from source before proceeding. If the application does not need the latest version of the ontology, the cached version is used to boost performance.

Eager versus Lazy Approach

This issue is similar to the issue with respect to compilation versus interpretation for programming languages.

As argued above, before information from multiple sources can be composed, the ontologies used to express their vocabularies must be articulated. An articulation generator

has to decide when to articulate the ontologies. A system taking an eager approach would pre-articulate the ontologies and establish the articulation rules in advance before the need arises to use the articulation rules. Such articulation rules are stored and used while answering queries or otherwise composing information from multiple sources. An alternative approach is to take a lazy approach towards establishing articulation rules. In such a scenario, the rules are established on demand in order to answer a query. The rules can then be cached for future use to avoid regenerating them in the near future. ONION is built on the eager approach - that is, it pre-articulates ontologies before their articulation is required. However, the question of whether an eager approach is better than a lazy one needs experimental validation in a realistic setting.

Articulation-Rule Generation Algorithms

Substantial scope exists to design better articulation-rule generation algorithms, since the performance of ONION and other state-of-the-art ontology and schema matchers leaves ample room for improvement. I discuss a few improvements I intend to perform in the near future. However, I believe that there exist entirely new paradigms for articulation-rule generation that remain unexplored and will be discovered in the near future.

Corpus-Based Matcher: SKAT matches concept names associated with ontology concepts using the corpus-based method. Typically, concept-names are constructed using less than ten words. However, some sources contain short textual descriptions, along with the concept names. If I treat these descriptions as extended "concept names" and run SKAT, preliminary results (not shown in the dissertation) show that SKAT can generate good matches. Continuing along, on the other end of the spectrum, entire documents can be associated with concepts and an articulation-rule generator can derive useful information and match concepts based on the documents associated with it. However, I suspect that in such cases document similarity algorithms will outperform SKAT's name-matching algorithm. How many words defines "large" concept names and the number beyond which ONION's name-matching algorithms perform worse than document similarity algorithms has not been studied in this dissertation and will be pursued in future.

Another investigation that needs to be performed is to identify metrics to evaluate how relevant the corpora the matching algorithm uses is to the domains of the source ontologies. Once the metric has been fixed, we will study the effect of the quality of the corpora on the

quality of the ontology matches generated automatically.

Instance-Based Matcher:

If the information about instances of concepts in ontologies are available, the information can be used to generate additional matches. Instance-based matching heuristics have been used to successfully match schemas in databases [YMHF01]. Such matchers look at data types, and extract other features like lengths of attributes, numerical or lexical statistics of attributes, and match classes based on such feature vectors. Though SKAT can handle ontologies, whose concepts also have instances associated with them, oftentimes, businesses are reluctant to make instances available. Thus, I have designed our algorithms assuming that no instance data is available.

However, if such information is available, one can extend the matcher to use instance information. An interesting approach is to use heuristic algorithms based on information theory. Kang and Naughton [JK03] have proposed using a schema-matching algorithm based on the mutual information of the values in pairs of columns in two database tables. I believe that instead of mutual information, conditional entropy is a better measure. I performed initial experiments that prove this belief.

For instance-based methods, there are several low-level questions that also need investigation. For example, how should we treat null-data? The fact that many instances of a column are null might be valuable information that might help us match the columns of a database better. So, in instance-based methods, do we generate better matches ignoring null-values, treating null-value as another type of value or by treating each null as potentially different values? The right semantics of null and how nulls should be treated is an open question.

An example of another question that requires empirical validation is how much data is required for instance-based methods to make statistically significant deductions.

Inference-based Heuristics:

Currently, I have not implemented inference-based heuristics, and ONION does not use inference-based heuristics. Here, I discuss how inference-based heuristics can be used for generating articulation rules for the sake of completeness. An inference engine can reason with the rules available with the ontologies and any seed rules provided by an expert to generate matches between the ontologies. A future enhancement to having the expert explicitly specify the matches is to use an inference engine like TRIPLE [DS02].

Inference-based heuristic matchers rely upon rules that express properties of relationships. Instead of specifying the entire set of individual matches that were not generated automatically, the expert can supply a much set of logical rules. Using the available facts, the expert-supplied logical rules and an inference engine, the articulation-rule generator can generate the set of articulation rules that are essential for the application. Since this feature has not yet been implemented, I have included some examples of cases where using an inference engine is essential below.

