ONTOLOGY AWARE SOFTWARE SERVICE AGENTS: MEETING ORDINARY USER NEEDS ON THE SEMANTIC WEB

by

Muhammed J. Al-Muhammed

A dissertation submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a dissertation submitted by

Muhammed J. Al-Muhammed

This dissertation has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	David W. Embley, Chair
Date	Charles D. Knutson
Date	Michael A. Goodrich
Date	Mark J. Clement
Date	Bryan S. Morse

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the dissertation of Muhammed J. Al-Muhammed in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date	David W. Embley Chair, Graduate Committee
Accepted for the Department	Parris K. Egbert Graduate Coordinator
Accepted for the College	Thomas W. Sederberg Associate Dean, College of Physical and Mathematical Sciences

ABSTRACT

ONTOLOGY AWARE SOFTWARE SERVICE AGENTS: MEETING ORDINARY USER NEEDS ON THE SEMANTIC WEB

Muhammed J. Al-Muhammed
Department of Computer Science
Doctor of Philosophy

To achieve the dream of the semantic web, it must be possible for ordinary users to invoke services. It is clear that users need simple-to-invoke-and-use services. This dissertation offers an ontological approach to declaratively create services that users can invoke using free-form, natural-language-like specifications. Our approach uses task ontologies as foundational knowledge. A task ontology consists of a domain ontology and a process ontology. The domain ontology encodes domain information such as possible constraints and instances in terms of object sets, relationship sets among these object sets, and operations over values in object sets and relationship sets. The process ontology consists of generic processes that are domain independent—coded once and work for all. Our system recognizes the constraints in a service request, discovers any missing information and obtains it from users, and formalizes the constraints in the context of the domain ontology. The system satisfies the constraints by obtaining information from databases associated with the domain ontology and providing users with solutions or near solutions when there is no way to satisfy all the constraints. Our experiments with our prototype implementation show that our approach can create services that satisfy end-user needs.

ACKNOWLEDGMENTS

Many thanks to my God; by his willing I achieved this dissertation.

I am very grateful to my advisor Dr. David W. Embley for his patience, understanding, and above all his tolerance. I owe him a lot and will never forget his passion and guidance. I am very grateful to Dr. Charles D. Knutson and Dr. Michael A. Goodrich who helped and encouraged me a lot in my research. I am also very grateful to Dr. Mark J. Clement and Dr. Bryan S. Morse for their time and valuable comments. God bless you all.

I am very grateful for all the people at the Department of Computer Science who taught me even a word or helped me in any way.

I am very grateful for all the people at BYU especially Dr. Sandra Rogers who is supporting me to succeed and Kate Andreason for her help. I am also grateful for the great BYU environment in which I have never felt that I am away from home.

Finally I am grateful for NSF for partially supporting this research under Grants 0083127 and 0414644.

Muhammed J. Al-Muhammed
Brigham Young University
Provo, Utah 84602
USA

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In open and ever-growing environments such as the world wide web, the amount of information is increasing at a tremendous rate. In addition, users are, or soon will be, overwhelmed with web services that give information, manage appointment calendars, sell products, and so on. This incredibly continuing increase in the amount of information and the number of services makes performing tasks such as finding information and services of interest quite challenging for web users. The semantic web along with personal software agents and web service systems purport to offer a solution to this challenge. But exactly how this solution will play out is still unclear.

The semantic web is an extension to the current web that makes the web not only human understandable but also "machine understandable." In particular, the semantic web is changing the content of the web—both information and services—to be both machine-interpretable and human-understandable [BLHL01]. This continuing change in the content of the web is increasing the ability of software agents to reason about the content of the web and to do tasks on behalf of users [Kun04, DW03, KKS+02, Hen01].

In this dissertation we offer a unique approach to turn the vision of semantic web pioneers into reality for everyday service requests such as scheduling appointments, selling, buying, renting apartments, and so forth. Our approach allows users to invoke services by only specifying their needs in a free-form request, rather than the typical approach of looking for services and using them. This approach lets users focus only on specifying what they need rather than on discovering services that can accomplish their needs and then figuring out how to use them, which is intuitively much harder. Further, our approach has capabilities to help users reach best solutions (or near solutions) for their requests and

therefore reduce the potential overload resulting from having to go through perhaps a long list of solutions (or near solutions).[1]

Our approach to allow specification-based service invocation centers around a *task ontology*. A task ontology can be thought of as having two component ontologies: a *domain ontology* and a *process ontology*. The domain ontology is (manually) created by a service developer who wants to provide a service in the domain. This ontology defines concepts in a domain of a service request along with relationships among these concepts and constraints and operations over the concepts and relationships. The process ontology defines generic processes for doing service requests and was pre-coded by the system developer. Interestingly, the process ontology is independent of any domain-specific information in the sense that it is coded once and works for all domains. The system automatically specializes it to a domain by passing the hand-crafted domain ontology to the process ontology. With a task ontology in hand, we address the following fundamental problems.

1. *Service-Request Specification.* The first key issue to address is how to allow users to specify their service requests in a simple way. In our system, we let users assume the existence of an intelligent agent within the system and specify their service requests textually in any way they wish.

2. *Service-Request Recognition.* After specification of the service request, our approach goes through a recognition process of the specified service request. This recognition process matches a service request specification against task ontologies to identify the task ontology that matches the service request the best. The system uses the ontology that matches best to generate a predicate-calculus formula that represents the constraints imposed on the service request and then uses this formula to execute the service request.

3. *Service-Request Execution.* Given a specified service request and the generated predicate-calculus formula for this service request, the system generates software that can do the service request. The software has the ability to gather the information it

[1] A solution is a fulfillment instance of a service request that satisfies all the imposed constraints while a near solution is a fulfillment instance that satisfies some (perhaps none) of the constraints. We will precisely define both a solution and a near solution in Chapter 4.

needs by interacting with the system and also the user, in case of incomplete specification. When a resolution is not immediate (i.e. when there are too many ways to service the request or there is no way to service the request given the imposed constraints), the software interacts with and helps a user find a satisfactory resolution to a request.

The chief objective of this dissertation is to show that our ontology-based approach can effectively resolve these problems. Our prototype system, which is implemented based on our ontological approach and which serves as a proof of concept, shows the potential of our approach for helping to achieve the vision of the semantic web for everyday service requests.

Before we give the details of our approach, we present an overview of the related work, the thesis statement, the contributions of the dissertation, and the organization of the dissertation.

1.1 Related Work

To the best of our knowledge our work is unique. We know of no research trying to satisfy free-form user service requests by establishing needed relationships within the context of the constraints imposed by an ontology. We do, however, wish to place our research within the framework of the current work on web services (Subsection 1.1.1). We also wish to compare our proposed system with work on task ontologies (Subsection 1.1.2) and software agents (Subsection 1.1.3), which have similarities to some aspects of our work.

1.1.1 Service-Oriented Architecture

Service-oriented architecture concepts have constituted "best practices" in enterprise-scale application development for the past three decades [KSB04, Erl05, GBR05, ABdB+06]. The key idea of a service-oriented architecture is that business functionality is packaged as reusable services that can be invoked through standard interfaces. Recently, the major standards-based approach to services has centered on web services as described by W3C's Web Services Description Language (WSDL) [CCMW01, CMRW07]. With XML as the

common underlying message format, WSDL defines a grammar for specifying services, and it is surrounded by a constellation of related standards including the Simple Object Access Protocol (SOAP) [W3C03], the Universal Description, Discovery, and Integration (UDDI) service registry mechanism [UDD03], and other proposals within W3C's Web Services Activity [WS05]. These technologies comprise the common foundation for current web services research.

However, this foundation is only an initial infrastructure, and advanced applications of web services require additional layers. WSDL, much like HTTP, is essentially a stateless protocol and thus requires additional architectural support for service discovery, composition, and transactional execution. For example, the Web Service Modeling Framework (WSMF) was designed to address decoupling and mediation needs [FB02]. The Internet Reasoning Service defines a framework for automatically publishing, locating, composing, and executing heterogeneous services [MDCG03]. BPEL4WS (Business Process Execution Language for Web Services), promoted by the OASIS (Organization for Advancement of Structured Information Standards) group, recognizes the need for higher levels of web-services "choreography" including, among other elements, support for transactional features not found in WSDL [ACD+03]. Research efforts related to semantic web services [MSZ01, SHS+02, PL04, CDM+04] often follow the DAML-S/OWL-S line of work [MPM+05, KA04, ABH+02, FB02, MA02]. The DAML Services Coalition observed early on that WSDL does not account for workflow needs, and so to support semantic-web services, they created what is now called OWL-S [OWL04]. OWL-S is an OWL-based web service ontology that provides constructs to describe web services in an unambiguous, machine-interpretable way with the hope to facilitate automatic web service discovery, composition, and execution.

Web services are characterized by many layers, and our approach builds on the existing web services stack in a complementary fashion. Our approach shows how ontologies can be used to provide semantic-web services in a user-friendly, highly automated way. We are not the first to use task and process ontologies to manage web services, but we do provide a unique approach that leverages the principles of ontology-based data extraction to hide complexity from the end user.

1.1.2 Task Ontologies and Web Service Derivation

Researchers have suggested the notion of a task ontology to define generic processes that can be assembled to do tasks. [MTI95] describes a task assembly system, called MULTS. In this approach, domain experts synthesize problem solving engines for their tasks from generic processes and building blocks defined in a "task ontology." [KG03] describes a system that uses a task ontology to represent web services. Users can compose a set of services to do a task and use this system to check whether the composition is valid. Our approach significantly differs from both of these efforts because it does not require either domain experts or other users to compose services.

Both [KA04] and [MA02] describe an approach for using a process ontology to index web services. Users can browse the process ontology or create queries using the defined process query language to find services of interest. Our approach significantly differs from these two approaches because it does not require users to look for services, and also it goes further by supporting specified tasks from beginning to end without requiring users to find or use services.

Researchers have proposed systems for the web that either make available web services from which users can choose or allow users to find, select, and compose their own tasks. [AHS03] describes a system that lets users browse web services and choose a web service to do a task or choose and compose several web services to do a task. [PKCH05] describes a prototype that lets users discover and select services of interest. This approach depends on the existence of a repository of web services described using OWL-S and on whether users can specify queries for services using OWL-S and can also provide a mapping between the terms in their queries and the terms used to describe the services. A matchmaking algorithm matches the requests against the advertised services and returns an ordered list of candidate services from which users can choose. [MDCG03] describes a system with pre-specified tasks that lets users choose from a set of available pre-specified tasks. [SHP03] describes a system that lets users select services from a list of known services and assists them to semi-automatically compose services to do some task. [SPW+04] describes a system that lets users select from a list of available services; it will then execute

the service. Our approach differs significantly from all these approaches in that our system does not require users to find services from among a set of available services, which is difficult, even as acknowledged by the authors of the cited papers themselves. In our approach the only requirements are that users specify their tasks, provide information in the typical case of incomplete task specification, and interact with the system when there is no solution because the imposed constraints are too strong or when there are many solutions because the imposed constraints are too weak. Service composition, however, remains in the hands of the user.

The METEOR-S [VSS+05] approach is geared toward enabling users to discover web services of interest. Users specify the web service using a service template, which usually consists of input and output information about the service. The system then finds the best matching services. Our approach goes further than just discovering a service of interest. It services a request for a service from beginning to end, including discovering a service, invoking the service, and helping users obtain solutions or agreed-upon near solutions.

The IRS-III [CDG+06] approach is a reference implementation for the Web Service Modeling Ontology (WSMO) [BBD+]. It enables service developers to describe their services using the WSMO ontology and register these descriptions in the system. Requesters can specify their requests, called goals, also using the WSMO ontology. The server matches goals with WSMO service descriptions and returns matches to users. Users then choose and invoke desired services. This approach differs from our approach in that (1) it requires users to select appropriate services from potentially lengthy list of services and invoke this service while in our approach a specification of a service is sufficient to invoke it and (2) it requires users to make service requests (goals) using the formal WSMO model while our approach allows them to make free-form requests.

1.1.3 Personal Software Agents

Agent research community has been actively producing methodologies [dSdL07, CGGS07] and development platforms [BBCP05] to create software agents that deliver services to end-users. Of these products, personal software agents are closely related to our work.

Researchers have described implementations of personal agents that operate on behalf of their owners to do useful tasks. [Shi06] describes Softgent, a personal software agent, that provides its user with an interface to email messages or to query databases. Users can, for instance, make natural language queries that must be grammatical to a database through the agent interface. The agent parses and submits the query to the database. This radically differs from our approach in that our approach supports free-form requests, not necessarily in terms of full grammatical sentences. Further, Softgent lacks the constraint recognition capabilities of our approach.

The authors of [CPC+04] describe an implementation of personal agents for assisting people in preparing the physical facilities of a meeting room. For each person attending a meeting, there is a personal agent that takes the person's model of preferences, such as adjustments of the speaker's sound and the level of lights, and attempts to prepare the meeting room in an optimal fashion. [WCRS04] describes personal agents that allow a user to obtain information about the entertainment facilities available in a location. Users communicate with the system through forms and provide their profiles and preferences (e.g. what movie they want to see and at what time). The system then provides responses based on user's preferences. [PSS02] describes an implementation for a conference attendee's personal assistant agent that brings scheduling information about conference schedules to its user's calendar. [PKC+01] describes an implementation of ITTalk, an integrated application using agent mediated services to disseminate event announcement. Users can provide their profiles, encoded in DAML, about presentations in which they are interested and about which they would like to be notified. The profiles can be provided either by filling in a web form or by providing URLs that link to such profiles. Our approach differs from these approaches in that while these agents in these approaches are preprogrammed to do prespecified tasks, our system is more general in the sense that the tasks are defined in terms of domain knowledge rather than preprogrammed.

1.2 Thesis Statement

It is possible to automate everyday service requests whose invocation results in establishing agreed-upon relationships in the context of a domain ontology. Examples of these services include scheduling appointments, buying and selling products, renting apartments and cars, and so forth. These systems use task ontologies as their foundational knowledge to identify users' needs and meet these needs. The system's behavior is limited only by the richness of task ontologies, which can be independently enriched by system specialists.

Our approach is a significant advancement over current approaches for the following reasons.

1. The approach does not impose any programming paradigm to specify tasks.
2. Unlike other approaches, there is no set of prespecified tasks from which users choose and there is no notion of composing task sequences, of which the user is aware.
3. The system dynamically determines the required processes to accomplish a task and dynamically generates software capable of performing the task.

1.3 Dissertation Contributions

As the number of services on the web continues to increase, users will increasingly suffer from issues such as finding and using appropriate services. This dissertation uniquely addresses these issues and makes the following contributions.

1. **Specification-based service invocation**. End-users can invoke services by only specifying their needs rather than having to find services and use them.
2. **Free-form service specification**. End-users can specify service requests using free-form, natural-language-like specifications.
3. **Service execution**. Our approach supports the execution of service requests from beginning to end with minimal required user intervention.
4. **Knowledge-level service creation**. The ontological basis of our approach renders it well-suited for creating semantic-web services. In order for service providers to provide

services in a new domain, it suffices to create a domain ontology—the cornerstone of the semantic web; no coding is required.

5. **Web-Principled Services**. We present a new vision for web services based more fundamentally on web principles. These web-principled services have two important characteristics. They decouple services and requesters, and they resolve data heterogeneity. The contribution here is threefold. First, we show that our ontology-based services are web-principled services. Second, we propose a request-oriented architecture for our vision for web-principled services that allows people to invoke these web-principled services by only specifying service requests using free-form specifications without having to discover them. Third, we show that the ontology-based service approach can bring web principles to traditional web services.

6. **Working prototype**. A fully implemented, working prototype system exists, which allows a user to specify service requests and which completely services these requests if there is an appropriate domain ontology for the request and appropriate domain data exists.

1.4 Dissertation Overview

Chapters 2, 3, 4, and 5 consist of published papers or papers accepted for publication. Thus, each chapter has its own abstract, introduction, contributions, experimental results, conclusions, and references. Chapter 6 is the usability study for our prototype, and Chapter 7 gives our conclusions and directions for future work.

Chapter 2 [AMEL05] presents a detailed overview of our system. It describes the components of a task ontology; a domain ontology and the process ontology. It also describes all the processing steps: a service request specification, task recognition, task execution including interacting with users to complete the service request if needed, and accomplishing the request. Further, the paper emphasizes and justifies, whenever appropriate, the domain-independence of our processing algorithms.

Chapter 3 [AME07] presents our ontological approach for recognition of constraints in free-form, natural-language-like service request specifications and the transformation of

these service request specifications to machine processable representations. In our ontology-based approach, a domain ontology encodes information such as applicable object sets, potential constraints over these object sets, and recognizers for instances of these object sets and constraints. The system recognizes the constraints in a service request by a two-fold, domain-independent process. (1) It matches a free-form service request against a collection of ontologies that belong to different domains to find the ontology that matches best. (2) It then selects from the given and implied constraints in the matched ontology those that are relevant to the service request to generate the constraints.

Chapter 4 [AME06] presents our approach to resolve under-constrained, and over-constrained systems of constraints. The paper offers an expectation-based process for eliciting additional constraints for under-constrained requests and for suggesting some constraints for users to relax for over-constrained requests. Further, the paper offers an ordering over solutions and an ordering over near solutions, and a selection mechanism based on Pareto optimality, to choose the best-m, with dominated solutions or dominated near solutions discarded.

Chapter 5 [AMELT07] presents our vision for web-principled services. Web-principled services are based on the more fundamental resource publication and access principles of the web. They enable decoupling among communicating applications, but, as a result, require heterogeneity resolution. In this paper, we show that ontology-based services discussed in Chapters 2 and 3 actually are web-principled services. We propose a request-oriented architecture that allows requesters to only specify their requests without requiring them to discover and reference any services capable of servicing their requests. We show that this architecture decouples services and requesters in the sense that requesters do not have to discover services. Additionally, the paper presents a way to turn a traditional web service into a web-principled service using the ontology-based service approach. As we will see, it suffices to describe a traditional web service using an ontology. Interestingly, turning a traditional web service into a web-principled service requires no changes to the actual interface and the internal implementation of the traditional web service.

Chapter 6 presents a user-oriented usability study for the fully working prototype system built on the concepts discussed in Chapters 2, 3, and 4. First, this chapter presents a

usage scenario of our system. This usage scenario highlights the interesting functionalities that we want users to evaluate. Second, we show how we elicit the data necessary for evaluating the usability of our prototype, present this data, and analyze it. We conclude the chapter by drawing conclusions about the usability of the prototype based on the data elicited from users.

Chapter 7 presents our concluding remarks and possible directions for future work.

Chapter 2

Conceptual Model Based Semantic Web Services

Abstract

To achieve the dream of the semantic web, it must be possible for ordinary users to invoke services. Exactly how to turn this dream into reality is a challenging opportunity and an interesting research problem. It is clear that users need simple-to-invoke-and-use services. This paper shows that an approach strongly based on conceptual modeling can meet this challenge for a particular type of service—those that involve establishing an agreed-upon relationship, such as making an appointment, setting up a meeting, selling and purchasing products, establishing employee job assignments, and many more. For these services, users can specify their requests as free-form text and then interact with the system in a simple way to complete the specification of a service request, if necessary, and invoke the service. Behind the scenes, the system uses a conceptual-model-based information extraction ontology to (1) recognize the request and match it with an appropriate ontology, (2) discover and obtain missing information, and (3) establish agreed-upon, conceptual-model-constrained relationships with respect to the desired service. The paper lays out our vision for this type of semantic web service, gives the status of our prototype implementation, and explains how and why it works.

Keywords: Services, semantic web services, service specification, service invocation, conceptual-model-based services.