Using Type Information in Matching:

Currently, in ONION the concept nodes are not typed. In the future, we can implement typing support. When type information is available, an articulation-rule generator can use the type information associated with a node to pre-screen matches. Matching nodes must be of the same type or one must be of a type that is a subtype of the other.

8.2.2 Extensions to the Ontology Composition Algebra

Articulation Rules

At present, the Ontology Composition Algebra can handle articulation-rule-generating functions that generate articulation rules all of whose nodes are in the source ontologies. In the future, one can easily extend the algebra to allow the more general form of articulation rules, where an articulation rule can introduce and use a concept not present in any of the source ontologies.

For example, while articulating between *PoundSterling* and *Guilders*, an articulation-rule-generating function f first generates an intermediate node called *Euro*. Then, f generates an articulation rule to convert between *PoundSterling* and the *Euro* and another rule between the *Guilder* and the *Euro*. Not surprisingly, the presence of such an intermediate node influences the properties of the algebraic operators. For example, if an articulation-rule-generating function generates intermediate nodes, the intersection operation between ontologies can not be guaranteed to be associative. Thus, I do not consider such articulation-rule-generating functions in this work but it is an interesting problem to handle in future.

Extending the Operators

Intersection: As future work, I intend to extend the system by incorporating a duplicate-eliminating *Intersection* operation. To use this extension, the application must indicate a preferred ontology out of the two source ontologies being articulated to the system. For all articulation rules of the form $O1.A\ EquivalentTo\ O2.B$, if the preferred ontology is $O1$,

$O2.B$ is not included in the set of nodes of the intersection ontology. For all edges e such that $Nodes(e)$ contains $O2.B$ are rewritten replacing $O2.B$ with $O1.A$. Similarly, if $O2$ is the preferred ontology, all occurrences of $O1.A$ are replaced by $O2.B$ in the ontology graph of the intersection ontology.

Like a duplicate-eliminating *Intersection*, I also intend to implement a duplicate-eliminating *Union*.

An investigation characterizing the duplicate-eliminating *Intersection* and *Union* also needs to be undertaken.

8.2.3 Information Extraction and Data Mining

In order to obtain maximum automation, especially in rapidly changing sources where manual extraction of information is inefficient, one must study the problems of extracting structured and unstructured data from the Web. It is important to investigate techniques for mining the data to extract accurate information.

8.2.4 Maintenance and its effect on Information Integration

Though I have designed the system to be easily maintainable, I have not extensively studied the problems caused by frequent changes of websites. As information and software inventories grow, one needs more and more resources to keep them up-to-date. What modifications or enhancements does ONION or a similar system need to allow rapid re-deployment of the toolkit in the face of frequent updates? Do the operators that I have defined suffice or do we need more to serve a wide range of real-world applications? More investigation is necessary to determine how we can rapidly regenerate articulations without having to run the entire articulation-generation toolkit and perform a thorough performance evaluation of the articulation-generation algorithms in such a setting.

8.2.5 Applications and Scalability

Apart from the Semantic Web for which ONION was developed, the problem of data integration and interoperation is acute in several fields. One must work closely with scientists who use large sources of distributed information like in domains such as bio-informatics, medical informatics, and operations research to find settings to deploy the tool I have developed.

I have not evaluated ONION for large systems and the issues of scalability in real-world applications will require many related investigations.

8.2.6 Query Rewriting and Optimization in Peer-to-Peer Systems

Peer-to-peer systems have several similarities to the distributed, autonomous and heterogeneous systems that I have worked on. My work on query rewriting and optimization has the potential to be used in peer-to-peer systems. We need to investigate the similarities and differences between the setting of this dissertation with peer-to-peer systems and accordingly modify our algorithm to design an efficient query rewriting and optimization algorithm for peer-to-peer systems. While researching information-processing systems, we need to design and use tools and algorithms from a wide range of fields from database system design, query optimization, data mining and information extraction to natural language understanding, graph visualization and human computer interaction, and computational logic.