2.1 Introduction

In open and ever-growing environments such as the world wide web, the amount of information and the number of services becoming available makes performing tasks such as finding information and finding and invoking services of interest quite challenging for web users. The semantic web along with personal software agents and web service systems purport to offer a solution to this challenge. But exactly how this solution will play out is still unclear.

In this paper we offer a unique approach to turn the vision of semantic web pioneers (e.g. [BLHL01]) into reality for everyday tasks such as scheduling appointments, selling, buying, making job assignments, and so forth. Our approach to this challenge centers around a *task ontology*. A task ontology can be thought of as having two component ontologies: (1) a *domain ontology* that defines concepts in a domain of a task along with relationships among these concepts and (2) a *process ontology* that defines generic processes for doing tasks. With a task ontology in hand, we address the following fundamental problems.

1. *Task Specification.* The first key issue to address is how to allow users to request services. We intend to let users assume the existence of an intelligent agent within the system and specify their service requests textually in any way they wish.

2. *Task Recognition.* Given a service request, our system attempts to recognize the specified task. This recognition process extracts information from the service request and matches it against known domain ontologies to find the proper task ontology for the service request.

3. *Task Execution.* Given a recognized task ontology and the information extracted from a service request, the system specializes the process ontology within the recognized task ontology to perform the service. The specialized process ontology has the ability to gather the information it needs by interacting with the system or the user or both to obtain missing required information. The specialized process ontology also has the ability to recognize specified constraints and determine whether they are satisfied. There may be a need to negotiate with users to relax task constraints when it is

apparent that it is not possible to complete the task given the current constraints. Finally, the specialized process ontology has the ability to establish the necessary relationships to complete the service request.

Our contribution in this paper is to show that it is possible to build a system to automate everyday tasks, such as scheduling appointments, buying and selling, and making job assignments, whose invocation results in establishing agreed-upon relationships in a domain ontology. Within this scope of services, the system's behavior is limited only by the richness of task ontologies, which can be independently enriched by system specialists. Our approach contributes to the vision of the semantic web in the sense that it offers the following significant advantages. (1) The system allows for free-form task specification, and thus, it does not impose any programming paradigm to specify tasks, nor does it have a set of pre-specified tasks from which a user can choose. (2) The system reasons about the request based on a task ontology and synergistically gathers the information it needs to generate software capable of performing the task.

We present our contributions as follows. We begin in Section 2.2 by placing our work in the context of other work on web services. We then present our vision for task-ontology-based services in three major parts: task specification, which allows users to textually specify tasks (Section 2.3); task recognition, which finds the domain of a specified task and specializes processes to perform the task (Section 2.4); and task execution (Section 2.5). We give the status of our prototype implementation along with the future work we are considering in Section 2.6, and we conclude with Section 2.7.

2.2 Related Work

To the best of our knowledge our work is unique. We know of no research trying to satisfy free-form user service requests by establishing needed relationships within the context of the constraints imposed by an ontology. We do, however, wish to place our research within the framework of the current work on web services.

Service-oriented architecture concepts have constituted "best practices" in enterprise-scale application development for the past three decades [GBR05]. The key idea

of service-oriented architecture is that business functionality is packaged as reusable services that can be invoked through standard interfaces. Recently, the major standards-based approach to services has centered on web services as described by W3C's Web Services Description Language (WSDL). With XML as the common underlying message format, WSDL defines a grammar for specifying services, and it is surrounded by a constellation of related standards including the SOAP communication protocol, the UDDI service registry mechanism, and other proposals within W3C's Web Services Activity [WS05]. These technologies comprise the common foundation for current web services research.

However, this foundation is only an initial infrastructure, and advanced applications of web services require additional layers. WSDL, much like HTTP, is essentially a stateless protocol and thus requires additional architectural support for service discovery, composition, and transactional execution. For example, the Web Service Modeling Framework (WSMF) was designed to address decoupling and mediation needs [FB02]. The Internet Reasoning Service defines a framework for automatically publishing, locating, composing, and executing heterogeneous services [MDCG03]. BPEL4WS, promoted by the OASIS group, recognizes the need for higher levels of web-services "choreography" including, among other elements, support for transactional features not found in WSDL [ACD+03]. Research efforts related to semantic web services [MSZ01] often follow the DAML-S/OWL-S line of work [FB02, MA02, KA04]. The DAML Services Coalition observed early on that WSDL does not account for workflow needs, and so to support semantic web services they created what is now called OWL-S [OWL04].

Web services are characterized by many layers, and our approach builds on the existing web services stack in a complementary fashion. Our approach shows how ontologies can be used to provide semantic web services in a user-friendly, highly automated way. We are not the first to use task and process ontologies to manage web services, but we do provide a unique approach that leverages the principles of ontology-based data extraction to hide complexity from the end user.

2.3 Task Specification

We explain task specification using an example. A typical usage of our approach is to schedule appointments. We use a somewhat simplified version of the example described by Berners-Lee, et al., in their vision paper, "The Semantic Web" [BLHL01]. In our example, a user of the semantic web wants to schedule an appointment with a service provider—a dermatologist. The user does not have any particular dermatologist in mind, but wants one that meets some constraints regarding appointment time, date, the location of the service provider, and the type of insurance the service provider accepts.

To use our approach to accomplish this task, the user first specifies the task by "simply" stating what needs to be done. Suppose the user states the following.

> I want to see a dermatologist next week; any day would be OK for me, at 4:00. The dermatologist should be within 5 miles from my home and must accept my insurance.

Before this statement is made, our proposed system has no clue regarding the domain of the task nor any clue regarding how it can be done. Therefore, this specification needs to go through a task recognition step, which we discuss next.

2.4 Task Recognition

The objective of task recognition is to determine the domain of a specified task. Our approach uses a task ontology to meet this objective. Therefore, we first introduce the two components of a task ontology, namely a *domain ontology* and a *process ontology*, respectively introduced in Sections 2.4.1 and 2.4.2. In Section 2.4.3, we describe how our approach determines which task ontology to use, among the many we assume exist.[1]

2.4.1 Domain Ontology

A *domain ontology* specifies named sets of objects, which we call *object sets* or *concepts*, and named sets of relationships among object sets, which we call *relationship sets*. Figure 2.1

[1]These task ontologies can be for different applications, or can even be competing task ontologies for the same application.

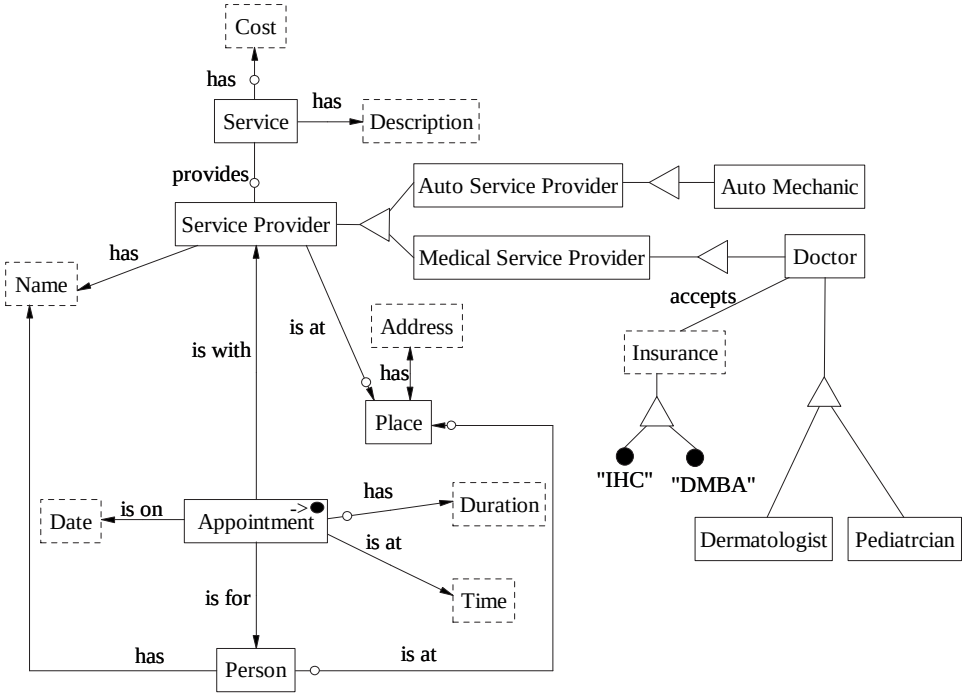


Figure 2.1: A conceptual-model representation of a domain ontology for appointments (partial).

shows a small part of a conceptual model representation of a domain ontology for scheduling an appointment.[2] The domain ontology consists of concepts such as *Date*, *Time*, and *Service Provider* that can be used to schedule appointments with service providers such as doctors and auto mechanics. The conceptual model has two types of concepts, namely lexical concepts (enclosed in dashed rectangles) and nonlexical concepts (enclosed in solid rectangles); it also provides for explicit concept instances (denoted as large black dots). A concept is *lexical* if its instances are indistinguishable from their representations. *Time* is an example of a lexical concept because its instances (e.g. "10:00 a.m." and "2:00 p.m.") represent themselves. A concept is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Dermatologist* is an example of a nonlexical concept because its instances are identifiers such as, say, "Dermatologist$_1$", which represents a particular person in the real world who is a dermatologist. We designate the main concept in a

[2]In practice, we would need a much larger and richer ontology for service providers. We have limited our ontology to those concepts needed in our running example, plus a few more to explicitly illustrate concepts not needed for our sample task. We indicate by having *Auto Mechanic* in our example, for instance, that there are many more types of service providers.

domain ontology by marking it with "–> •" in the upper right corner.[3] We designate the concept *Appointment* in Figure 2.1 as the main concept because this domain ontology is for making appointments. Figure 2.1 also shows relationship sets among concepts, represented by connecting lines, such as *Appointment is on Date*. The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *Service Provider has Name* is functional from *Service Provider* to *Name* (i.e. a service provider has only one name), and *Service Provider provides Service* is many-many (i.e. a service provider can provide many services and a service can be provided by many service providers). A small circle near the connection between an object set O and a relationship set R represents optional, so that an instance of O need not participate in a relationship in R. For example, the circle on the *Appointment* side of the relationship set *Appointment has Duration* states that an instance of *Appointment* may or may not relate to an instance of *Duration* (i.e. there need not be a specified duration for an appointment). A triangle in Figure 2.1 defines a generalization/specialization with a generalization connected to the apex of the triangle and a specialization connected to its base. For example, *Dermatologist* is a specialization of *Doctor*.

The concepts in our domain ontology are augmented with data frames [Emb80]. A *data frame* describes the information about a concept. We capture the information about a concept in terms of its external and internal representation, its contextual keywords or phrases that may indicate the presence of an instance of the concept, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the concept along with contextual keywords or phrases that indicate the applicability of an operation. Figure 2.2 shows sample (partial) data frames for the concepts *Time*, *Date*, *Address*, *Distance*, *Dermatologist*, and *Appointment*. The *Time* data frame, for example, captures instances of this concept that end with "AM" or "PM" (e.g. "2:00 PM" and "2:00 p.m"). As Figure 2.2 shows, we use regular expressions to capture external representations. A data frame's context keywords/phrases are also regular expressions (often simple lists of keywords/phrases separated with "|"). For example, the

[3]This notation denotes that when this ontology is used to create an appointment, the object set *Appointment* becomes ("–>") an object instance ("•").

Time
…
textual representation: ([2-9]|1[012]?):([0-5]\d)[aApP]\.?[mM]\.? | …
…
end

Date
…
NextWeek(d: Date)
returns (Boolean)
context keywords/phrases: next week |
week from now | …
end

Tomorrow (s: String)
returns (Date)
context keywords/phrases: tomorrow | next day | …
…
end

Address
…
DistanceBetween (a1: Address, a2: Address)
returns (Distance)
…
end

Distance
internal representation : real
textual representation: ((\d+(\.\d+)?)|(\.\d+))
context keywords/phrases: miles | mile | mi | kilometers | kilometer | meters | meter | …
…
LessThan(d1: Distance, d2: Distance)
returns (Boolean)
context keywords/phrases: less than | < | …
…
end

LessThanOrEqual(d1: Distance, d2: Distance)
returns (Boolean)
context keywords/phrases: within | not more than |
≤ | …
…
end

Dermatologist
internal representation: object id
…
context keywords/phrases: [Dd]ermatologist | skin
doctor | …
…
end

Appointment
internal representation: object id
…
context keywords/phrases: appointment |
want to see a[n]? | …
…
end

Figure 2.2: Some sample data frames (partial). (The "..." in the textual-representation part of the *Time* data frame in Figure 2.2 indicates that there are other representations of *Time* such as military time. In general the presence of ellipses show that there are many incomplete components of our data frames.)

Distance data frame in Figure 2.2 includes context keywords such as "miles" or "kilometers". In the context of one of these keywords, if a number appears, it is likely that this number is a distance. The operations of a data frame can manipulate a concept's instances. For example, the *Distance* data frame includes the operation *LessThan* that takes two instances of *Distance* and returns a Boolean. The context keywords/phrases of an operation indicate an operation's applicability, for example, context keywords/phrases such as "less than" and

"<" apply to the *LessThan* operation. A nonlexical concept such as *Dermatologist* often only has context keywords or phrases. Figure 2.2 shows that the *Dermatologist* data frame includes a regular expression which includes words and phrases that could indicate the presence of the concept of a dermatologist.

2.4.2 Process Ontology

A *process ontology* describes an execution pattern in a domain. Figure 2.3 shows our process ontology as specialized for scheduling appointments. We represent process ontologies and specialized process ontologies as statenets [EKW92], a representation that lets us specify standard Event-Condition-Action (ECA) rules [WC95, PPW03]. The statenet in Figure 2.3 is "specialized" from the general pattern in the sense that (1) all but the two final actions (schedule and do not schedule) are parameterized by the domain ontology but otherwise fixed over all services for which the system operates; and (2) given the domain ontology, the system can fully generate these two final actions. Thus, any specialized process ontology depends only on the domain ontology. Significantly (and somewhat surprisingly), this means that system developers need never write code for services of the type our system handles; specifying domain ontologies is sufficient to fully specify the services.

In this section, we describe the ECA rules used to construct a process ontology and the execution pattern for the process ontology. We leave the details of the subprocesses on which the process ontology depends to be discussed in Section 2.5. As we shall see, all of these subprocesses are domain-independent. Domain-independence is what makes it possible to automatically generate specialized process ontologies without having to write any code.

A process ontology consists of states, represented as rounded rectangles (e.g. *ready* and *initial task-view ready* in Figure 2.3), and transitions, represented as divided rectangles (e.g. the *@create/initialize* transition in Figure 2.3). In the top part of a transition, we specify triggers, which are events or conditions or both. Events are prefixed by "@" (read "at"); examples include *@process-ontology(domain-ontology)* and *@task-view complete*, where the former is a parameterized event that triggers the transition when the event occurs (is called from some other process), and the latter is a non-parameterized event that triggers the

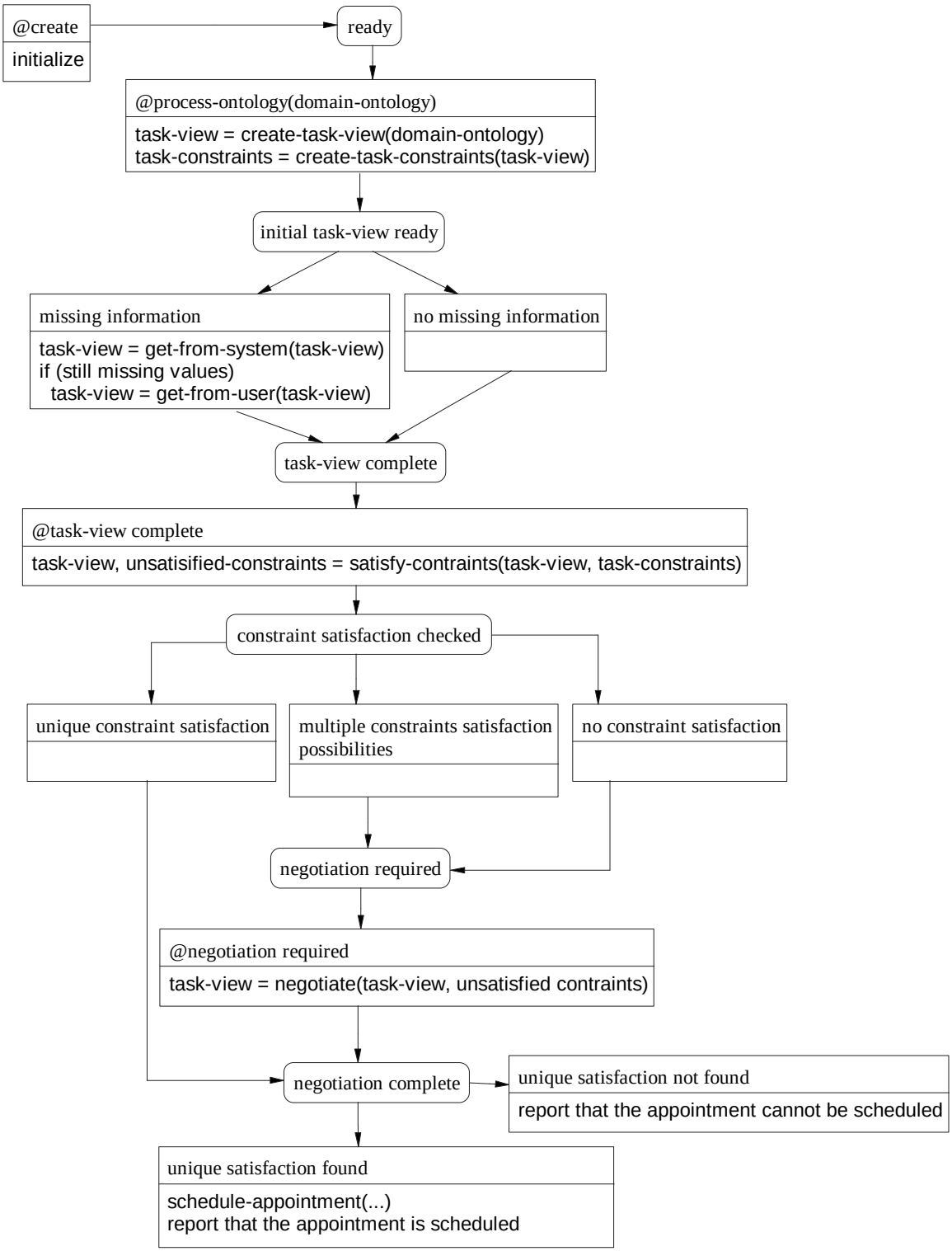


Figure 2.3: A process ontology specialized for scheduling appointments.

transition when the task view is complete. Actions appear in the bottom part of divided rectangles. The actions in a particular transition execute when the trigger of the transition fires. Examples include *create-task-view(domain-ontology)* and *get-from-system(task-view)*, which both invoke subprocesses in our system.

The general flow of the process ontology is as follows. Once triggered and given a domain ontology, the process ontology invokes the subprocess *create-task-view(domain-ontology)* to create a *task-view*, which is the part of a domain ontology that matches with the user-specified task, and then invokes the subprocess *create-task-constraints(task-view)* to find and list the applicable constraints for the task. If all concepts in the task view that are required to have values have already obtained their values from the user-given task specification, the process ontology enters the *task-view complete* state; otherwise the process ontology obtains values for these concepts from system repositories using the subprocess *get-from-system(task-view)* and obtains values from the user it cannot obtain from system repositories using the subprocess *get-from-user(task-view)*. Next, the process ontology checks for constraint satisfaction using the process *satisfy-constraints(task-view, task-constraints)* and enters the *constraint satisfaction checked* state. If constraint satisfaction is unique (exactly one set of values satisfies the constraints), no negotiation is necessary, so the process enters the *negotiation complete* state; otherwise if there are multiple sets of values that satisfy the constraints or if there are no sets of values that satisfy the constraints, the process ontology enters the *negotiation required* state. During the negotiation phase, the system and user work together in an attempt to find a unique solution. If a unique solution is found, the process ontology schedules the appointment; otherwise the process ontology reports that the appointment cannot be scheduled.

2.4.3 Task Ontology Recognition

The task ontology recognition process selects from among potentially many domain ontologies deployed on the semantic web the (correct) domain ontology for a task specification. The recognition process takes the set of available domain ontologies and a task specification as input, and returns the domain ontology that best matches with the task specification as output. The recognition process works in two steps. First, for each domain ontology, the

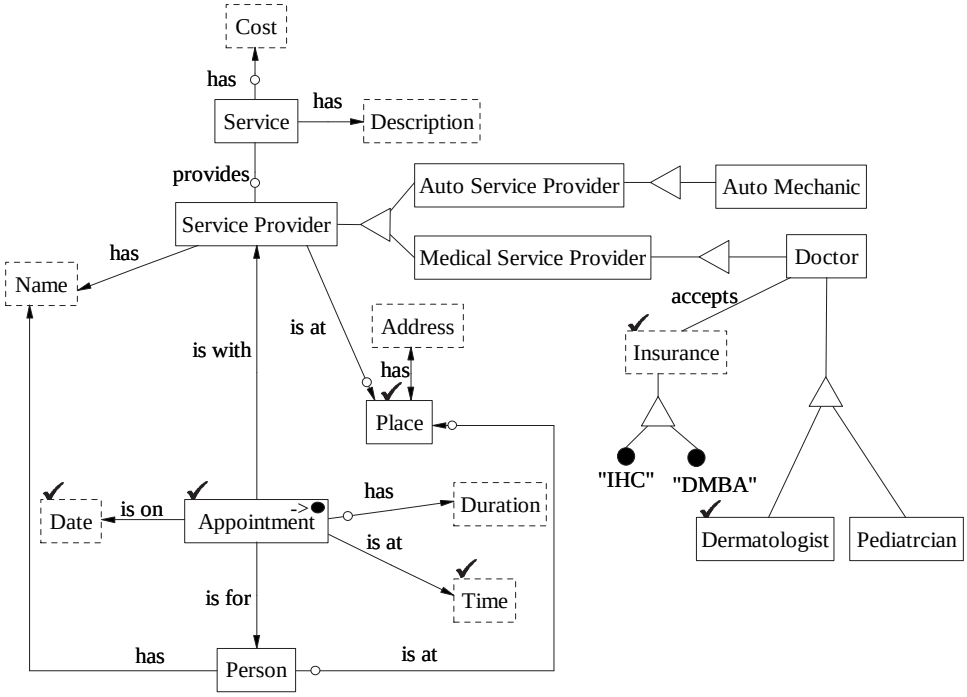


Figure 2.4: The output of the recognition-process.

recognition process applies concept recognizers in the data frames to the task specification and marks every concept that matches a substring in the task specification. Second, the process computes a rank value for each domain ontology with respect to the task specification and then selects the domain ontology with the highest rank value.

Referring to our running example, when the recognition process executes for the domain ontology in Figure 2.1, the data frames in Figure 2.2, and the task specification in Section 2.3, it produces the output in Figure 2.4. The concept recognizer in the data frame for *Dermatologist* recognizes the constant value "dermatologist" in the task specification, and therefore the concept *Dermatologist* is marked ($\surd$). Likewise, a recognizer in the *NextWeek* operation in the *Date* data frame recognizes "next week"; in the *Time* data frame recognizes the constant value "4:00"; in the *Appointment* data frame recognizes "want to see a"; in the *Place* data frame recognizes "my home"; and would, in the *Insurance* data frame, recognize "insurance"; and therefore these concepts are marked. The recognized substrings cover a large part of the task specification; for our running example, we assume

that no other ontology covers the task specification as well and therefore that the system selects the *Appointment* task ontology.[4]

2.5 Task Execution

The process ontology is responsible for executing tasks. As mentioned in Section 2.4.2, the process ontology invokes subprocesses that either execute the same for all domain ontologies or can be automatically generated from any given domain ontology. In this section we discuss these subprocesses and justify our claim that all code needed to execute any service can be fixed in advance or automatically generated.

2.5.1 Task View Creation

Task view creation takes a marked domain ontology as input and produces a task view as output. Although not quite so simple because spurious object sets may be marked, the process basically operates on its input as follows. It keeps the main concept of the domain ontology (the concept marked with "–> •"), all marked concepts, and all concepts that mandatorily depend on the main concept. It prunes away all other concepts along with all their relationships as well as any marked concepts considered to be spurious because they conflict with other marked concepts in the sense that constraints of the ontology do not allow both, and these other marked concepts rank higher in applicability to the task specification. In addition, the process replaces generalization concepts by marked specializations, if any, and replaces non-lexical concepts by lexical concepts when there is a one-to-one correspondence. The derived sub-ontology, consisting of the concepts and relationships among the concepts that remain, is called the *task view*. Observe that this process is domain independent—it operates identically for any defined domain ontology.

Referring to our running example, the resulting task view is in Figure 2.5. The process does not prune *Appointment* because it is the main concept. It does not prune

[4]It is interesting to think about possible errors in the selection process. The task specification should establish enough context for the system to select the right ontology. If the system selects the wrong task ontology, it will respond in terms foreign to what the user expects, just like humans sometimes do. We can correct the system by providing more or better context. As another possibility, the system may have no task ontology for the service being requested; the system should be able to recognize this possibility and respond by saying the service cannot be provided.

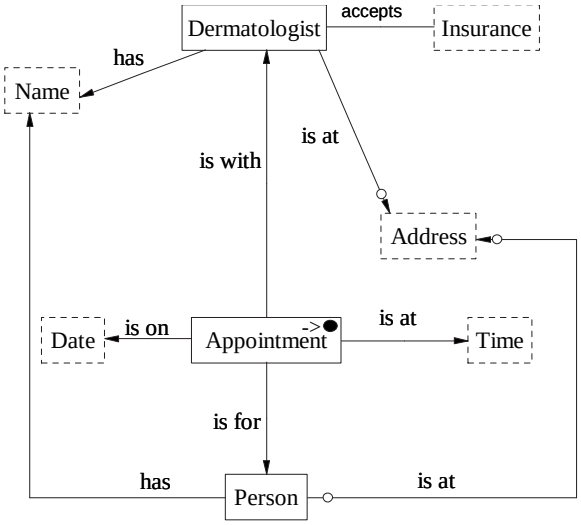


Figure 2.5: The task view for the specified task in Section 2.3.

Date, *Time*, *Place*, *Insurance*, and *Dermatologist* because they are marked, and it does not prune *Person*, *Name*, and *Service Provider* because they mandatorily depend on the main concept. Finally, the marked specialization *Dermatologist* replaces its generalization *Service Provider*, and the lexical concept *Address* replaces the non-lexical concept *Place* with which it has a one-to-one correspondence.

2.5.2 Task Constraint Creation

Given the task view and the operations in the data frames associated with the concepts of a task view, the system generates any additional constraints imposed on a task beyond those that are already part of the conceptual-modeling constraints of the task view. It then combines them with the constraints in the task view to produce the full set of constraints. The result is a formal statement in terms of predicate calculus that must be satisfied in order to service the request.

Task constraint creation operates as follows.

1. *Get the Boolean operations implied by the task specification.* The task-constraint-creation process finds all operations in the data frames whose recognizers match substrings in the task specification and whose return types are Boolean. In our running example, the process finds the operator *NextWeek(d: Date)* because, as Figure 2.2

shows, it is Boolean and one of its context phrases is "next week", which appears in the task specification in Section 2.3. Similarly, the process finds *LessThanOrEqual(d1: Distance, d2: Distance)* based on recognizing "within", "5", and "miles" and recognizes *Equal(i1: Insurance, i2: Insurance)* based on recognizing "insurance".

2. *Get constant values from the task specification that can serve as parameters of the Boolean operations.* The data frames recognize and extract "5" as a *Distance* and "4:00" as a *Time*. In our running example, we thus obtain *LessThanOrEqual(d1: Distance, "5")* and *Time("4:00")*.[5]

3. *Get operations that depend on the task view and can provide values for parameters of the Boolean operations.* For each Boolean operation, the process considers the type(s) of the input parameter(s). If one or more input parameters has a type that does not match a concept in the task view, the process tries to find an operation in the data frames whose input parameter types are concepts in the task view and whose return type matches the type of the input parameter; if successful, it replaces the input parameter with this operation. Referring to our example, *LessThanOrEqual(d1: Distance, "5")* has the input *d1* of type *Distance*, which does not appear in the task view. Since, however, *Address* does appear in the task view and the operation *DistanceBetween(a1: Address, a2: Address)* returns a *Distance*, the task-constraint-creation process can do a substitution, yielding *LessThanOrEqual(DistanceBetween(a1: Address, a2: Address), "5")*.

4. *Get sources, within the task view, for values of Boolean operations.* To determine the source of values for the input parameters of the Boolean operations, the process makes use of the relationships in the task view. For example, the operation *DistanceBetween(a1: Address, a2: Address)* has two input parameters of type *Address*. According to the relationships between the concepts in the task view, *Address* is related to both *Dermatologist* and *Person*. The process can therefore infer that the value of one of the address parameters comes from a relationship in *Dermatologist is at Address* and value of the other comes from a relationship in *Person is at Address*.

[5]Note that *Time(x)* is a one-place predicate. Every object set in a domain ontology is a one-place predicate, and every n-ary relationship set is an n place predicate.

$$
\begin{aligned}
&Appointment(x_0)\, is\, with\, Dermatologist(x_1) \wedge Appointment(x_0)\, is\, for\, Person(x_2)\\
&\wedge Appointment(x_0)\, is\, on\, Date(x_3) \wedge Appointment(x_0)\, is\, at\, Time(\text{“4:00”})\\
&\wedge Dermatologist(x_1)\, has\, Name(x_4) \wedge Dermatologist(x_1)\, is\, at\, Address(x_5)\\
&\wedge \exists x(Dermatologist(x_1)\, accepts\, Insurance(x) \wedge Insurance(x_6) \wedge Equal(x, x_6))\\
&\wedge Person(x_2)\, has\, Name(x_7)\ \wedge Person(x_2)\, is\, at\, Address(x_8)\\
&\wedge ...\\
&\wedge NextWeek(x_3) \wedge LessThanOrEqual(DistanceBetween(x_5, x_8),\ \text{“5”})\\
&\wedge ...\\
&\wedge \forall x \forall y(Person(x)\, is\, at\, Address(y) \Rightarrow Person(x) \wedge Address(y))\\
&\wedge \forall x(Person(x) \Rightarrow \exists^{\leq 1} y(Person(x)\, is\, at\, Address(y)))\\
&\wedge ...
\end{aligned}
$$

Figure 2.6: Generated predicate calculus statement. (The statement is partial—it omits several more referential integrity constraints, several more participation constraints, and all unnecessary unary predicates for individual object sets.)

The process leaves any input parameter that it cannot determine as a free variable. Because *Insurance* is related only to *Dermatologist*, the process determines that the source of the value of one input parameter comes from a relationship in *Dermatologist accepts Insurance* and leaves the other as a free variable. Also, since the relationship set *Dermatologist accepts Insurance* is many-many, the process binds the parameter *i2* with the existential quantifier to declare that any one value of *i2* that satisfies the generated predicate calculus statement $\exists i_2(Dermatologist(x) accepts Insurance(i2) \wedge Equal(i1, i2)) \wedge Insurance(i1) \wedge Insurance(i2))$ is enough.

Figure 2.6 shows the resulting predicate calculus statement.[6] Our objective, as we continue, will be to provide values for free variables such that there is one and only one appointment (i.e. one and only one value for the non-lexical argument x_0 in Figure 2.6). Before continuing, however, observe that the process that generates the predicate calculus statement is domain independent because its algorithms are the same for all domains. The process makes use of only the information provided by the task view and the associated data frames. Once these are available, the process can discover constraints and produce a predicate calculus statement using fixed algorithms that work for all domains.

[6]For those acquainted with description logics [BN03], we point out that this expression can be converted into the language $ALCN$ and its satisfiability is thus decidable and its complexity is *ExpTime-complete*. With regard to complexity, we show in the next section that for our special case, we can reduce the execution to a straightforward select-project-join query over a standard relational database to obtain the result we need. We conjecture, and will prove as part of our future work, that all predicated-calculus expressions generated from any domain ontology by the process we have described here will be decidable and, for the results we need, will reduce to a straightforward relational-database query.

$\{< x_1, x_3, x_4, x_5, x > |$
$\quad Appointment\,is\,with\,Dermatologist(x_1)\,and\,is\,on\,Date(x_3)\,and\,is\,at\,Time(\text{“}4{:}00\text{”})$
$\wedge\, Dermatologist(x_1)\,has\,Name(x_4) \wedge Dermatologist(x_1)\,is\,at\,Address(x_5)$
$\wedge\, Dermatologist(x_1)\,accepts\,Insurance(x)\}$

Figure 2.7: Generated predicate calculus statement.

2.5.3 Obtaining Information from the System

Given the predicate calculus statement in Figure 2.6, the system can generate a query for the system's appointment databases.[7] Assuming that the appointment database has a view definition that corresponds with its ontology, the generation of a relational calculus query is straightforward. We simply cut the predicate calculus statement down to include only those relationship sets that appear in the database's ontological view, and in one case, namely for the primary object set, we combine the relationship sets from the database's ontological view that are connected to the primary object set. For our running example, the generated predicate calculus query is in Figure 2.7. In Figure 2.7 the *Appointment is with Dermatologist and is on Date and is at Time* is the relationship set obtained by combining *Appointment is with Dermatologist*, *Appointment is on Date*, and *Appointment is at Time*; note that we do not also combine *Appointment is for Person* because this relationship set is not part of the stored appointment database for making appointments—only available appointment dates and times are in the database. Observe that given a predicate calculus statement generated by the task-constraint-creation process and the ontological view of the selected task ontology, the system can always generate this query; thus, this query generation process is domain independent.

Execution of the generated query returns a set of partially filled-in interpretations. For our running example, we show two partial interpretations in Figure 2.8, which we assume are the only partial interpretations returned as a result of executing the relational calculus query in in Figure 2.7. The meaning of the first of these interpretations is that $Dermatologist_0$, who is *Dr. Carter* has an appointment available on *5 Jan 05*, which is "next week" with respect to our assumed execution date, *30 Dec 04*. The meaning of the second is similar, but is for $Dermatologist_1$ rather than $Dermatologist_0$.

[7] We assume that the system has databases that store real-world instances of concepts of all domain ontologies known to the system.

$Interpretation_1$:
$Appointment(x_0)\, is\, with\, Dermatologist(Dermatologist_0)$
$\quad \wedge\ Appointment(x_0)\, is\, for\, Person(x_2)$
$\wedge\ Appointment(x_0)\, is\, on\, Date(\text{“}5\, Jan\, 05\text{”}) \wedge\ Appointment(x_0)\, is\, at\, Time(\text{“}4:00\text{”})$
$\wedge\ Dermatologist(Dermatologist_0)\, has\, Name(\text{“}Dr.\, Carter\text{”})$
$\wedge\ Dermatologist(Dermatologist_0)\, is\, at\, Address(\text{“}600\, State\, St.,\, Orem\text{”})$
$\wedge\ (Dermatologist(Dermatologist_0)\, accepts\, Insurance(\text{“}IHC\text{”})$
$\qquad \wedge\ Insurance(x_6) \wedge Equal(\text{“}IHC\text{”}, x_6)$
$\quad \vee\ Dermatologist(Dermatologist_0)\, accepts\, Insurance(\text{“}DMBA\text{”})$
$\qquad \wedge Insurance(x_6) \wedge Equal(\text{“}DMBA\text{”}, x_6))$
$\wedge\ Person(x_2)\, has\, Name(x_7) \wedge Person(x_2)\, is\, at\, Address(x_8)$
$\wedge\ ...$
$\wedge NextWeek(\text{“}5\, Jan\, 05\text{”}) \wedge LessThanOrEqual(DistanceBetween(\text{“}600\, State\, St.,$
$\quad Orem\text{”}, x_8),\ \text{“}5\text{”})$
$\wedge\ ...$

$Interpretation_2$:
$Appointment(x_0)\, is\, with\, Dermatologist(Dermatologist_1)$
$\quad \wedge\ Appointment(x_0)\, is\, for\, Person(x_2)$
$\wedge\ Appointment(x_0)\, is\, on\, Date(\text{“}6\, Jan\, 05\text{”}) \wedge\ Appointment(x_0)\, is\, at\, Time(\text{“}4:00\text{”})$
$\wedge\ Dermatologist(Dermatologist_1)\, has\, Name(\text{“}Dr.\, Peterson\text{”})$
$\wedge\ Dermatologist(Dermatologist_1)\, is\, at\, Address(\text{“}12\, Main\, St.,\, Lindon\text{”})$
$\wedge\ (Dermatologist(Dermatologist_1)\, accepts\, Insurance(\text{“}DMBA\text{”}) \wedge Insurance(x_6)$
$\quad \wedge\ Equal(\text{“}DMBA\text{”}, x_6))$
$\wedge\ Person(x_2)\, has\, Name(x_7) \wedge\ Person(x_2)\, is\, at\, Address(x_8)$
$\wedge\ ...$
$\wedge\ NextWeek(\text{“}6\, Jan\, 05\text{”}) \wedge LessThanOrEqual(DistanceBetween(\text{“}12\, Main\, St.,$
$\quad$ Lindon”, x_8), “5”)
$\wedge\ ...$

Figure 2.8: Partial interpretations.

2.5.4 Obtaining Information from a User

Observe that the remaining free variables are x_0, which is the object we are trying to establish; x_2, which is the person for whom the appointment is being made; x_6, which is the insurance in the request; x_7, which is the name of the person for whom the appointment is being made; and x_8, which is the address of the person for whom the appointment is being made. We can establish the variable x_0, which represents the appointment, if we can obtain the remaining variables, which, of course, are exactly the ones we need to obtain from the user.

When the system can recognize which which free variables need values (equivalently, which concepts need values), the process of obtaining values from the user is domain independent. Thus, if a lexical concept C requires a value from the user, the system can prompt

the user with the standard phrase, "What is the C?" For nonlexical concepts, the system can generate object identifiers, as needed. For our running example, the system would ask: (1) "What is the Insurance?", (2) "What is the Name?", and (3) "What is the Address?". These may well be understood in the context of the task being specified, but it is likely to be better if the system can ask questions in context by verbalizing[8] the context with respect to the primary concept. In this case, for (2) the system would say, "Appointment is for Person, and Person has Name. What is the Name?" and for (3) would say, "Appointment is for Person, and Person has Address. What is the Address?". There is no context for the insurance other than the context established in the statement of the task specification in Section 2.3, so for (1) the system would just say, "What is the Insurance?".

For our running example, we assume that the user responds by answering the three questions as: (1) "IHC", (2) "Lynn Jones", and (3) "300 State St., Provo". Observe that if the user had originally added the sentence, "The appointment is for my daughter, Lynn Jones; we live at 300 State St. in Provo; and my insurance is IHC." to the task specification in Section 2.3, then the system could have extracted this information, and could have executed without any need for asking the user for additional information.

2.5.5 Constraint Satisfaction and Negotiation

At this point in the process, the system has one free variable, namely the nonlexical object we are trying to establish (the *Appointment* for our example). If there is only one interpretation that satisfies all the constraints, we are ready establish the object and finalize the process. In our running example, since the insurance is "IHC", the second interpretation in Figure 2.8 cannot be satisfied because the only insurance $Dermatologist_1$ accepts is "DMBA." The first interpretation can be satisfied if $DistanceBetween($"600 *State St.*, *Orem*", "300 *State St.*, *Provo*"$)$ is less than 5 miles. In this case, the system can make the appointment for the user.