8.3 The Web

Much work is needed to extend the ONION system or build a new successor system that can handle a variety of formats available on the web, and deals with the rapid changes of large amounts of information. Many of the existing solutions work for small domains but when confronted with very large numbers of information sources and large amounts of information — a significant part of which might overlap — the existing techniques fall flat. While dealing with information on the World-Wide-Web the issues like enabling security, and privacy, also take on paramount importance and must be researched thoroughly. There is also the issue of large amounts of information hidden in the deep Web that needs to be extracted and interoperation among such deep websites enabled.

Bibliography

[ACM01] Serge Abiteboul, Sophie Cluet, and Tova Milo. Correspondence and translation for heterogeneous data. *To appear in Theoretical Computer Science http://osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=abiteboul*, 2001.

[AK97] N. Ashish and C. A. Knoblock. Wrapper generation for semi-structured internet sources. In *Proc. Workshop on Management of Semistructured Data*, Tucson, 1997.

[all04] Project halo. available at http://www.projecthalo.com/, May 2004.

[ALM02] Foto N. Afrati, Chen Li, and Prasenjit Mitra. Answering queries using views with arithmetic comparisons. In *21st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 209–220, 2002.

[ALM04] Foto N. Afrati, Chen Li, and Prasenjit Mitra. On containment of conjunctive queries with arithmetic comparisons. In *Advances in Database Technology - EDBT 2004, 9th International Conference on Extending Database Technology*, volume 2992 of *Lecture Notes in Computer Science*. Springer Verlag, March 2004.

[alt] http://www.altavista.com.

[AYL95] Marie-Christine Rousset Alon Y. Levy. Combining rules and description logics: An overview of carin. In *ILPS*, page 635, 1995.

[BCM+03] Franz Baader, Diego Calvanese, Deborah McGuiness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook, Theory Implementation and Applications*. Cambridge University Press, January 2003.

[Ber03] P.A. Bernstein. Applying model management to classical meta data problems. In *CIDR*, 2003.

[BLP00] P.A. Bernstein, A.Y. Levy, and P. Pottinger. A vision for management of complex models. Technical report, Microsoft Research, June 2000.

[BLR97] C. Beeri, A. Y. Levy, and M. Rousset. Rewriting queries using views in description logics. In *Proc. of the 16th ACM Symposium on Principles of Database Systems*, pages 99–108, 1997.

[Boo54] George Boole. *An investigation into the Laws of Thought, on Which are founded the Mathematical Theories of Logic and Probabilities.* 1854.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[Bro95] Frederick P. Brooks. Addison-Wesley Publishing Company, 20th anniversary edition (2nd edition) edition, August 1995.

[CFM02] Gilberto Camara, Frederico Fonseca, and Antonio Miguel Monteiro. Algebraic structures for spatial ontologies., September 25-28 2002.

[CGL99a] D. Calvanese, G. De Giacomo, and M. Lenzerini. Answering queries using views in description logics. In *Proceedings of the 6th International Workshop on KRDB*, 1999.

[CGL99b] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Answering queries using views in description logics. In *Knowledge Representation Meets Databases*, pages 6–10, 1999.

[Cha00] H. Chalupsky. Ontomorph: A translation system for symbolic knowledge. In *KR 2000*, pages 471–482. Morgan Kaufmann Publishers, Apr 2000.

[CHG] M. D. Siegel C. H. Goh, S. E. Madnick. Semantic interoperability through context interchange: Representing and reasoning about data conflicts in heterogeneous and autonomous systems http://citeseer.nj.nec.com/191060.html.

[CIA00] Cia factbook: http://www.cia.gov/cia/publications/factbook/. 2000.

[CMM01] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109–118, 2001.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[cyc] Cyc knowledge base, http://www.cyc.com/.

[dama] Daml-features. available at http://www.daml.org/language/features.html#qualified, obtained on may 24th, 2004.

[damb] Daml http://www.daml.org/.

[dam01] Daml+oil available at http://www.daml.org/2001/03/daml+oil-index. 2001.

[DDH01] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD 2002*, 2001.

[DMDH02] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *WWW 2002*, 2002.

[Doa02] Anhai Doan. *Learning to Map between Structured Representations of Data*. PhD thesis, University of Washington, 2002.

[DS02] S. Decker and M. Sintek. Triple—a query, inference, and transformation language for the semantic web. http://triple.semanticweb.org. In *1st International Semantic Web Conference (ISWC)*, June 2002.

[GBMS99] C. H. Goh, S. Bressan, S. Madnick, and M. Siegel. Context interchange: new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3):270–270, 1999.

[GKD97] M.R. Genesereth, A.M. Keller, and O.M. Duschka. Infomaster: An information integration system. In *ACM SIGMOD '97*, pages 539–542. ACM, 1997.

[GMF+02] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crube'zy, H. Eriksson, N. F. Noy, and S. W. Tu. The evolution of prot'ge': An environment for knowledge-based systems development. Technical report, SMI, 2002.

[GMJ] Avidor Gal, Giovanni Modica, and Hasan Jamil. Improving web search with automatic ontology matching.

[GPVG90] M. Gyssens, J. Paredaens, and D. Van Gucht. A graph-oriented object database model. In *Proc. PODS*, pages 417–424, 1990.

[Gru93] Tom R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In Nicola Guarion, editor, *Padua Workshop on Formal Ontology*, march 1993.

[Guh91] R. V. Guha. *Contexts: A Formalization and Some Applications*. PhD thesis, Stanford University, 1991.

[Har90] S. Harnad. The symbolic grounding problem. *Physica D*, 42:335–346, 1990.

[HBLM01] J. Hendler, T. Berners-Lee, and E. Miller. The semantic web. *Scientific American*, May 2001.

[Hov98] E.H. Hovy. Combining and standardizing large-scale, practical ontologies for machine translation and other uses, 1998.

[HPK] Hpkb crisis management challenge problem specification. as obtained from http://www.iet.com/projects/hpkb/y2/ on may 24th, 2004.

[Hsu98] C. Hsu. Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules. In *AAAI-98 Workshop on AI and Information Integration*, pages 66–73. AAAI Press, 1998.

[htm] Hypertext markup language (html) homepage. as obtained from "http://www.w3.org/markup" on may 24th, 2004. Technical report.

[Jan00] J. Jannink. *A Word Nexus for Systematic Interoperation of Semantically Heterogeneous Data Sources*. PhD thesis, Stanford University, 2000.

[JK03] Jeffrey F. Naughton Jaewoo Kang. On schema matching with opaque column names and data values. In *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, June 2003.

[Jon98] T. Capers Jones. *Estimating Software Costs*. McGraw-Hill, 1998.

[JSV+98] Jan Jannink, Pichai Srinivasan, Danladi Verheijen, , and Gio Wiederhold. Encapsulation and composition of ontologies, July 1998.

[Kar96] P. D. Karp. A strategy for database interoperation. *Journal of Computational Biology*, 2(4):573–583, 1996.

[KL94] K. Knight and S. Luk. Building a large knowledge base for machine translation. In *Proceedings of the American Association of Artificial Intelligence Conference AAAI*, 1994.

[KLSS95] Thomas Kirk, Alon Y. Levy, Yehoshua Sagiv, and Divesh Srivastava. The information manifold. In C. Knoblock and A. Levy, editors, *Information Gathering from Heterogeneous, Distributed Environments*, Stanford University, Stanford, California, 1995.

[KMA+98] C.A. Knoblock, S. Minton, J.L. Ambite, N. Ashish, P.J. Modi, Ion Muslea, A.G. Philpot, and S. Tejada. Modeling web sources for information integration. In *Proc. of the Fifteenth National Conf. on Artificial Intelligence, Madison, WI*, 1998.

[Kus00] N. Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118(1-2):15–68, 2000.

[KWD97] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 729–737, 1997.

[Law] Kincho Law. Communications with prof. kincho law.

[LMSS95] A.Y. Levy, A. Mendelzon, D. Srivastava, and Y. Sagiv. Answering queries using views. In *Proc. of Symposium on Principles of Database Systems, San Jose, CA*, pages 163–173, 1995.

[LRO96] A.Y. Levy, A. Rajaraman, and J.O. Ordille. Query-answering algorithms for information agents. In *Proc. of the 13th National Conference on Artificial Intelligence, AAAI-96*, 1996.

[Mah96] Kavi Mahesh. Ontology development for machine translation: Ideology and methodology. http://citeseer.ist.psu.edu/mahesh96ontology.html. Technical report, NMSU, 1996.