For the sake of further discussing constraint satisfaction and negotiation, we assume, however, that the distance between the addresses is 6 miles. In this case, no interpretation

[8] Verbalization according to [Hal04] and verbalization with respect to the model-equivalent language for OSM [LEW00] are examples of worked-out verbalizations that could be used in our application.

satisfies the constraints, and the system must either fail to make an appointment or find a way to relax one or more constraints. One way to negotiate would be to take a generated potential interpretation that does not satisfy the constraints, output the constraints that are not satisfied, and ask the user what to do. Note that this way of negotiating is fully independent of the domain, since the system is able to identify and list each constraint that is not satisfied. For our running example, the system would display the following (hopefully, sprinkled with a lot more syntactic sugar than exemplified here):

> The following constraint(s) are not satisfied:
> LessThanOrEqual(DistanceBetween("600 State St., Orem", "300 State St., Provo"), "5")
> where DistanceBetween("600 State St., Orem", "300 State St., Provo") = 6
> What do you wish to do?

Unfortunately, the system is now beginning a free-form conversation, which it is not in a position to handle. The system can, however, ask the user to edit the task specification, giving looser constraints and then reentering it, by adding, "Please respond by editing your task specification, giving constraints that can be satisfied." The user may, of course, not wish to edit the task specification, in which case, the system reports that it cannot make an appointment satisfying all the constraints.

To further discuss the possibilities, we next consider the system response if there are two or more interpretations that satisfy all the constraints. For our running example, if the user had specified "DMBA" as the insurance and "20" as the mileage extent, both $Interpretation_1$ and $Interpretation_2$ would have satisfied all the constraints. In this case, the system can respond by offering the two alternatives and allowing the system to select one. If there are many interpretations that satisfy all the constraints, we must provide a way to control the potential overload on the user. In the worst case, we can arbitrarily present any k possibilities, where k is small, and let the user select one. As part of our future work, we can likely find ways to have the system rank them, and then present the top-k to the user.

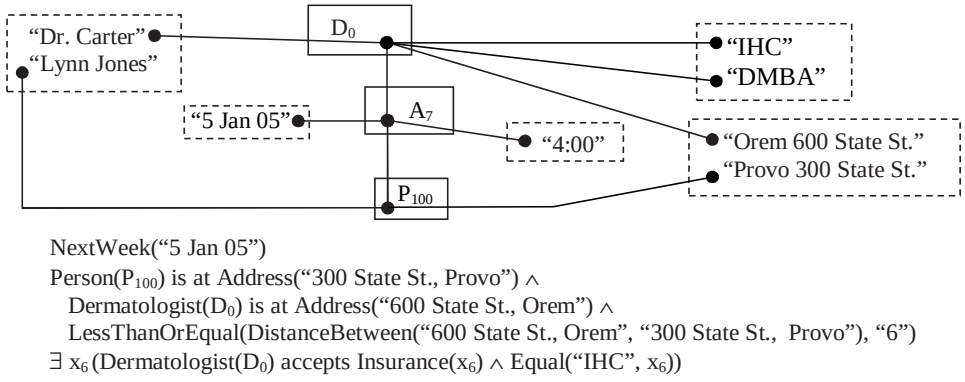


Figure 2.9: The scheduled appointment.

2.5.6 Process Finalization

At this point in the process, the system either has a single solution or has agreed with the user that there is no solution. Hence, it is straightforward to know what to do. What is interesting here is that although this code cannot be written in advance because it does depend on the domain ontology, it can be generated on-the-fly by code that can be written in advance.

In our running example, we assume that the user replaces "5 miles" with "6 miles," and thus that there is a unique solution. Hence, the process ontology schedules the appointment using the action *schedule-appointment("Lynn Jones", ...)*. Figure 2.9 shows the scheduled appointment. As shown, $Appointment_7$ is scheduled for $Person_{100}$ whose name is "*Lynn Jones*", with $Dermatologist_0$ whose name is "*Dr. Carter*" on date "*5 Jan 05*" at time "*4:00*" at address "*600 State St., Orem*". The process ontology notifies the user that the appointment is successfully scheduled.

The subprocess *schedule-appointment(...)* is domain dependent because it needs knowledge about what object sets should be filled in with which objects in order to schedule an appointment. However, given the ontology and the values for the free variables obtained from the unique interpretation created by this time during the execution of the process ontology, the system can use this knowledge to automatically generate code for this last part of the process. Observe that this holds for any domain so long as the objective is to insert an object into an object set of interest and then satisfy all applicable constraints.

Thus, since this is exactly the kind of service our system provides, it is always possible for the system to generate the finalization step for any domain ontology.

2.6 Prototype Implementation Status and Future Work

The success of our conceptual-model-based, semantic-web-services system will depend largely on its ability to successfully extract information from free-form text specifications of desired services. Our long-standing work on information-extraction ontologies [ECJ+99] provides the basis for this component of our system. Building on our work on information-extraction ontologies, we have implemented an initial end-to-end prototype that accepts free-form text specifications; finds an appropriate task ontology for the specification (if one exists); produces a task view for the specification; determines whether there is missing information; and, if not, establishes the primary object and thus performs the service. Our system does not yet obtain and use task-specified Boolean functions and other functions on which they depend; does not yet use its system database to obtain values for free variables, does not yet do negotiation. Adding these features is part of our current work.

As for future work, we expect to investigate more explicitly the boundaries of applicability. Should the system, for example, be required to handle more complex cases? The system, as currently envisioned, does not for instance, allow users to compose tasks nor to specify conditional tasks or iterative tasks. As currently proposed, a user can compose tasks only by specifying two successive tasks, e.g. make an appointment with a dermatologist and then make an appointment for a haircut. For conditional tasks, such as "If I can see Dr. Peterson within a week, make an appointment with Dr. Peterson; otherwise make an appointment with Dr. Carter.", the user would have to query the system to determine whether an appointment could be made with Dr. Peterson within a week and then, depending on the answer, either make an appointment with Dr. Peterson or Dr. Carter. Further, it is also not clear whether the system needs to handle complex task specifications that would require the system to use natural language processing or other techniques to disambiguate sentence structures or to resolve pronoun references. As the system now stands, users would have

to become used to its limited ability to actually understand.[9] Whether this is sufficient to be serviceable for the general public on a broad enough basis to be useful is not yet known, but there is reason to believe it is, and there is reason to believe that this approach will be more widely acceptable to ordinary users than systems that allow service selection [AHS03] or require an artificial, formalized subset of natural language [BKF05].

2.7 Concluding Remarks

We have described a system that makes it possible for ordinary users to invoke services using form-free task specifications. As salient features, the system strongly relies on (1) conceptual modeling, which forms the basis for both domain ontologies and process ontologies, (2) extraction ontologies, which allows the system to obtain the information it needs to match user requests with an appropriate domain ontology, and (3) domain-independent process ontologies that can be automatically specialized for any given domain, which makes our approach work across domains without need for manual configuration.

Acknowledgements

This work is supported in part by the National Science Foundation under grant IIS-0083127 and by the Kevin and Debra Rollins Center for eBusiness at Brigham Young University under grant EB-05046.

[9] By analogy, this is not unlike trying to ask for a service such as a taxi ride or a hotel reservation in a foreign country with only a limited ability to speak the language. People can succeed because they are in the right context and know enough to say to specify their needs.

Bibliography

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. Website, May 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel.

[AHS03] S. Agarwal, S. Handschuh, and S. Staab. Surfing the Service Web. *Springer-Verlag Berlin Heidelberg*, 2870(3):211–226, 2003.

[BKF05] A. Bernstein, E. Kaufmann, and N. E. Fuchs. Talking to the Semantic Web – A Controlled English Query Interface for Ontologies. *AIS SIGSEMIS Bulletin*, 2(1):42–47, January-March 2005.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[BN03] F. Baader and W. Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 2, pages 43–95. Cambridge University Press, Cambridge, UK, 2003.

[ECJ+99] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, and R. D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[EKW92] D. W. Embley, B. K. Kurtiz, and S. N. Woodfield. *Object-Oriented Systems Analysis: A Model Driven Approach*. Yourdon Press, Englewood Cliffs, New Jersey, 1992.

[Emb80] D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[GBR05] B. Gold-Bernstein and W. Ruh. *Enterprise Integration.* Addison Wesley, Boston, Massachusetts, 2005.

[Hal04] T. Halpin. Business Rule Verbalization. In *Proceedings of the 3rd International Conference on Information Systems Technology and its Applications*, pages 39–52, Salt Lake City, Utah, July 2004.

[KA04] M. Klein and B. Abraham. Towards High-Precision Service Retrieval. *IEEE Internet Computing*, 8(1):30–36, January 2004.

[LEW00] S. W. Liddle, D. W. Embley, and S. N. Woodfield. An Active, Object-Oriented, Model-Equivalent Programming Language. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, pages 333–361. MIT Press, Cambridge, Massachusetts, 2000.

[MA02] K. Mark and B. Abraham. Searching for Services on the Semantic Web using Process Ontologies. In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, editors, *The Emerging Semantic Web-Selected papers from the first Semantic Web Working Symposium*, pages 159–172. Amsterdam, Netherlands, August 2002.

[MDCG03] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 306–318, Sanibel Island, Florida, October 2003.

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

[OWL04] OWL-S Coalition, OWL-S 1.0 Release. Website, 2004. http://www.daml.org/services/owl-s/1.0/.

[PPW03] G. Papamarkos, A. Poulovassilis, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In I. Cruz, V. Kashyap, S. Decker, and R. Eckstein, editors, *Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB 2003)*, pages 309–327, Humboldt-Universität, Berlin, Germany, September 2003.

[WC95] J. Widom and S. Ceri. *Active Database Systems*. Morgan–Kaufmann, San Mateo, California, 1995.

[WS05] W3C. Web Services Activity home page. Website, 2005. http://www.w3.org/2002/ws.

Chapter 3

Ontology-Based Constraint Recognition for Free-Form Service Requests

Abstract

Automatic recognition and formalization of constraints from free-form service requests is a challenging problem. Its resolution would go a long way toward allowing users to make requests using free-form, natural-language-like specifications. In this paper, we address this challenge by offering an ontology-based, semantic-data-modeling approach to recognize constraints in free-form service requests. We encode domain information such as possible constraints and instances within a domain ontology in terms of object sets, relationship sets among these object sets, and operations over values in object sets and relationship sets. Our system recognizes the constraints in a service request by finding the domain ontology that best matches the request and then by using relationships and operations relevant to the request in the matched ontology to generate the service-request constraints. In experiments conducted with our prototype implementation, our system achieved an average of 96% recall and 99% precision.

3.1 Introduction

Allowing users to specify service requests using fully free-form specifications is likely, if successful, to enhance their ability to obtain needed services. Consider, for example, the free-form request for an appointment with a dermatologist in Figure 3.1: "I want to see a dermatologist between the 5th and the 10th, at 1:00 PM or after. The dermatologist should be within 5 miles of my home and must accept my IHC insurance." To handle this request, a system must somehow recognize the constraints involved and transform them to a formal specification such as the one in Figure 3.2. If the system can recognize the constraints in Figure 3.1 and represent them in a predicate-calculus formalism like the one in Figure 3.2, then servicing this request becomes a matter of instantiating the free variables, the x_i's, such that the constraints are satisfied.

This paper proposes a particular way to recognize constraints from free-form service requests such as the request in Figure 3.1. Rather than use traditional natural language approaches that depend on syntax analysis (e.g. [LYJ06]) or statistical analysis (e.g. [PAE04]), this paper introduces an ontological approach that depends on the long-standing notion of a semantic data model. In our ontology-based approach, a domain ontology encodes information such as applicable object sets, potential constraints over these object sets, and recognizers for instances of these object sets and constraints. The system recognizes the constraints in a service request by a two-fold process. (1) It matches a free-form service request against a collection of ontologies that belong to different domains to find the ontology that matches best. (2) It then selects from the given and implied constraints in the matched ontology those that are relevant to the service request to generate the constraints.

The semantic data model of our approach characterizes the type of service requests our system is capable of handling. Specifically, our approach handles service requests whose objective is to instantiate an object set of interest in the domain ontology with a single value such that all applicable constraints are satisfied. The objective of the appointment request in Figure 3.1, for example, is to instantiate the variable x_0 in Figure 3.2 with a value of type *Appointment* such that constraints on *Date*, *Time*, *Distance*, and *Insurance* are satisfied. This type of service covers a wide range of everyday service requests. Examples include

I want to see a dermatologist between the 5th and the 10th, at 1:00 PM or after. The dermatologist should be within 5 miles of my home and must accept my IHC insurance.

Figure 3.1: A free-form appointment request.

$$
\begin{aligned}
&//I\ want\ to\ see\ a\ dermatologist \\
&Appointment(x_0)\, is\, with\, Dermatologist(x_1) \wedge Appointment(x_0)\, is\, for\, Person(x_2) \\
&\quad \wedge Dermatologist(x_1)\, has\, Name(x_3) \wedge Person(x_2)\, has\, Name(x_4) \\
&//between\ the\ 5th\ and\ the\ 10th \\
&\wedge Appointment(x_0)\, is\, on\, Date(x_5) \wedge DateBetween(x_5,\ \text{“}the\ 5th\text{”},\ \text{“}the\ 10th\text{”}) \\
&//at\ 1{:}00\ PM\ or\ after \\
&\wedge Appointment(x_0)\, is\, at\, Time(x_6) \wedge TimeAtOrAfter(x_6,\ \text{“}1{:}00\ PM\text{”}) \\
&//within\ 5\ miles\ from\ my\ home \\
&\wedge Dermatologist(x_1)\, is\, at\, Address(x_7) \wedge Person(x_2)\, is\, at\, Address(x_8) \\
&\quad \wedge DistanceLessThanOrEqual(DistanceBetweenAddresses(x_7,\ x_8),\ \text{“}5\text{”}) \\
&\quad //accept\ my\ IHC\ insurance \\
&\wedge Dermatologist(x_1)\, accepts\, Insurance(x_9) \wedge InsuranceEqual(x_9,\ \text{“}IHC\text{”})
\end{aligned}
$$

Figure 3.2: The predicate-calculus formalism for the appointment request in Figure 3.1.

scheduling appointments, buying and selling products, renting apartments, renting cars, making hotel reservations, setting up meetings, and many more.[1]

Further, our initial work is for handling free-form service requests with conjunctive constraints. Therefore, our system in its current state does not handle service requests with negated constraints such as “not at 1:00 PM,” disjunctive constraints such as “at 10:00 AM or after 3:00 PM,” and conditional constraints such as “if the appointment can be next week, schedule me with Dr. Carter; otherwise with Dr. Jones.” Conjunctive requests are common, are a restriction to which users can likely adjust, and may be sufficiently useful by themselves. In any case, they represent a fundamental starting point from which our approach may be extended to cover other types of constraints.

Our ontology-based approach also has the interesting advantage of being fully declarative. The algorithms to find the ontology that matches best, generate constraints, and produce a formal representation for the constraints are fixed. They work across domains with no need for recoding or reconfiguration. As a consequence, to produce formal rep-

[1]We intend the word “service” to be thought of in accordance with its typical meaning—“an act of assistance or benefit.” Technically, we define a very special type of service (as described herein). We do not intend our services to be thought of in other technical ways such as registering services with a broker so that they can be found by expressing their functionality in terms of inputs, outputs, and capabilities.

resentations for service requests belonging to a new domain, it is sufficient to specify the domain ontology—no coding is necessary.

The paper makes the following contributions. First, it proposes an ontological approach to recognize and formalize constraints in free-form service requests. Second, it makes a significant step toward allowing users to invoke services by specifying them using only free-form specifications. Third, the proposed ontological approach allows service providers to define services belonging to a domain by only specifying knowledge (a domain ontology) not behavior (algorithms and code).

We present our contributions as follows. Section 3.2 introduces domain knowledge. It describes the explicitly given domain knowledge encoded in terms of a domain ontology (Subsections 3.2.1 and 3.2.2) and the knowledge implied by a domain ontology (Subsection 3.2.3). Section 3.3 shows how to match a free-form service request to a domain ontology and obtain the ontology that matches best. Section 3.4 explains how to use the matched ontology to produce formal representations. It shows how to identify the parts of the best matching domain ontology that are relevant to a service request (Subsection 3.4.1) and how to identify any needed operations (Subsection 3.4.2). It then shows how to use the ontology, including both given and implied relationships sets and operations, to generate predicates (Subsection 3.4.3). In Section 3.5 we evaluate our approach. In Section 3.6 we compare our approach to other related work, and in Section 3.7 we give concluding remarks and directions for future work.

3.2 Domain Knowledge

In this section we describe the knowledge our system needs to generate a formal representation for a service request in terms of a domain ontology. First, the system requires explicit knowledge of basic concepts related to the service request. This explicit knowledge is encoded in terms of a domain ontology, which consists of two major components: (1) a semantic data model declaring sets of objects, sets of relationships, and constraints over the object and relationship sets (Subsection 3.2.1) and (2) instance semantics declaring recognizers for object set data values as well as operations applicable to these data values

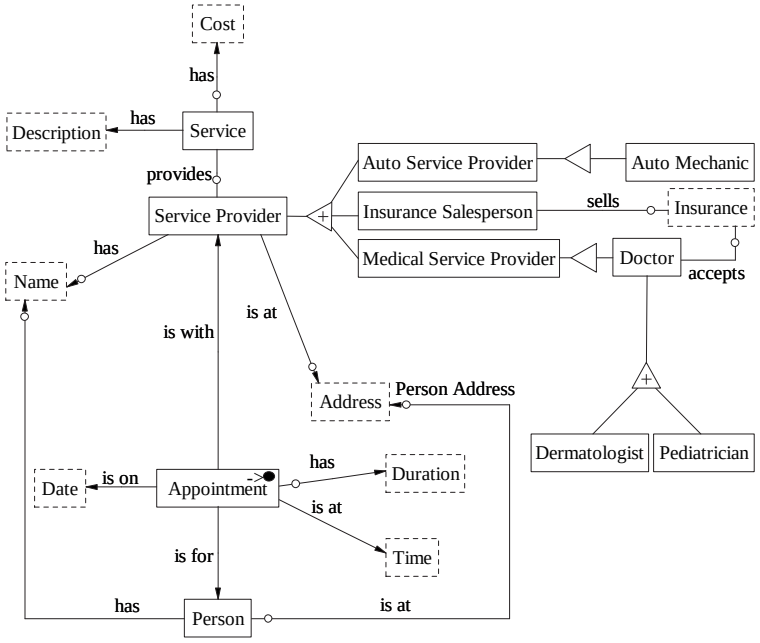


Figure 3.3: Semantic-data-model view of a domain ontology for appointments (partial).

(Subsection 3.2.2). Second, the system includes implicit knowledge—implied object sets, relationship sets, and constraints, which are based on knowledge explicitly given in the domain ontology (Subsection 3.2.3).

3.2.1 Semantic Data Model

A *semantic data model* specifies named sets of objects, which we call *object sets*, named sets of relationships among object sets, which we call *relationship sets*, and constraints over object and relationship sets. Figure 3.3 shows a small part of a semantic data model representation of a domain ontology for scheduling an appointment. The semantic data model consists of object-set concepts such as *Date*, *Time*, and *Service Provider* that can be used to schedule appointments with service providers such as doctors and auto mechanics. The semantic data model has two types of object sets, those that are lexical (enclosed in dashed rectangles) and those that are nonlexical concepts (enclosed in solid rectangles). An object set is *lexical* if its instances are indistinguishable from their representations. *Time* is an example of a lexical object set because its instances (e.g. "10:00 a.m." and "2:00 p.m.") represent themselves. An object set is *nonlexical* if its instances are object identifiers, which

represent real-world objects. *Dermatologist* is an example of a nonlexical object set because its instances are identifiers such as, say, "D_1", which represents a particular person in the real world who is a dermatologist. Each object set maps to a one-place predicate. For instance, the predicate $Date(x)$ is derived from the object set *Date* in Figure 3.3. The variable x in the predicate $Date(x)$ represents a place holder.