[MBR01] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *VLDB 2001*, pages 49–58. Morgan Kaufmann, September 2001.

[ME] Philippe Martin and Peter Eklund. Knowledge representation and reuse on the www: Conventions, rdf extentions, and simple notations.

[Mel] Sergey Melnik. http://www.interdataworking.com/converter/.

[Mel00] S. Melnik. Declarative mediation in distributed systems. In *Proceedings of the International Conference on Conceptual Modeling (ER'00)*, 2000.

[Mel02] Sergey Melnik. Rdf api draft, June 2002.

[MFRW00] D.L. McGuiness, R. Fikes, J. Rice, and S. Wilder. The chimaera ontology environment. In *Seventh National Conference on Artificial Intelligence (AAAI-2000)*, 2000.

[MGMR02] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proc. of the 12th Int. Conf. on Data Eng. (ICDE)*. IEEE Computer Society, February 2002.

[Mit01] Prasenjit Mitra. An algorithm for answering queries efficiently using views. In *Proceedings of the 12th Australasian conference on Database technologies*, pages 99–106. IEEE Computer Society, 2001.

[MKW00] P. Mitra, M. Kersten, and G. Wiederhold. A graph-oriented model for articulation of ontology interdependencies. In *Advances in Database Technology- EDBT 2000*, Lecture Notes in Computer Science, 1777, pages 86–100. Springer-Verlag, 2000.

[MMAU03] Bill MacCartney, Sheila A. McIlraith, Eyal Amir, and Tomas Uribe. Practical partition-based theorem proving for large knowledge bases. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence, (IJCAI-03)*, August 2003.

[MMK01] I. Muslea, S. Minton, and C. A. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 4(1/2):93–114, 2001.

[MRB+96] A.T. McCray, A.M. Razi, A.K. Bangalore, A.C. Browne, and P.Z. Stavri. The umls knowledge source server: A versatile internet-based research tool. In *Proc. AMIA Fall Symp*, pages 164–168, 1996.

[MRB03] S. Melnik, E. Rahm, and P.A. Bernstein. Rondo: A programming platform for generic model management. In *ACM SIGMOD*, June 2003.

[MW02] Prasenjit Mitra and Gio Wiederhold. Resolving terminological heterogeneity in ontologies., July 2002.

[MWJ99] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[nat] North atlantic treaty organization. as obtained from http://www.nato.int/ on 24th may, 2004.

[NM00] N.F. Noy and M.A. Musen. Prompt: Algorithm and tool for automated ontology mergin and alignment. In *7th National Conference on Artificial Intelligence (AAAI-2000)*, 2000.

[NM01] N. F. Noy and M. A. Musen. Anchor-prompt: Using non-local context for semantic matching. In *Workshop on Ontologies and Information Sharing at the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-2001). Available as SMI technical report SMI-2001-0889*, August 2001.

[Oli00] Diane E. Oliver. *Change Management and Synchronization of Local and Shared Versions of a Controlled Vocabulary.* PhD thesis, Stanford University, 2000.

[Ont] Ontolingua, http://www-ksl-svc.stanford.edu:5915/doc/project-papers.html.

[ope] Opec. as obtained from http://www.opec.org/ on may 24th 2004.

[OSSM98] D.E. Oliver, Y. Shahar, E.H. Shortliffe, and M.A. Musen. Representation of change on controlled medical terminologies. In *Proc. AMIA Conference*, Oct. 1998.

[owl04] Web ontology language (owl) available at http://www.w3.org/2004/owl/, July 2004.

[pea] Re: Assertion, question, suggestion - final draft. email communication to daml-all mailing list, archived at http://www.daml.org/listarchive/daml-all/0173.html. obtained on may 24th, 2004.

[PGMJ95] Y. Papakonstantinou, H. Garcia-Molina, and Widom J. Object exchange across heterogeneous information sources, March 1995.

[PGMU96] Yannis Papakonstantinou, Hector Garcia-Molina, and Jeffrey D. Ullman. Medmaker: A mediation system based on declarative specifications. In Stanley Y. W. Su, editor, *Proceedings of the Twelfth International Conference on Data Engineering, February 26 - March 1, 1996, New Orleans, Louisiana*, pages 132–141. IEEE Computer Society, 1996.