We designate the main object set in a semantic data model by marking it with "–> •" in the upper right corner. This notation,"–> •", denotes that when an ontology is used to satisfy a service request, the main object set becomes ("->") an object ("•"). We designate *Appointment* in Figure 3.3 as the main object set because this domain ontology is for making appointments. The system satisfies a service request by instantiating the main object set with a single value.

Figure 3.3 also shows relationship sets among object sets, represented by connecting lines, such as *Appointment is on Date*. The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *Service Provider has Name* is functional from *Service Provider* to *Name* (i.e. a service provider has only one name), and *Service Provider provides Service* is many-many (i.e. a service provider can provide many services and a service can be provided by many service providers). A small circle near the connection between an object set O and a relationship set R represents optional, so that an instance of O need not participate in a relationship in R. For example, the small circle on the *Appointment* side of the relationship set *Appointment has Duration* states that an instance of *Appointment* may or may not relate to an instance of *Duration* (i.e. there need not be a specified duration for an appointment). Each relationship set of arty n ($n \geq 2$) maps to an n-place predicate. For instance, $Appointment(x_0)$ *is with Service Provider*(x_1) is a two-place predicate derived from the relationship set *Appointment is with Service Provider* in Figure 3.3.

Constraints over unary predicates (object sets) and n-ary predicates (relationship sets) are closed predicate-calculus formulas. Referential integrity holds; thus, for example, for our ontology in Figure 3.3 we have $\forall x \forall y(Doctor(x)$ *accepts* $Insurance(y) \Rightarrow Doctor(x) \wedge Insurance(y))$. Each functional constraint from an object set O to some other object

set over a binary[2] relationship set R has the form $\forall x(O(x) \Rightarrow \exists^{\leq 1} yR(x, y))$. For instance, $\forall x(\textit{Service Provider}(x) \Rightarrow \exists^{\leq 1}\text{y}(\textit{Service Provider}(x)\ \textit{has Name}(y)))$ is the functional constraint for the relationship set from *Service Provider* to *Name*. Each constraint for a mandatory object set O for a binary relationship set R has the form $\forall x(O(x) \Rightarrow \exists^{\geq 1} yR(x, y))$. For instance, $\forall x(\textit{Service Provider}(x) \Rightarrow \exists^{\geq 1}\text{y}(\textit{Service Provider}(x)\ \textit{has Name}(y)))$ is the mandatory constraint for *Service Provider* in the *Service Provider has Name* relationship set.

A triangle in an ontology diagram (see Figure 3.3) denotes generalization/specialization. The generalization object set connects to the apex of the triangle, and specialization object sets connect to its base. For each generalization/specialization, we write the constraint $\forall x(S_1(x) \vee ... \vee S_n(x) \Rightarrow G(x))$, where G is the generalization object set and S_1, ..., S_n are the specialization object sets. If the generalization/specialization has mutual-exclusion constraint (represented by the "+" in the triangle in Figure 3.3), we also write the constraints $\forall x(S_i(x) \Rightarrow \neg S_j(x))$ for $1 \leq i,\ j \leq n,\ i \neq j$. In Figure 3.3, for example, the constraint $\forall x(\textit{Dermatologist}(x) \vee \textit{Pediatrician}(x) \Rightarrow \textit{Doctor}(x))$ states that dermatologists and pediatricians are specializations of doctors, and the constraints $\forall x(\textit{Dermatologist}(x) \Rightarrow \neg \textit{Pediatrician}(x))$ and $\forall x(\textit{Pediatrician}(x) \Rightarrow \neg \textit{Dermatologist}(x))$ state that dermatologists and pediatricians are mutually exclusive.

Every connection between an object set and a relationship set is a role. A role designates the set of objects of an object set that participate in a relationship set. If we wish to name the role, we place the role name near the connection between its object set and its relationship set. For instance, the role *Person Address* in Figure 3.3 appears near the connection between the object set *Address* and the relationship set *Person is at Address*. A named role is a specialization of the object set to which it connects. *Person Address* thus represents the subset of addresses that associate with persons.

[2]The definition of the constraints for binary relationship sets can easily be extended to n-ary relationship sets for $n > 2$.

3.2.2 Data Frames

Each object set (including each named role) in a domain ontology has an associated data frame [Emb80], which describes instances for the object set. Data frames capture the information about object-set instances in terms of their external and internal representation, their context keywords or phrases that may indicate their presence, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the object set along with context keywords or phrases that indicate the applicability of an operation and operands in an operation. Figure 3.4 shows sample (partial) data frames for *Time*, *Date*, *Address*, *Person Address*, *Dermatologist*, *Appointment*, *Insurance*, and *Distance*.

As Figure 3.4 shows, we use regular expressions to capture external textual representations. The *Time* data frame, for example, captures instances that end with "AM" or "PM" (e.g. "2:00 PM" and "9:30 a.m."). A data frame's context keywords/phrases are also regular expressions. For example, the *Distance* data frame in Figure 3.4 includes context keywords such as "miles" or "kilometers". In the context of one of these keywords, if a number appears, it is likely that this number is a distance. A nonlexical object set such as *Dermatologist* has only context keywords or phrases. Figure 3.4 shows that the *Dermatologist* data frame includes a regular expression, which includes keywords and phrases that could indicate the presence of an instance of a dermatologist.

The operations in data frames manipulate object-set instances. For example, the operation *DistanceBetweenAddresses*(*a1*: *Address*, *a2*: *Address*) computes the distance between its two address arguments *a1* and *a2*. Boolean operations represent possible general constraints in the domain. For instance, the Boolean operation *TimeAtOrAfter*(*t1*: *Time*, *t2*: *Time*) in the *Time* data frame returns *true* if time *t1* is the same as or comes after time *t2*.

The context keywords/phrases for an operation indicate the possible applicability of the operation. The context keywords/phrases are regular expressions that include keywords or phrases and possibly expandable expressions represented by operand names enclosed in braces. The system expands these expressions by finding the types of their

```
Time
  ...
  text representation:
    ([2-9]|1[012]?):([0-5]\d)\s*[aApP]\.?[mM]\.?|...
  TimeAtOrAfter(t1: Time, t2: Time)
    returns (Boolean)
    context keywords/phrases:
      (at\s+)?{t2}\s+or\s+after|...
  TimeEqual(t1: Time, t2: Time)
    returns (Boolean)
    context keywords/phrases: (at\s+)?{t2}
  ...
Date
  ...
  text representation:
    ...|(the\s+)?([1-9]|[12]\d|3[01])\s*(th|...)|...
  DateBetween(x1: Date, x2: Date, x3: Date)
    returns (Boolean)
    context keywords/phrases:
      between\s+{x2}\s+and\s+{x3}|...
  ...
Address
  ...
  DistanceBetweenAddresses(a1: Address, a2: Address)
    returns (Distance)
  ...
Person Address
  ...
  context keywords/phrases:
    (my\s+)?home|(my\s+)?house|where\s+I\s+live|...
  ...
Dermatologist
  internal representation: object id
  context keywords/phrases:
    [Dd]ermatologist|skin\s+doctor|...
  ...
Appointment
  internal representation: object id
  context keywords/phrases:
    appointment|want\s+to\s+see\s+an?|...
  ...
Insurance
  ...
  text representation: IHC|DMBA|...
  context keywords/Phrases:
    insurance|medical\s+insurance|...
  InsuranceEqual(i1: Insurance, i2: Insurance)
    returns (Boolean)
    context keywords/phrases: {i2}
  ...
Distance
  internal representation: real
  text representation: \d+(\.\d+)?|(\.\d+)
  context keywords/phrases: miles?|kilometers?|...
  DistanceLessThanOrEqual(d1: Distance, d2: Distance)
    returns (Boolean)
    context keywords/phrases: (within|...)\s+{d2}|...
  ...
```

Figure 3.4: Some sample data frames (partial). (The "..." in the textual-representation part of the *Time* data frame indicates that there are other representations of *Time* such as military time. In general the presence of ellipses show omissions needed to complete the data frames.)

operands and substituting the textual representations in the data frames of the types for these expressions. The advantage of marking these expandable expressions with operands is that when context keywords/phrases for an operation match substrings in a service request, the system can record which values are for which operands. For instance, the context keywords/phrases associated with the operation *DateBetween* in Figure 3.4 has the regular expression *between\s+{x2}\s+and\s+{x3}*, which includes the expandable expressions {*x2*} and {*x3*}. As Figure 3.4 shows, the operands of these two expressions are of type *Date*. When this regular expression matches a substring in a request such as "make the appointment between the 10th and the 15th," the system can record that the first date value ("the 10th") is for *x2* and the second date value ("the 15th") is for *x3*.

3.2.3 Implied Knowledge

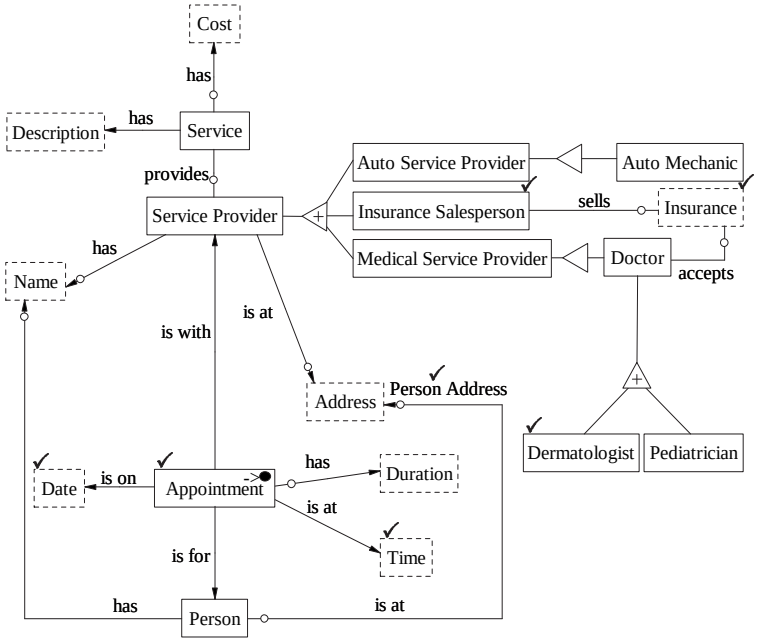
Object sets, relationship sets, and constraints that can be computed from the domain ontology constitute the implied knowledge. For example, the system can derive a relationship set between *Appointment* and *Name* from the given relationship sets *Appointment is with Service Provider* and *Service Provider has Name*. The system can also determine that *Name* mandatorily depends on *Appointment* from the given constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *is with Service Provider*$(y))$) and $\forall$x(*Service Provider*$(x) \Rightarrow \exists^{\geq 1} z($*Service Provider*$(x)$ *has Name*$(z))$), where the former states that *Service Provider* is mandatory for *Appointment* and the latter states that *Name* is mandatory for *Service Provider*. Further, the system can determine that *Name* functionally depends on *Appointment* from the given constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\leq 1} y(Appointment(x)$ *is with Service Provider*$(y))$) and $\forall$x(*Service Provider*$(x) \Rightarrow \exists^{\leq 1} z($*Service Provider*$(x)$ *has Name*$(z))$). As additional examples, there are many implied generalization/specialization constraints derivable from the constraints in Figure 3.3. For instance, the system can derive the implied constraint $\forall x(Dermatologist(x) \Rightarrow Service\ Provider(x))$ by transitivity from the following given constraints: $\forall x(Dermatologist(x) \Rightarrow Doctor(x))$, $\forall x(Doctor(x) \Rightarrow Medical\ Service\ Provider(x))$, and $\forall x(Medical\ Service\ Provider(x) \Rightarrow Service\ Provider(x))$.

The connections between operands of an operation in a data frame and the relationship sets of a semantic data model may be implicit. Consider, for example, the

operation *DistanceBetweenAddresses* in the *Address* data frame, which computes the distance between two addresses. It is not explicitly given in the domain ontology, Figures 3.3 and 3.4, whether this operation computes the distance given two service-provider addresses, two person addresses, or a service-provider address and a person address. The system, however, can reason that for an appointment, if there is a constraint on distance, then it must be between a service-provider address and a person address. The system reasons as follows. The constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\leq 1} y(Appointment(x)$ *is with Service Provider*$(y))$) and $\forall$x($Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *is with Service Provider*$(y))$) allow the system to infer the implicit constraint $\forall$x($Appointment(x)$ $\Rightarrow \exists^{1} y(Appointment(x)$ *is with Service Provider*$(y))$), which states that for any appointment there exists exactly one service provider. The system can derive from the constraints $\forall$x(*Service Provider*$(x) \Rightarrow \exists^{\leq 1} y($*Service Provider*$(x)$ *is at Address*$(y))$) and $\forall$x(*Service Provider*$(x) \Rightarrow \exists^{\geq 1} y($*Service Provider*$(x)$ *is at Address*$(y))$) the constraint $\forall$x(*Service Provider*$(x) \Rightarrow \exists^{1} y($*Service Provider*$(x)$ *is at Address*$(y))$), which states that there is exactly one address for a service provider. Given these two derived constraints, since there is at most one service-provider address for an appointment, the system can certainly exclude the possibility that *DistanceBetweenAddresses* computes distances between two addresses of service providers when it makes an appointment. Likewise, the system can exclude the possibility that *DistanceBetweenAddresses* computes the distance between addresses of persons. Thus, the system can infer that the two operands *a1* and *a2* of the operation *DistanceBetweenAddresses* must obtain their values from addresses in the relationship sets *Service Provider is at Address* and *Person is at Address*.

3.3 Domain Ontology Recognition

In our ontology-based approach, the objective of the domain ontology recognition process is to find a domain ontology that best matches a service request. The process takes a set of available ontologies belonging to different domains and a service request as input and returns a marked-up domain ontology that best matches the service request as output.



(a) Matched (√) object sets in the semantic data model in Figure 3.3.

√Distance
√TimeAtOrAfter(t1: Time, "1:00 PM")
√DateBetween(x1: Date, "the 5th", "the 10th")
√DistanceLessThanOrEqual(d1: Distance, "5")
√InsuranceEqual(i1: Insurance, "IHC")

(b) Matched (√) object sets and operations in the data frames in Figure 3.4.

Figure 3.5: Output of the recognition process—the marked-up domain ontology.

For each domain ontology, the system applies all the recognizers in the data frames of every object set in the domain ontology to the service request. It marks every object set whose recognizers match a substring in the service request and every operation whose applicability recognizers match a substring in the service request. The result of the matching is a set of marked-up domain ontologies.[3]

When the recognition process executes for the domain ontology in Figures 3.3 and 3.4 and the appointment request in Figure 3.1, it produces as output the marked-up ontology in Figure 3.5. Figure 3.5(a) shows the matched (√) object sets in the semantic data model in

[3]To scale this part of the ontology recognition process to a large set of ontologies, we would need to use some form of indexing or do some light-weight filtering to reduce the large set of ontologies to a small set of candidate ontologies.

Figure 3.3, and Figure 3.5(b) shows the matched ($\surd$) operations and the additional object sets from Figure 3.4. The recognizers in the data frame in Figure 3.4 for *Dermatologist* recognize the context keyword "dermatologist" in the service request in Figure 3.1, and therefore *Dermatologist* is marked ($\surd$). Likewise, as Figure 3.4 makes evident, recognizers in the *Date* data frame recognize "between the 5th and the 10th"; in the *Time* data frame recognize "at 1:00 PM or after"; in the *Distance* data frame recognize "within 5 miles"; in the *Appointment* data frame recognize "want to see a"; in the *Insurance* data frame recognize the context keyword "insurance" and the constant value "IHC"; and in the *Person Address* data frame recognize the context phrase "my home". Therefore these object sets are marked. Although not included in Figure 3.4, we assume that the recognizer for context keywords in the *Insurance Salesperson* data frame would recognize "insurance". Therefore *Insurance Salesperson* is marked.

Given the data frames in Figure 3.4, additional matched operations and object sets may have been expected. For example, the context keywords/phrases for the operation *TimeEqual* in the *Time* data frame would match "at 1:00 PM" and the *Cost* data frame may have recognizers that match "within 5". We eliminate these matches, however, based on a subsumption heuristic. This heuristic uses the positions of the matched substrings in a service request to determine whether a matched substring subsumes another matched substring. The system does not mark an object set or an operation if its matched substring is properly subsumed by another matched substring. We assume that there is only one match for a string and that the subsuming substring is a better match. Thus, although the context keywords/phrases for the operation *TimeEqual* would recognize "at 1:00 PM", the system would not mark the operation *TimeEqual* because it matches with only the substring "at 1:00 PM", which is subsumed by the substring "at 1:00 PM or after", matched by the operation *TimeAtOrAfter*.

To choose the marked-up domain ontology that best matches the service request, the system ranks them. In our approach, the system grants rank values for each marked-up domain ontology based on the marked object sets. The marked main object set of the marked-up ontology has the highest weight for obvious reasons. Marked mandatory object sets contribute with the next highest weight because they represent the necessary

requirements to establish the main concept. Marked optional object sets contribute with lower weights because they are not necessary for establishing the main concept.[4] To continue with our running example, we assume that the system selects our appointment ontology as the best matched ontology for the service request in Figure 3.1.

3.4 Formal Representation Generation

A formal representation of a free-form service request is a predicate-calculus formula. The system generates the predicates of a formal representation for a free-form service request only from the given and implied knowledge. It cannot generate predicates for constraints in a service request that refer to object sets, relationship sets, constraints, or operations beyond this knowledge. For instance, if the appointment ontology designer leaves out the *Insurance* object set, any constraint in a service request about insurance such as "must accept my IHC insurance" will be ignored.

The input to the formal representation generation process is a marked-up ontology. The output is a predicate-calculus formula. Not all knowledge in a marked-up ontology is relevant. Irrelevant knowledge should be pruned away. Otherwise, the system will generate an overconstrained predicate-calculus formula. The system, therefore, should find the sub-ontology including object sets, relationship sets, and operations that are relevant to the service request. We call this sub-ontology the *service request view*. In Subsections 3.4.1 and 3.4.2, we explain how the system generates the components of the service request view. In Subsection 3.4.3, we show how the system uses the service request view to generate the formal representation.

[4]The actual weights that we used in our experiments were 3 for the main object set, 2 for each object set that mandatorily depends on the main object set, and 1 for each optional object set with respect to the main object set. Our system was able to uniquely select the right ontology using these weights. However, more sophisticated weights and heuristics may be necessary as the number of ontologies and the overlap among these ontologies increase.

3.4.1 Relevant Object Set and Relationship Set Identification

In this section, we explain how the system uses the explicit and implicit knowledge in a marked-up ontology to find the object sets and the relationship sets that are relevant for a service request. In general, the relevant object sets and relationship sets are:

1. the main object set (the object set marked with "–> •") because we must establish an object in this object set to satisfy the service request;
2. the object sets that mandatorily depend on the main object set either directly or transitively because they are the essential requirements to establish an object in the main object set;
3. the marked optional object sets because they represent additional, user-chosen requirements on the requested service; and
4. the relationship sets that connect these object sets.

All other object sets and relationship sets are pruned away.

The system obtains the object sets that mandatorily depend on the main object set from the given and implied relationship sets that involve the main object set and from the given and implied constraints for these relationship sets. In our running example, the given relationship set *Appointment is with Service Provider* shows that *Service Provider* is related to *Appointment*, and the given constraint $\forall x(Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *is with Service Provider*$(y)))$ shows that *Service Provider* is mandatory. Further, as we discussed in Subsection 3.2.3, there is an implied relationship set between *Appointment* and service-provider *Name*, and an implied constraint for this implied relationship set that makes *Name* mandatorily depend on *Appointment*. Likewise, the system can infer that *Date*, *Time*, *Person*, service-provider *Address*, and person *Name* are all mandatory.

The object set *Duration* optionally depends on the main object set because of the absence of the constraint $\forall x(Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *has Duration*$(y)))$, which allows the object set *Duration* to be optional. Since *Duration* is not marked, the system does not include it as a relevant concept for the service request. Likewise, since the object sets *Service*, *Price*, and *Description* are optional with respect to the main object set

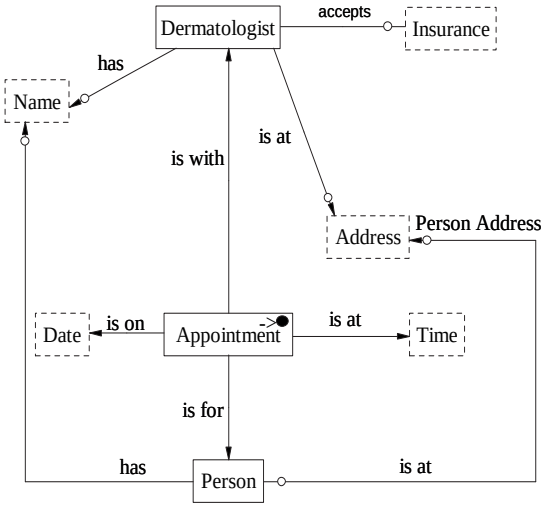


Figure 3.6: The relevant object sets and relationship sets for the appointment in Figure 3.1.

and unmarked, the system does not included them. Although *Person Address* optionally depends on the main object set *Appointment*, the system keeps it because it is marked.

To determine what the system keeps in a generalization/specialization (is-a) hierarchy, the system considers the constraints imposed by the main object set on an is-a hierarchy and the constraints that the hierarchy imposes on its object sets. If the constraints imposed by the main object set on the is-a hierarchy allow only one instance of a marked specialization and the marked specializations are mutually exclusive, the system keeps only one marked specialization. The reason is that the instance in this case can be in only one marked specialization.

Referring to our example, the implied constraint $\forall x(Appointment(x) \Rightarrow \exists^1 y(Appointment(x)$ *is with Service Provider*$(y)))$ requires exactly one instance value in the is-a hierarchy to be associated with an appointment. Further, the implied mutual exclusion constraint between the marked specializations, *Dermatologist* and *Insurance Salesperson*, allows the system to infer that the single instance must belong to only one of these marked specializations. To determine which one of the marked specializations, the system ranks them. Each marked specialization receives a rank value according to: (1) the number of strings in a service request matched by the data frame recognizers of the specialization, (2) the number of the marked object sets directly related to the specialization, and (3) the distance between the locations of the strings in the service request matched by the special-

ization and the locations of the strings in the service request matched by the main object set. For the first criterion for our example, *Dermatologist* matches with more strings (two occurrences of "dermatologist") than does *Insurance Salesperson* (matches with the single string "insurance"). For the second criterion, both the marked specializations relate to one marked object set, *Insurance*. (Observe that since *Dermatologist* in Figure 3.5 is a *Doctor*, it inherits all the relationship sets in which *Doctor* is involved, and thus *Dermatologist* is connected to *Insurance*.) For the third criterion, the location of the first occurrence of "dermatologist" in the service request is closer to the location of the string "want to see a", matched by the main object set than is the location of the string "insurance", matched by *Insurance Salesperson*. Thus, the system keeps only the marked specialization *Dermatologist* in the is-a hierarchy. The system removes all the other specializations and collapses the is-a hierarchy. Figure 3.6 shows the resulting relevant object sets and relationship sets for the appointment request in Figure 3.1.

When the constraints imposed by the main object set allow only one marked specialization, but mutual-exclusion constraints in the is-a hierarchy do not force the single instance to be in only one marked specialization, it is possible that the single instance could belong to one or more of the marked specializations. For this case we find the least upper bound object set O_{LUB} in the is-a hierarchy to which instances of all marked specializations belong. We then prune away all unmarked specializations in the is-a hierarchy, collapse all specializations to O_{LUB}, and replace the root object set with O_{LUB}. In doing so, we also keep all relationship sets that lead from object sets in the is-a hierarchy that are not pruned away to other marked object sets. These other marked object sets are related mandatorily or optionally to O_{LUB} depending on given or implied constraints.

When the constraints imposed by the main object set allow more than one marked specialization, we find the least upper bound object set O_{LUB} for the marked specializations. We then prune away all the other specializations from the is-a hierarchy and collapse the is-a hierarchy as described for the previous case.

Finally, if there is no marked specialization in an is-a hierarchy but an element in the is-a hierarchy is mandatory, we keep the root of the is-a hierarchy and prune away all its specializations. We also keep all relationship sets that lead to marked object sets, if any,

and optionally connect them to the root. If no element in the is-a hierarchy is mandatory and none is marked, we discard the entire hierarchy and all connected relationship sets.

3.4.2 Relevant Operation Identification

The operations relevant to a service request are the Boolean operations whose applicability recognizers match strings in the service request and operations on which operands of these Boolean operations may depend for values. For our appointment example, the Boolean operations in Figure 3.5(b) are the relevant Boolean operations.

The system needs to bind the operands of the operations that, as of yet, are not instantiated to value sources. Value sources can be the relevant object sets for the service request or operations in the data frames that compute values for the operands. In our running example, the operation *DateBetween* has the uninstantiated operand *x1* of type *Date*. Since *Date* is involved in one relationship set *Appointment is on Date*, the system binds *x1* to this relationship set yielding the constraints *Appointment*(*x0*) *is on Date*(*x1*) $\wedge$ *DateBetween*(*x1*, "*the 5th*", "*the 10th*").[5] Similarly, the system binds the uninstantiated operands *t1* in *TimeAtOrAfter* to yield the constraint *Appointment*(*x0*) *is at Time*(*t1*) $\wedge$ *TimeAtOrAfter*(*t1*, "*1:00 PM*") and the uninstantiated operand *i1* in *InsuranceEqual* to yield the constraint *Dermatologist*(*x3*) *accepts Insurance*(*i1*) $\wedge$ *InsuranceEqual*(*i1*, "*IHC*").

The operand *d1* of the operation *DistanceLessThanOrEqual* is of type *Distance*, which is not involved in any given relevant object set in Figure 3.6. The system, therefore, must find an operation that depends on the relevant object sets and computes values for this input parameter. If the system cannot find such an operation, the operation is ignored. The operand *d1* can potentially be computed by the operation *DistanceBetweenAddresses*, which depends on the relevant object set *Address*. The system, therefore, binds *d1* to the operation *DistanceBetweenAddresses*. As we discussed in Subsection 3.2.3, the system can infer from constraints on the relationship sets on which the operation *DistanceBetweenAddresses* depends that the address values *a1* and *a2* come respectively from the *Address* object sets in *Dermatologist is at Address* and *Person is at Address*.

[5] Note that it is important here to be able to assign values recognized by expandable expressions to their respective operands.

- *Appointment*(x_0) *is at Date*(x_1) ∧ *DateBetween*(x_1, "*the 5th*", "*the 10th*")
- *Appointment*(x_0) *is at Time*(*t1*) ∧ *TimeAtOrAfter*(*t1*, "*1:00 PM*")
- *Dermatologist*(x_3) *is at Address*(*a1*) ∧ *Person*(x_2) *is at Address*(*a2*) ∧ *DistanceLessThanOrEqual*(*DistanceBetweenAddresses*(*a1*, *a2*), "*5*")
- *Dermatologist*(x_3) *accepts Insurance*(*i1*) ∧ *InsuranceEqual*(*i1*, "*IHC*")

Figure 3.7: The relevant operations for the appointment request in Figure 3.1.

Figure 3.7 shows the relevant operations for the appointment request in Figure 3.1. Each input parameter is either instantiated with a value or bound to an operation that yields a value or to a (possibly unknown but nevertheless specific) value in an object set.

3.4.3 Predicate-Calculus Formula Generation

The system conjoins the predicates generated as described in Subsection 3.4.1 and Subsection 3.4.2 to generate the formal representation for a free-form service request. For our running example, the system conjoins the predicates for each relationship set in Figure 3.6 with the formulas in Figure 3.7 to produce the formal representation for the service request in Figure 3.1. After renaming variables, we have exactly the predicate-calculus formula in Figure 3.2.

We point out that the algorithms to identify the relevant object sets, relationship sets, and the operations work on general ontological knowledge. The algorithms consider whether object sets are marked or not, and they consider constraints over relationships and among operations in data frames. The knowledge the algorithms consider is independent of a specific domain. As a significant consequence, these algorithms are fixed and work across domains with no need to recode them.

3.5 Performance Analysis

We conducted experiments to evaluate our system. The objective was to evaluate the system performance in finding the predicates of a formal representation for a free-form service request and values for predicate arguments. We tested the system on service requests belonging to the following domains: scheduling appointments with medical doctors, purchasing cars, and renting apartments.

Assume that you want make an appointment with some doctor. Assume further that you have software that can schedule the appointment by allowing you to specify your appointment using natural language (English). Use your own words to write an appointment request for some doctor (use only one of these doctor specialties: dermatologist, pediatrician, dentist, gynecologist, and neurologist). You can specify constraints on the requested appointment such as date, time, how far are you willing to go, type of insurance that the doctor should accept, and so forth.

In your request, please do not include alternative-choice constraints such as "I want the appointment at 10:00 AM *or* after 3:00 PM" or negated constraints such as "the appointment should *not* be at 9:00 AM."

Figure 3.8: Instructions for subjects for the appointments domain.

We asked subjects from Brigham Young University to make free-form, natural-language-like service requests belonging to these domains using their own words. The subjects ranged from savvy computer users and online shoppers to users with limited computer skills. We provided the subjects with no information about the structure of the underlying domain ontologies or the recognizers or operations in the data frames. We asked the subjects to make service requests with only conjunctive constraints and positive literals—the type of constraints that our system is capable of recognizing. To avoid technical terms (e.g. "conjunctive" and "positive literals"), we provided users with illustrative examples to use for formulating their requests. Figure 3.8 shows the instructions we provided for subjects to write appointment requests for medical doctors.

Table 3.1 shows the number of service requests and the number of included predicates and values in these requests for each of the three domains. We received a total of 31 requests, which included a total of 548 constraints and a total of 170 constant values. We reviewed all service requests we received, manually extracted the included constraints and constant values in each service request, assigned each constant value to its respective operand, manually generated a formal representation for each request, and stored it in a format similar to the way the system records results. We then fed each service request to the system, which created the formal representation for the request, compared this formal representation against the manually generated request, and automatically computed the recall and precision.

Table 3.1: Number of service requests, predicates, and arguments.

	Requests	Predicates	Arguments
Appointment	10	126	34
Car Purchase	15	315	98
Apt. Rental	6	107	38
Totals	31	548	170

Table 3.2 shows the performance of the system in finding predicates and constant values in terms of the recall and precision for each one of the three domains along with the overall recall and precision. The system performed surprisingly well.

As Table 3.2 shows, the recall for predicates was high for all three domains. The recall numbers for constant values (arguments) were a little lower, but nevertheless quite high. The system did not recognize these variations of date for appointments: "any Monday of this month" and "most days of the week", these features for cars: "power doors and windows" and "v6" (the engine size), and these features for apartments: "a nook", "dryer hookups", and "extra storage". Therefore, the recall for arguments in the appointments, cars, and apartments rental domains dropped off from 100%. Further, missing these constant values caused the system to miss the constraints over these values causing the recall for predicates in the appointments, cars, and apartments rental to be lower than they otherwise would have been.

We can fix these recall problems by providing better recognizers that can better cover the space of possible values for the object sets. We recognize, however, that this may not always be easy. In [KCGS96], for example, the authors describe an automaton with 1,223 states and 21,006 arcs to correctly recognize strings representing a date. Although not always easy to obtain full coverage, it is possible, with reasonable effort, to obtain near full coverage. The advantage gained may very well be worth the effort.

The precision was near 100% for both predicates and arguments. When the system selects the right ontology for a service request, the system almost cannot obtain irrelevant constraints because our ontology is narrowly focussed on the service. The only way the system can produce an irrelevant predicate is when the system incorrectly marks an operation or an object set based on the appearance of some constant value or a context

Table 3.2: Recall and precision.

		Recall	Precision
Appointment	predicates	0.978	1.000
	arguments	0.941	1.000
Car Purchase	predicates	0.998	0.999
	arguments	0.979	0.997
Apt. Rental	predicates	0.968	1.000
	arguments	0.921	1.000
All	predicates	0.981	0.999
	arguments	0.947	0.999

keyword/phrase and the ontological knowledge is not enough to enable the system to prune it away. Consider, for example, this constraint "I want a Toyota with a cheap price, 2000 would be great ...", which was taken from one of the requests and for which our system incorrectly generated the constraint, *PriceEqual*(*p1*: *Price*, "*2000*"). The appearance of the contextual keyword "price" close to the number 2000 makes our system recognize 2000 as a price value rather than a year value. The type of ambiguity in this constraint is not easy to handle (perhaps not even easy for humans) because it is not so clear whether the subject meant the price to be 2000 or the year to be 2000.[6]

3.6 Related Work

Some researchers in the natural language processing community work on systems that transform natural language to a formal specification such as predicate calculus, as we do here. These systems, called logic form generation or transformation systems [BBGW04, AP04, MMP04], use parsers to parse a syntactically correct sentence and identify its constituents such as nouns, verbs, and adjectives. Each constituent defines a predicate. The syntactic structure of a parsed sentence defines the relationships among the constituents, which are captured through shared arguments among the predicates. Based on reported results in [BBGW04], [AP04], and [MMP04] and in [Rus04], which compares the performance of three other approaches, these systems are able to achieve a recall within

[6]Note that the "a" that would usually have appeared in front of "2000" really is missing. If it had been there, our system would have correctly extracted the "2000" as a year.

the interval [78%, 90%] and a precision within [81%, 87%] at the predicate level, and a recall within [65%, 77%] and a precision within [72%, 77%] at the argument level.

For many years, researchers in the database community have also worked on generating constraints from natural language queries. Older approaches, surveyed in [ART95], parse their input using either syntactic parsers or sematic parsers to produce parse trees. The main difference between syntactic and semantic parsing is that the grammar categories for semantic parsing directly correspond to database elements such as table names and attributes names rather than syntactic concepts such as noun phrases. In both cases, the parse tree is used to generate a database query with the help of mapping rules that specify how each element in the parse tree maps to an element in the database query.

Newer approaches build on these older approaches by introducing additional techniques that improve results. The approach proposed in [LYJ06] uses a dependency parser to determine how the words in a sentence depend on each other. A query is parsed to create the parse tree, which captures the dependencies between the query tokens, and then each node in the parse tree is classified according to XQuery components (e.g. a **return** clause). If the system cannot classify some node in the parse tree, it asks a user to rephrase the query. The **where** clause constraints are created based on patterns that appear in a dependency tree. For instance, the appearance of the pattern "$\langle variable \rangle$ + $\langle constant \rangle$"—i.e. a variable followed by a constant value—maps to the constraint "*variable* = *constant*" in **where** clause. Experiments reported in [LYJ06] show that this approach is able to achieve 95.1% precision and 97.6% recall. These results are for queries that are correctly parsed and whose resulting parse-tree nodes are correctly classified. With respect to all queries, however, the reported recall and precision were respectively 90.1% and 83%.

The PRECISE system, proposed first in [PEK03] and later enhanced with a semantic model to correct some parser errors [PAE04], uses a statistical parser and lexicons, consisting of names of relations, attributes, and values of the attributes as well as *wh*-designators (what, which, where, who, and when designators) attached to the attributes. A natural language query is parsed with respect to the lexicon that matches each main word in the query to one or more database elements (table name, attribute name, value, and *wh*-designator). The system then constructs attribute-value mappings, which are validated

by the relationships produced by the parser. The **where** clause in the generated SQL query is a conjunction of attributes with mapped values along with join conditions that reflect the join paths among tables. The reported results for experiments on three domains show that PRECISE is able to achieve 100% precision and a recall within the interval [∼75%, ∼93%] for "semantically tractable queries." Like our proposed system, PRECISE extensively exploits the schema of the database. Since neither system generates constraints beyond its schema, precision tends to be high. Improper constraints can only be generated by false positives within the purview of the database schema.

The approach described in [MC99] is quite close to our approach. It uses a semantic model of an underlying database, which is a graph that consists of nodes representing database relations and attributes and edges representing connections among relations. Keywords or keyword phrases are attached as labels to nodes and operators (standard operators such as "<", ">", or "="). Operators are attached to the attributes if these operators apply to values of the attributes. The system matches a natural language query to the keywords attached to semantic-model elements and uses a statistical approach (n-grams) to disambiguate matches. As with our approach, this approach does not seem to require syntactically correct queries. No empirical results are reported in [MC99], and therefore it is hard to assess its performance.

All these approaches, except [MC99], expect syntactically correct sentences. We do not. Further, generally speaking, our approach performed with better recall and precision. We believe that our approach has two important novelties that contribute to its performance. First, the semantic data model captures the relationships among objects and constraints over these objects in the domain, and therefore we avoid precision errors introduced when parsers try to determine relationships among constituent parts of the input. Further, as an added benefit of our particular service-oriented paradigm, the semantic data model allows the system to derive relationships that are necessary for satisfying a service request even though the service request does not specify them at all. Second, the semantics associated with the object sets through data frames allow our approach to capture constraints through operations in these data frames. This means that once a constraint in a service request is recognized by the applicability recognizers of an operation, then this constraint is correctly

formalized by means of this operation. Our approach, however, does require designers of service-request ontologies to produce a proper semantic data model that appropriately covers the scope of the service and to produce recognizers in data frames that correctly recognize appropriate value and keyword instances. We believe, however, that because of the narrow focus of a particular service, this task is as easy (and possibly easier) than producing required lexicons, parsers, and similar components for alternative approaches.

3.7 Conclusions and Future Work

We proposed an ontology-based approach for recognizing constraints in a free-form, fully unconstrained service requests and formally representing them in terms of predicate calculus formulas. We tested our proposed approach and found that it achieved a recall averaging 98.1% for predicates and 94.7% for arguments, and achieved a precision of near 100% for both predicates and arguments. Thus, we believe that our approach is likely to be a valuable alternative in situations where (1) the input is a free-form service request with conjunctive constraints, (2) the request provides enough of a hint to allow our system to find a matching domain ontology, and (3) the request can be satisfied by inserting a single object in an object set of interest in a domain ontology and then by inserting other mandatory and optional objects required for the request.

We have two main objectives for future work. First, we have recently extended the capabilities of our system to recognize and process disjunctive and negated constraints. We intend to conduct a user study to evaluate the performance of our augmented system. Second, we plan to integrate the work reported here with other work we have done [AME06] to produce the overall system we have envisioned [AMEL05]. The system we have envisioned transforms a service request into a predicate-calculus formula as explained here. It uses the predicate-calculus formula to create a query to a databases associated with the domain ontology from which the formula was generated to instantiate as many variables of the formula as possible. The system then discovers the variables in the predicate-calculus formula that are yet to be instantiated and interacts with a user to obtain values for these variables. When all the variables are instantiated, the system checks whether the constraints

of the formula are satisfied. Constraint satisfaction can yield too many solutions or no solution. As reported in [AME06], the system controls the potential overload on users when there are too many solutions by returning the best-m solutions rather than all of them or offers users the best-m near solutions when there is no solution. When a user chooses one of the suggested solutions or near solutions, the system completes the service request by inserting an object of interest (e.g. an appointment) in the main object set of the domain ontology and by inserting other mandatory and optional objects and relationships and thus satisfies the service request.