[pro04] The protégé project, 2004.

[rdf99] Resource description framework(rdf) model and syntax specification, w3c recommendation http://www.w3.org/tr/rec-rdf-syntax. 1999.

[rdf00] Resource description framework(rdf) schema specification 1.0, w3c recommendation http://www.w3.org/tr/rdf-schema. Technical report, W3C: World Wide Web Consortium, 2000.

[RGG+94] A.L. Rector, A. Gangemi, E. Galeazzi, A.J. Glowinski, and A. Rossi-Mori. The galen core model schemata for anatomy: Towards a re-usable application-independent model of medical concepts. In *Twelveth International Congress of the European Federation for Medical Informatics, MIE-94*, pages 229–233, 1994.

[RKS+94] O. Ritter, P. Kocab, M. Senger, D. Wolf, and S. Suhai. Prototype implementation of the integrated genomic database. *Computers and Biomedical Research*, 27:97–115, 1994.

[RL02] Stephen Reed and Douglas Lenat. Mapping ontologies into cyc, July 2002.

[RSB98] C. Rosse, L. G. Shapiro, and J. F. Brinkley. The digital anatomist foundational model: Principles for defining and structuring its concept domain.

In *American Medical Informatics Association Fall Symposium. Available at http://sigpubs.biostr.washington.edu/archive/00000098/ on May 29th, 2004*, pages 820–824, 1998.

[Rus12] Bertrand Russell. Home University Library, 1912.

[RW86] V.V. Raghavan and S.K. Wong. A critical analysis of vector space model for information retrieval. 37(5):279–87, 1986.

[SBDM02] Quan Z. Sheng, Boualem Benatallah, Marlon Dumas, and Eileen Oi-Yan Mak. Self-serv: A platform for rapid composition of web services in a peer-to-peer environment. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, September, 2002, Hong Kong, China*. Morgan Kaufmann, 2002.

[Sch92] Hinrich Schuetze. Dimensions of meaning. In *Supercomputing*, pages 787–796, 1992.

[sgm] Sgml and xml as (meta-) markup languages. as obtained from http://xml.coverpages.org/sgml.html on may 24th, 2004.

[Smu95] Raymond M. Smullyan. *First-Order Logic*. Dover Publications, January 1995.

[Sod99] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3):233–272, 1999.

[Tek97] Teknowledge. High-performance knowledge-bases (hpkb), maintained by teknowledge corp. for darpa. as obtained from http://www.teknowledge.com/darpa on 24th may, 1998, 1997.

[Ull97] J.D. Ullman. Information integration using logical views. In *Proc. of the International Conference on Database Theory, Delphi, Greece*, 1997.

[uml00] Unified modeling language: http://www.omg.org/technology/uml/index.htm. 2000.

[URI] Naming and addressing: Uris, urls, ... as obtained from http://www.w3.org/addressing/ on may 24th 2004.

[vee] Xml: Metadata for the rest of us (part 1). as obtained from http://hotwired.lycos.com/webmonkey/tools/97/27/index0a.html on january 20th, 1998.

[Wie94] Gio Wiederhold. An algebra for ontology composition. In *Proceedings of 1994 Monterey Workshop on Formal Methods*, pages 56–61. U.S. Naval Postgraduate School, sep 1994.

[wor] Wordnet - a lexical database for english. http://www.cogsci.princeton.edu/ wn/. Technical report, Princeton University.

[WSKR03] Kevin Wilkinson, Craig Sayers, Harumi Kuno, and Dave Reynolds. Efficient rdf storage and retrieval in jena2. Technical Report HPL-2003-266, HP Laboratories, 2003.

[xml] Xml schema part 0:primer.

[xml99] Extensible markup language (xml) 1.0 http://www.w3.org/tr/rec-xml, Feb 1999.

[xml04] Extensible markup language (xml) 1.1. as obtained from http://www.w3.org/tr/xml11 on may 24th, 2004. Technical report, 2004.

[YMHF01] L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *ACM SIGMOD*, 2001.

[ZB04] Songmao Zhang and Olivier Bodenreider. Comparing associative relationsh8ips among equivalent concepts across ontologies. In *Medinfo 2004: Building High Performance Health Organizations (to appear)*. American Medical Informatics Association (AMIA), September 2004.