Acknowledgements

This work is supported in part by the National Science Foundation under grants 0083127 and 0414644. Also, we would like to appreciate the help of all the subjects from Brigham Young University who participated in the experiments.

Bibliography

[AME06] M. J. Al-Muhammed and D. W. Embley. Resolving Underconstrained and Overconstrained Systems of Conjunctive Constraints for Service Requests. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*, pages 223–238, Luxembourg, June 2006.

[AMEL05] M. J. Al-Muhammed, D. W. Embley, and S. W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 288–303, Klagenfurt, Austria, October 2005.

[AP04] S. Anthony and J. Patrick. Dependency Based Logical Form Transformation. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 54–57, Barcelona, Spain, July 2004.

[ART95] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural Language Interfaces to Database: An Introduction. *Journal of Natural Language Engineering*, 1(1):29–81, March 1995.

[BBGW04] S. Bayer, J. Burger, W. Greiff, and B. Wellner. The MITRE Logical Form Generation System. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 69–72, Barcelona, Spain, July 2004.

[Emb80] D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980.

[KCGS96] L. Karttunen, J. P. Chanod, G. Grefenstette, and A. Schille. Regular Expressions for Language Engineering. *Natural Langauge Engineering*, 2(4):305–328, December 1996.

[LYJ06] Y. Li, H. Yang, and H.V. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT 2006)*, pages 737–754, Munich, Germany, March 2006.

[MC99] F. Meng and W. W. Chu. Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation. Technical Report CSD-TR 990003, University of California, Los Angeles, California, 1999.

[MMP04] A. Mohammed, D. Moldovan, and P. Parker. Sensevale 3 Logic Form: A System and Possible Improvements. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 163–166, Barcelona, Spain, July 2004.

[PAE04] A. M. Popescu, A. Armanasu, and O. Etzioni. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *Proceedings of the 20th International Confereence on Computational Linguistics*, pages 30–39, University of Geneva, Switzerland, August 2004.

[PEK03] A. M. Popescu, O. Etzioni, and H. Kautz. Toward a Theory of Natural Languages Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157, Miami, Florida, January 2003.

[Rus04] V. Rus. A First Evaluation of Logic Form Identification Systems. In *Proceedings of the 3rd International Workshop on Evaluation of Systems for Semnatic Analysis for Text*, pages 37–40, Barcelona, Spain, July 2004.

Chapter 4

Resolving Under-constrained and Over-constrained Systems of Conjunctive Constraints for Service Requests

Abstract

Given a service request such as scheduling an appointment or purchasing a product, it is possible that the invocation of the service results in too many solutions that all satisfy the constraints of the request or in no solution that satisfies all the constraints. When the invocation results in too many solutions or no solution, a resolution process becomes necessary for agreeing on one of the solutions or finding some agreeable resolution. We address this problem by imposing an ordering over all solutions and over all near solutions. This ordering provides a way to select the best-m with dominated solutions or dominated near solutions eliminated. Further, we provide an expectation-based resolution process that can take the initiative and either elicit additional constraints or suggest which constraints should be relaxed. Experiments with our prototype implementation show that this resolution process correlates substantially with human behavior and thus can be effective in helping users reach an acceptable resolution for their service requests.

Keywords: Service requests, under-constrained systems of constraints, over-constrained systems of constraints, ordered solutions and near solutions, dominance, expectation-based resolution.

4.1 Introduction

We described in a previous paper [AMEL05] a system that allows users to specify service requests and invoke services. This approach is strongly based on conceptual modeling and supports a particular type of service whose invocation involves establishing an agreed-upon relationship in the conceptual model. Examples of these types of services include scheduling appointments, setting up meetings, selling and purchasing products, making travel arrangements, and many more.[1] It is possible that the invocation of service requests for any of these services results in too many satisfying solutions or in no solution at all although there may be near solutions.

In our approach users can specify services such as the following request for scheduling an appointment with a dermatologist.

> I want to see a dermatologist on the 20th, 1:00 PM or after. The dermatologist should be within 5 miles from my home and must accept my IHC insurance.

Our approach uses conceptual-model-based information extraction to map service requests to a domain ontology. This mapping transforms the service request into a formal representation, which consists of concepts along with relationships among these concepts and constraints over the values of these concepts in a domain ontology. Figure 4.1 shows the formal representation of the appointment request as a conjunctive predicate calculus statement—we added some comments prefixed with "//" to provide more readability and to correlate the request with the predicate calculus statement. To resolve the appointment request, the system tries to instantiate each variable in the formal representation with values such that all the constraints are satisfied. The values come from a databases associated with the domain ontology, or are extracted from the service request, or are obtained interactively from users.[2]

[1] We intend the word "service" to be thought of in accordance with its typical meaning—"an act of assistance or benefit." Technically, we define a very special type of service (as described herein). We do not intend our services to be thought of in other technical ways such as registering services with a broker so that they can be found by expressing their functionality in terms of inputs, outputs, and capabilities.

[2] The details of producing formal representations and instantiating them are not the focus of this paper and can be found elsewhere [AMEL05].

$$
\begin{aligned}
&//I\ want\ to\ see\ a\ dermatologist\\
&Appointment(x_0)\ is\ with\ Dermatologist(x_1) \wedge Appointment(x_0)\ is\ for\ Person(x_2)\\
&\\
&//on\ the\ 20th\\
&\wedge Appointment(x_0)\ is\ on\ Date(\text{“}the\ 20th\text{”})\\
&\\
&//1{:}00\ PM\ or\ after\\
&\wedge Appointment(x_0)\ is\ at\ Time(x_3) \wedge TimeAtOrAfter(x_3,\ \text{“}1{:}00\text{”})\\
&\\
&//within\ 5\ miles\ from\ my\ home\\
&\wedge Dermatologist(x_1)\ is\ at\ Address(x_4) \wedge Person(x_2)\ is\ at\ Address(x_5)\\
&\quad \wedge LessThanOrEqual(DistanceBetween(x_4, x_5),\ \text{“}5\text{”})\\
&\\
&//accept\ my\ IHC\ insurance\\
&\wedge Dermatologist(x_1)\ accepts\ Insurance(\text{“}IHC\text{”})
\end{aligned}
$$

Figure 4.1: The predicate calculus statement for the appointment request.

A *solution* for a request is an instantiation for all the variables that satisfies all the constraints. A *near solution* is an instantiation for all the variables that satisfies a proper subset (maybe empty) of the constraints and, in a way to be made precise later, comes close to satisfying the constraints not (yet) satisfied. Ideally, our system would find just one solution or would find a handful of solutions from which a user could select a desired one. More typically, however, our system may return no solution or too many solutions. When our system returns no solution, the request is *over-constrained*, and when it returns too many solutions, the request is *under-constrained*.

A resolution for over-constrained requests is to offer the best-m near solutions. Figure 4.2 shows three near solutions for our appointment request. Both s_1 and s_2 violate the date and distance constraints at different degrees in the sense that s_1 is closer to the 20th and violates the distance constraint less than s_2. Consequently, it is reasonable to impose a greater penalty on s_2 than on s_1. Further, the penalty provides a way to recognize dominated near solutions. Near solution s_1 dominates near solution s_2 because s_1 has less of a penalty for each violated constraint. Penalties provide a way to offer the best-m near solutions by ordering the near solutions based on their penalties and discarding the dominated ones. Additionally, suggesting constraints for users to relax provides another way to offer the best-m near solutions. For instance, if prior appointment requests reveal that users are more likely to impose constraints on date and time than on distance, it makes sense to suggest that users relax constraints on distance. Thus, for example, the resolution

	Date	Time	Distance
s_1	the 21th	1:00 PM	6 miles
s_2	the 22th	1:30 PM	8 miles
s_3	the 20th	2:20 PM	20 miles

Figure 4.2: Near solutions for the appointment request.

	Make	Price	Year	Mileage
s_1	Dodge	$13,999	2005	15,775 miles
s_2	Dodge	$13,999	2004	30,038 miles

Figure 4.3: Solutions for the car purchase request.

process can suggest the relaxation of the constraint on distance and possibly offer s_3 as the best near solution in Figure 4.2.

A resolution for under-constrained requests is to offer the best-m solutions. Consider, for example, the following request for a car purchase.

> I want to buy a Dodge, a 2002 or newer. The mileage should be less than 80,000, and the price should not be more than $15,000.

For this request, www.cars.com offered 168 solutions when probed in November 2005, two of which are in Figure 4.3. Presenting all the solutions or m arbitrarily chosen ones to users is not likely to be very helpful. A way to reduce the number of solutions and offer the best-m solutions is to elicit additional constraints. If prior car purchase requests reveal that users often impose constraints on the car model, for example, it makes sense that a resolution process elicits a constraint on the model of the car. In addition, some solutions satisfy constraints better than others. As Figure 4.3 shows, s_1 better satisfies the year constraint than s_2 because the car in s_1 is newer. Therefore, we can grant s_1 a reward for better satisfying the request. Further, the reward can provide a way to recognize dominated solutions. As Figure 4.3 shows, the solution s_2 is dominated by s_1 because the car in s_1 is newer and has less mileage although both have the same price. Rewards provide a way to offer the best-m solutions by ordering the solutions in a decreasing order based on their rewards and discarding the dominated ones.

This paper offers ways to handle under-constrained and over-constrained service requests. First, the paper offers an expectation-based process for eliciting additional con-

straints for under-constrained requests and for suggesting some constraints for users to relax for over-constrained requests. Second, the paper offers an ordering over solutions and an ordering over near solutions, and a selection mechanism based on Pareto optimality [Par97, Fel80], developed in the late 1800's, to choose the best-m, with dominated solutions or dominated near solutions discarded.

We present these contributions as follows. Section 4.2 discusses an extension to constraints that allows for ordering solutions based on the degree of satisfiability and for ordering near solutions based on how close they are to satisfying the constraints. For under-constrained requests, Section 4.3 introduces expectation declarations as domain knowledge and proposes an expectation-based process to select concepts for which to elicit constraints. In addition, we define an ordering of solutions based on the extension to constraint satisfaction introduced in Section 4.2 and use it along with Pareto optimality to select the best-m solutions. For over-constrained requests, Section 4.4 shows how to define an ordering over near solutions and use it along with Pareto optimality to select the best-m near solutions. It also introduces an expectation-based process to suggest constraints for users to relax. We evaluate our proposed techniques in Section 4.5, and give concluding remarks and directions for future work in Section 4.6.

4.2 Constraints

A *constraint* is an n-place predicate, which for a tuple t of n values evaluates to either *true* or *false* depending on whether t satisfies or violates the constraint. This *true-false* binary view of a constraint allows us to only differentiate tuples based on whether they satisfy or violate a constraint. Researchers have extended this view to differentiate between tuples that violate a constraint by assigning to these tuples increasing positive real numbers that represent different degrees of violation [LHL97, Arn02]. Although this extension allows for distinguishing between tuples that violate a constraint, it does not allow for distinguishing between tuples that satisfy a constraint because this extension lacks the notion of degree of satisfiability. A constraint evaluates to zero for all tuples that satisfy that constraint, which means all the tuples necessarily have the same degree of satisfiability. We, therefore,

further extend the binary view to not only consider degree of violation, but also to consider degree of satisfiability by granting tuples increasing rewards based on how well they satisfy a constraint.

Definition 1 *Let C be an n-place constraint and let D_i be the domain of the i^{th} place of C, $1 \leq i \leq n$. A constraint is a function $C : D_1 \times ... \times D_n \longrightarrow \mathcal{R}$ that maps a tuple $t = \langle v_1, ..., v_n \rangle \in D_1 \times ... \times D_n$ to a real number in $\mathcal{R}$. An evaluation of the constraint C on a tuple t is defined as $C(t) = \alpha$, where $\alpha \in \mathcal{R}^+ \cup \{0\}$, which is a positive real number $\mathcal{R}^+$ or zero, is the value of the evaluation if t satisfies C, and $C(t) = \beta$, where $\beta \in \mathcal{R}^-$, which is a negative real number $\mathcal{R}^-$, is the value of the evaluation if t violates C.*

The value α in Definition 1 represents the *reward* granted to a tuple t for satisfying a constraint C. A higher value for the reward α denotes greater satisfaction. The value β represents the *penalty* imposed on a tuple t for violating the constraint. A lower negative value for α denotes a greater degree of violation. Observe that in Definition 1, we try to capture the intuitive idea behind a reward and a penalty by letting the reward be a non-negative real number (rewards are positive) and the penalty be a negative real number (penalties are negative).

Designers should make domain decisions about the amount of a reward α and a penalty β. For instance, in a car purchase domain, designers may give a greater reward for newer cars. Therefore, they may define the evaluation for a constraint on a year in which a car was made such as "a 2000 or later" as $\geq(y, 2000) = y - 2000$. Observe that a 2001 car has a reward of 1 and a 2002 car has a reward of 2, which means that a 2002 car has a greater satisfiability degree according to this evaluation. Also observe that a 1999 car has a penalty of -1 and a 1980 car has a penalty of -20, which means that a 1999 car has much less of a penalty than a 1980 car.

An evaluation function can also impose a fixed penalty when ordering between values is not obvious. As an example, a constraint of the form "Brand = Canon" on digital camera brands can be defined as

$$BrandEqual(x, \text{“}Canon\text{”}) = \begin{cases} 0, & \text{if } x = \text{“}Canon\text{”}; \\ -1, & \text{otherwise} \end{cases}$$

We imposed a fixed penalty for any brand other than "Cannon", as Arnal suggested [Arn02], because it is not obvious how we can order penalties between brands other than "Cannon".

For equality constraints over which a penalty ordering is possible, designers can declare penalties. For instance, a designer may choose the evaluation for *EqualAppointmentTime(t, 10:00 AM)* to be $-(f(t) - f(10{:}00\ AM))^2$, where f is a function that converts a time to a unitless number. For example, the time 2:15 PM, which is the military time 14:15, could be converted to the integer 1415. For illustration purposes, we have assumed that the designer has chosen to square the difference to give proportionally less of a penalty to times close to 10:00 AM.

4.3 Under-constrained Service Requests

Under-constrained service requests admit too many solutions. In this section, we discuss two ways to provide users with the best-m solutions out of n solutions. First, we propose an expectation-based elicitation process to elicit additional constraints and apply them to solutions. Applying additional constraints to solutions may reduce the number of solutions and may also make the resulting solutions more desirable [SL01, FPTV04]. Second, we propose an ordering over solutions based on our extension for constraints in Definition 1 along with Pareto optimality based on this ordering to select the best-m solutions.

4.3.1 Constraint Elicitation Using Expectations

We associate expectations with concepts of a domain ontology. An *expectation* is the probability that value(s) for a concept appear in a service request. The expectation is, therefore, a number in the interval [0, 1], where the low and high extremes of the interval mean, respectively, that a value for the concept is not and is certainly expected to appear in a service request. Values in the open interval (0, 1) represent varying degrees of expectations.

Domain ontology designers estimate the expectations associated with concepts. Although there may be several ways to estimate the expectations, we suggest two general ways. First, designers can estimate the expectation using their knowledge of the domain. Second, designers can analyze service requests in the domain of the ontology and count

the frequency of appearance for each concept in the domain ontology. Further, this latter method leads to the possibility that the expectations can be adjusted as the system runs.

Unlike other approaches to constraint elicitation (e.g. [LHL97, SL01, PFK03]), which are built on an assumption that users can impose additional constraints if they review some examples of solutions, we let the resolution process take the initiative and suggest the concepts on which to impose constraints according to the associated expectations with these concepts. The intuitive idea is that the resolution process can order the concepts based on their associated expectations and make reasonable suggestions to users to constrain concept values, starting from the concept associated with the highest expectation for which there is, as of yet, no constraint.

The elicitation process terminates when one of the following three conditions holds. First, the most recent elicited constraint is unsatisfiable in which case the service request becomes over-constrained and the resolution process uses the techniques in Section 4.4 to handle this situation. Second, the solution space is reduced to m or fewer solutions, in which case the system offers these solutions to users to evaluate and choose one. Third, there is no other concept in the ordering of concepts associated with an expectation that exceeds a prespecified threshold.

To demonstrate the idea of constraint elicitation using expectations, note that the car purchase request in Section 4.1 does not specify a constraint on the model of the car. Assuming that the expectation associated with *Model*, say 0.6, is the highest among the unconstrained concepts and is above the threshold, say 0.5, the resolution process suggests that the user could impose a constraint on the model. If a user wishes to constrain *Model* to be "Stratus" the resolution process can restrict the solutions to Dodge Stratuses.

4.3.2 Selecting the Best-m Solutions

Our extension to the binary view of constraints (Definition 1) provides a way to impose an ordering over solutions based on rewards granted to each solution for satisfying the service request constraints. Let $S = \{s_1, ..., s_n\}$ be a set of solutions each of which satisfies every constraint in the set of constraints $C = \{C_1, ..., C_k\}$, which are imposed on a service request.

The evaluation of the set of constraints C for a solution $s_i \in S$ returns a set of real numbers $\{C_1(s_i), ..., C_k(s_i)\}$, which are the rewards granted to s_i for satisfying the constraints.

Before computing an aggregate reward for a solution s_i over all constraints in C, we first divide each reward $C_j(s_i),\ 1 \leq j \leq k$, by $max_{1 \leq i \leq n} C_j(s_i)$, the maximum reward value over all solutions for constraint C_j. This normalizes the rewards to the interval [0, 1]. The purpose of the normalization is to discard the relative effects of large magnitude rewards across different constraints and thus to make it unnecessary to correlate values across different constraints. Let us denote the set $\{C_1(s_i), ..., C_k(s_i)\}$ after doing the normalization by $C^* = \{C_1^*(s_i), ..., C_k^*(s_i)\}$. Researchers have suggested several ways to compute combined evaluations (see [MA04] for a thorough survey). We linearly combine rewards in C^* yielding a combined reward ρ for a solution s_i as follows:

$$\rho_{C^*}(s_i) = \sum_{j=1}^{k} C_j^*(s_i); for\ i = 1, ..., n.$$

Definition 2 *Let s_i and s_j be two solutions and $C = \{C_1, ..., C_k\}$ be a set of constraints. We say that s_i is better than or equivalent to s_j, $s_i \succeq_\rho s_j$, with respect to C if $\rho_{C^*}(s_i) \geq \rho_{C^*}(s_j)$.*

To demonstrate the idea of reward-based ordering, let us suppose that we have a set of constraints $C = \{\leq$*(mileage, "30,000 miles"), $\leq$(price, "\$20,000")*$\}$ and two solutions $s_1 =$ {mileage = "29,000 miles", price = "\$19,000"} and $s_2 =$ {mileage = "29,900 miles", price = "\$18,000"}, then designers might decide to grant a reward of 1000 for s_1 and of 100 for s_2 for satisfying the mileage constraint, and a reward of 1000 for s_1 and a reward of 2000 for s_2 for satisfying the price constraint. Given these rewards, we can normalize them to [0, 1] by dividing the mileage rewards by 1000 and the price rewards by 2000, yielding the normalized rewards 1 and 0.1 for s_1 and s_2 respectively for satisfying the mileage constraint and the normalized rewards 0.5 and 1 for s_1 and s_2 respectively for satisfying the price constraint. Based on Definition 2, $s_1 \succeq_\rho s_2$ because $\rho_{C^*}(s_1) = 1.5$ and $\rho_{C^*}(s_2) = 1.1$.

The ordering $\succeq_\rho$ sorts the solutions according to their combined rewards from the solution with the highest combined reward to the lowest. (Any solutions with identical rewards appear in a random order within their own equality group.) Although this ordering does sort the solutions, it does not necessarily imply that the first m solutions are the best-m

solutions. The sorting procedure considers only the combined rewards, but does not consider the rewards granted to the solutions for satisfying each individual constraint. The rewards of the individual constraints, C_1, ..., C_k, in C provide additional knowledge to differentiate among solutions based on Pareto optimality, which divides solutions into dominating and dominated solutions based on a dominance relation.

Definition 3 *Let $C = \{C_1, ..., C_k\}$ be a set of constraints and $S = \{s_1, s_2, ..., s_n\}$ be a set of solutions. Let s_i, $s_j \in S$ be any two distinct solutions, we say that s_i dominates s_j if $\forall_{p \in \{1, \ ..., \ k\}}(C_p(s_i) \geq C_p(s_j))$ and $\exists_{q \in \{1, \ ..., \ k\}}(C_q(s_i) > C_q(s_j))$.*

Definition 3 says that the solution s_i, which dominates s_j, has rewards from all the constraints that are at least equal to the rewards for s_j and for at least one of the constraints s_i has a strictly higher reward. Observe the that Definition 3 does not explicitly consider the combined reward $\rho_{C^*}(s_k)$. However, the combined reward is implicit in this definition in the sense that a solution can never dominate another solution with a higher combined reward.

Definition 3 provides the basis for our variation of Pareto optimality, a concept which Pareto defined over a century ago [Par97].

Definition 4 *Let $S = \{s_1, s_2, ..., s_n\}$ be a set of solutions for a service request. A solution $s_i \in S$ is said to be Pareto optimal if there does not exist an $s_j \in S$ such that s_j dominates s_i.*

The key idea in Definition 4 is that a solution cannot be Pareto optimal if it is dominated by another solution.

4.3.3 Resolution of Under-constrained Requests

To demonstrate our resolution procedure, consider our request for a Dodge (in the introduction). The system first uses expectations to elicit additional constraints to reduce the number of solutions. Since the request does not constrain the model of the car and the expectation associated with the model is the highest among all the unconstrained concepts, the system suggests that the user constrains the model. Adding the constraint that the

Solution	Make	Model	Price	Year	Mileage	$\rho_{C^*}(s_i)$	Pareto Optimal
s_1	Dodge	Stratus	13,999.00	2005	15,775	2.499	$\surd$
s_2	Dodge	Stratus	11,998.00	2004	23,404	2.497	$\surd$
s_3	Dodge	Stratus	14,200.00	2005	16,008	2.476	$\times$
s_4	Dodge	Stratus	14,557.00	2005	16,954	2.431	$\times$
s_5	Dodge	Stratus	10,590.00	2003	38,608	2.360	$\surd$
s_6	Dodge	Stratus	14,253.00	2004	17,457	2.332	$\times$
s_7	Dodge	Stratus	10,987.00	2004	56,377	2.267	$\surd$
s_8	Dodge	Stratus	13,999.00	2004	30,038	2.230	$\times$
s_9	Dodge	Stratus	12,995.00	2004	40,477	2.226	$\times$
s_{10}	Dodge	Stratus	12,577.00	2003	33,163	2.216	$\times$
s_{11}	Dodge	Stratus	14,620.00	2004	32,406	2.149	$\times$
s_{12}	Dodge	Stratus	8,975.00	2003	75,689	2.140	$\surd$

Figure 4.4: Solutions for the car purchase request.

model be a "Stratus" drops the number of solutions to 53, which is still too many. Since there are no more concepts with an expectation higher than the threshold, 0.5, the system uses the ordering $\succeq_\rho$ and Pareto optimality to return the best-m solutions. Figure 4.4 shows the top 12 solutions ordered in ascending order based on their combined rewards $\rho_{C^*}(s_i)$. The rightmost column in Figure 4.4 shows whether a solution is Pareto optimal ($\surd$) or not ($\times$). For instance, the solution s_3 is not Pareto optimal because s_1 dominates it—s_1 is cheaper and has a lower mileage, although both have the same year. Since we have chosen $m = 5$, the system returns the first five Pareto optimal solutions, s_1, s_2, s_5, s_7, and s_{12}.

4.4 Over-constrained Service Requests

Over-constrained service requests admit no solution. As in Section 4.3, we discuss two ways to provide the best-m near solutions. First, we propose an ordering over near solutions and use it along with Pareto optimality to offer the best-m near solutions. Second, we propose an expectation-based relaxation process that suggests unsatisfied constraints for a user to relax.

4.4.1 Ordering Near Solutions

We combine the penalties and rewards, if any, of each near solution, and order the near solutions according to their combined penalties and rewards. Let $S = \{s_1, ..., s_n\}$ be a set of near solutions each of which violates one or more constraints from a set of constraints

$C = \{C_1, ..., C_k\}$. The evaluation of a set of constraints C for a near solution $s_i \in S$ returns a set of real numbers $\{C_1(s_i), ..., C_k(s_i)\}$, where each $C_k(s_i)$ is either a reward or a penalty. We divide these real numbers $C_j(s_i),\ 1 \leq j \leq k$ by $\underset{1 \leq i \leq n}{max} |C_j(s_i)|$, the maximum absolute reward or penalty value over all near solutions for constraint C_j. This normalizes the rewards and penalties to the interval [-1, 1]. Let us denote the set $\{C_1(s_i), ..., C_k(s_i)\}$ after normalization by $C^* = \{C_1^*(s_i), ..., C_k^*(s_i)\}$. We combine each $C_j^*(s_i)$ in C^* linearly, as before, yielding a combined penalty/reward ϕ for each near solution s_i as follows:

$$\phi_{C^*}(s_i) = \sum_{j=1}^{k} C_j^*(s_i); for\ i = 1, ..., n.$$

Greater values of $\phi_{C^*}(s_i)$ indicate lower penalties on s_i and (possibly) higher rewards. Thus, a high value of $\phi_{C^*}(s_i)$ denotes a better near solution s_i.

Definition 5 *Let s_i and s_j be two distinct near solutions and $C = \{C_1, ..., C_k\}$ be a set of constraints. We say that s_i is better than or equivalent to s_j, $s_i \succeq_\phi s_j$, with respect to C if $\phi_{C^*}(s_i) \geq \phi_{C^*}(s_j)$.*

We define a dominance relation and Pareto optimality based on the ordering $\succeq_\phi$ in Definition 5 in the same way as we defined them in Definitions 3 and 4.

4.4.2 Constraint Relaxation Using Expectations

For constraint relaxation we use the same expectation values for constraints as discussed in Subsection 4.3.1, but consider the lowest expectation values, rather than the highest, to be the candidates for relaxation. In addition, we consider the violation degree when we suggest constraints for relaxation. For instance, it is likely to be better to suggest relaxing a time constraint violated by 10 minutes than to suggest relaxing a distance constraint violated by 50 miles even though a distance constraint is likely to be associated with a lower expectation value. Further, since we should not badger the user with questions, the number of suggested unsatisfied constraints should not exceed a prespecified threshold. Taking all these ideas into consideration, the system selects the constraints to suggest for relaxation based on the following procedure.

1. To avoid overloading the user with suggestions, select only near solutions that violate fewer constraints than a prespecified threshold.
2. To take the expectation values into account, compute the cost of the relaxation for each near solution based on the expectation using the equation $r(s_i) = \sum_k e_k C_k^*(s_i)$, where e_k is the expectation value associated with the constraint C_k and $C_k^*(s_i)$ is the normalized penalty imposed on s_i for C_k.
3. To take the overall degree of violation into account, select the near solution s_i with the lowest absolute value of $r(s_i)$ and suggest relaxing the constraints that s_i violates only to the degree necessary to satisfy the constraints of s_i.

We give an example in the next subsection.

4.4.3 Resolution of Over-constrained Requests

To demonstrate our resolution procedure, consider our request for an appointment (in the introduction). Figure 4.5 shows 8 near solutions for the request ordered in ascending order based on the combined penalty/reward $\phi_{C^*}(s_i)$, which appears in the second column from the right. The system tries first to suggest some constraints to relax using the expectations associated with the constraints. Figure 4.6 shows the constraints along with their associated expectation values and their rewards/penalties for each near solution. The rightmost column in Figure 4.6 shows the computed relaxation cost $r(s_i)$ for each near solution. Based on our relaxation procedure, the system could consider the near solution s_4 for suggesting relaxation because it has the lowest relaxation cost $r(s_i)$. The system does not, however, because s_4 violates three constraints, which exceeds the threshold we set, namely fewer than three constraints. The near solution s_3 satisfies our procedure requirements in the sense that s_3 violates two constraints and has the next lowest relaxation cost $r(s_i)$. The system therefore suggests letting the time be 12:40 PM instead of 1:00 PM and letting the date be the 19th instead of the 20th. If the user accepts these relaxed constraints, the system can offer s_3 as the best solution.

For the sake of further discussing the possibilities, we assume that the user does not accept the suggestion to relax the time and date constraints. To compute the best-m near

Near Solution	Insurance	Distance	Time	Date	$\phi_{C*}(s_i)$	Pareto Optimal
s_1	IHC	16	1:00 PM	the 19th	-0.160	$\surd$
s_2	IHC	18	1:10 PM	the 19th	-0.180	$\times$
s_3	IHC	4	12:40 PM	the 19th	-0.257	$\surd$
s_4	IHC	6	12:50 PM	the 19th	-0.264	$\surd$
s_5	IHC	20	3:00 PM	the 19th	-0.271	$\times$
s_6	IHC	8	1:40 PM	the 18th	-0.382	$\surd$
s_7	IHC	18	2:20 PM	the 22nd	-0.479	$\times$
s_8	IHC	3	11:30 AM	the 16th	-1.049	$\surd$

Figure 4.5: Near solutions for the appointment request.

	Insurance="IHC" Expectation=0.4	Distance$\leq$ 5 Expectation=0.3	Time$\geq$("1:00 PM") Expectation=0.8	Date="the 20th" Expectation=0.9	$r(s_i)$
s_1	0.000	-0.076	0.167	-0.250	-0.248
s_2	0.000	-0.090	0.160	-0.250	-0.252
s_3	0.000	0.007	-0.014	-0.250	-0.236
s_4	0.000	-0.007	-0.007	-0.250	-0.233
s_5	0.000	-0.102	0.083	-0.250	-0.256
s_6	0.000	-0.021	0.139	-0.500	-0.456
s_7	0.000	-0.090	0.111	-0.500	-0.477
s_8	0.000	0.014	-0.062	-1.000	-0.950

Figure 4.6: Rewards and penalties for the near solutions.

solutions, the system sorts the near solutions based on the combined penalty/reward $\phi_{C^*}(s_i)$ and discards the dominated near solutions using the rewards and penalties information in Figure 4.6, i.e. $\phi_{C^*}(s_1) = -0.160 = -0.076 + 0.167 - 0.250$; $\phi_{C^*}(s_2) = -0.180 = -0.090 + 0.160 - 0.250$; and so forth. The rightmost column in Figure 4.5 shows whether a near solution s_i is Pareto optimal ($\surd$) or not ($\times$). Since $m = 5$, the system returns the first 5 Pareto optimal near solutions, which in our example are s_1, s_3, s_4, s_6, and s_8.

A closer look at the results in Figures 4.5 and 4.6 reveals that the returned near solutions are better than the ones filtered out. For instance, comparing the near solution s_1 to the discarded near solution s_2, we find that although both violate the date constraint to the same degree and satisfy the time constraint, s_1 violates the distance constraint less than s_2 and is closer to the requested time, 1:00 PM. Therefore, from the Pareto-optimality's viewpoint, given s_1 as a possibility, no user is likely to accept the near solution s_2.

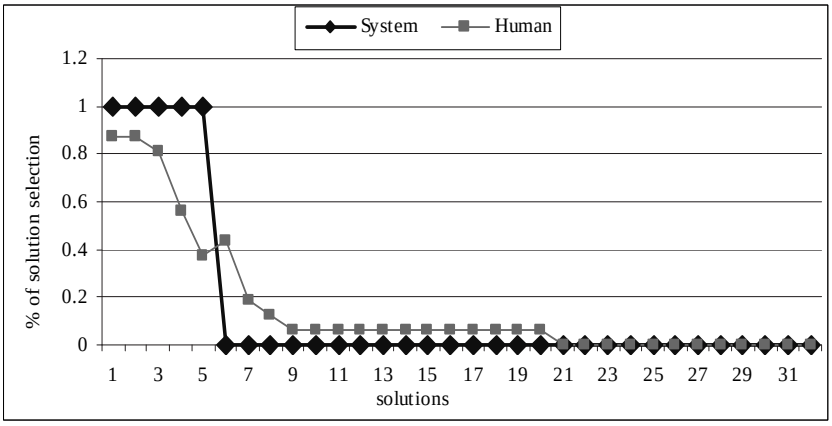


Figure 4.7: Human solution selection compared to system solution selection.

4.5 Performance Analysis

To evaluate the performance of our system, we conducted a user study. The goal was to test whether there is a statistically significant difference between human choices and system choices. The subjects in our study were from both genders and from different academic disciplines and education levels—professors, graduate students, and undergraduate students at Brigham Young University. We gave every subject a request from a car purchase domain along with 32 cars that each satisfies all the constraints of the request, and another request from an appointment scheduling domain along with 19 near solutions that each satisfies some but not all the constraints of the appointment request. All the solutions and near solutions were randomly shuffled so as not to provide the subjects with any ordering information. We asked each subject to select and order the best-5 solutions out of 32 solutions for cars and the best-5 near solutions out of 19 near solutions for appointments.

To visualize the degree of agreement between system choices and human choices, we counted the number of times each solution was chosen by the 16 subjects for the car experiment and the number of times each near solution was chosen by the 15 subjects for the appointment experiment.[3] Figures 4.7 and 4.8 show the percentage of human subjects who chose each solution or near solution respectively. The first five solutions and near solutions are the ordered best-5 Pareto optimal solutions and near solutions. The remaining solutions

[3]One of the subjects did not make choices for the appointment experiment.

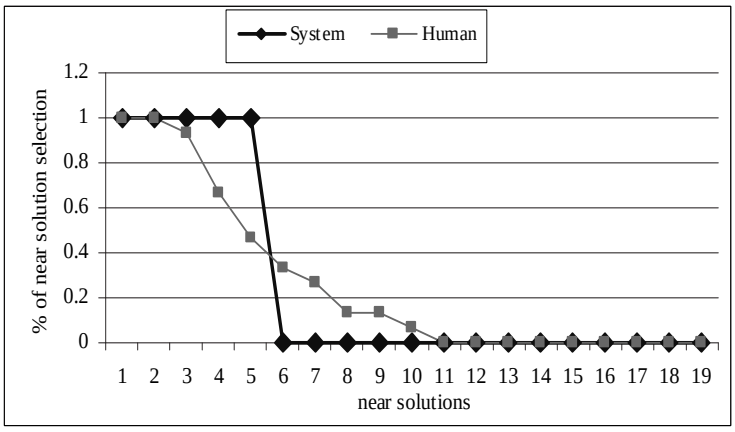


Figure 4.8: Human near solution selection compared to system near solution selection.

System / Human	The best-5 solutions	Not the best-5 solutions	Total
The best-5 solutions	56	24	80
Not the best-5 solutions	24	408	432
Total	80	432	512

Figure 4.9: Human versus system choices for the car experiment.

and near solutions are ordered by decreasing percentage of selection by human subjects. As the figures show there is a high degree of agreement between the system's choices and the human subjects' choices. As Figure 4.7 shows, over 87% of the subjects chose solutions 1 and 2, and over 81% of the subjects chose solution 3. Figure 4.8 shows an even higher degree of agreement. All the subjects chose near solutions 1 and 2, and over 96% of them chose near solution 3. The solutions and near solutions that were not chosen by the system as being among the best-5 were also selected less often by our human subjects, with the exception of solution 6 in Figure 4.7, which was selected by 43% of the human subjects, and the near solutions 6 and 7, which were chosen by the 33% and 26% of the human subjects. Interestingly, the system chose both solution 6 and near solution 6 as the 6^{th} Pareto optimal solution and near solution. Near solution 7, however, is not Pareto optimal. All the other solutions and near solutions were chosen by 20% or fewer of the subjects. Figures 4.7 and 4.8 reveal a definite pattern: human subjects chose a high percentage of the best-5 choices and a low percentage for choices not among the best-5 system choices.

Human \ System	The best-5 near solutions	Not the best-5 near solutions	Total
The best-5 near solutions	61	14	75
Not the best-5 near solutions	14	196	210
Total	75	210	285

Figure 4.10: Human versus system choices for the appointment experiment.

Agreement index	Type of agreement	Car experiment	Appointment experiment
P_o	overall	0.91	0.90
P_{pos}	the best-5	0.70	0.81
P_{neg}	not the best-5	0.94	0.93
P_e	due to chance	0.73	0.61
Cohen kappa κ	chance corrected	0.67	0.74
95% Confidence interval for κ		[0.58, 0.76]	[0.65, 0.83]

Figure 4.11: Statistical summary.

To statistically measure the degree of agreement between system choices and human subjects choices, we ran an inter-observer agreement test [LK77] using the MINITAB 14 software package [min05]. The inter-observer agreement per observer pair (system and human) was determined with respect to the dichotomy: the best-5 solutions or the best-5 near solutions and not the best-5. Figures 4.9 and 4.10 show the distribution of agreement and disagreement between the system and our human subjects. We disregarded the order in which each subject ordered the best-5 solutions or near solutions, and tallied the number of solutions and near solutions chosen by subjects that belong to the best-5 solutions and the best-5 near solutions selected by the system. We also tallied the number of solutions and near solutions that were not chosen by the system and the subjects as the best-5. For instance, the 16 subjects for the car experiment made 80 choices of which 56 belong to the best-5 system choices and 24 do not. Further, of the 432 solutions not chosen, 24 were among the best-5 system choices while 408 were also not chosen by the system. Figure 4.11 shows the statistical summary for the car and appointment experiments. The overall agreement, P_o, and the agreement due to chance, P_e, for the car experiment are 0.91 and 0.73 respectively with a Cohen kappa κ value of 0.67, and for the appointment experiment are 0.90 and 0.61 with a κ value of 0.74. Based on the Landis-Koch interpretation for κ values [LK77],

the two κ values indicate "substantial" agreement between the system and the subjects. The 95% confidence intervals for κ in Figure 4.11, however, indicate that the agreement may range from "moderate" (0.58) to "substantial" (0.76) for the car experiment and from "substantial" (0.65) to "almost perfect" (0.83) for the appointment experiment. It is useful also, as suggested in [CF90], to compute two more indices, namely the positive agreement P_{pos} on the best-5 and the negative agreement P_{neg} on those not among the best-5. The positive agreement, P_{pos}, for the car experiment and for the appointment experiment were respectively 0.70 and 0.81 whereas the negative agreement, P_{neg}, were respectively 0.94 and 0.93. All these numbers show a high agreement between the system and human subjects on both the best-5 and not among the best-5 (near) solutions. We next considered how the system and each subject ordered the best-5 solutions and near solutions. The κ values for the car experiment was 0.43 and for the appointment experiment was 0.61, indicating respectively "moderate" and "substantial" agreement between system ordering and subject ordering for the best-5 solutions and the best-5 near solutions.

4.6 Conclusions and Future Work

We proposed techniques to handle under-constrained and over-constrained systems of conjunctive constraints for service requests. These techniques depend on defining an ordering over the solutions or near solutions along with Pareto optimality to discard dominated solutions or near solutions. From among the ordered Pareto optimal solutions or near solutions, we select the best-m. We also introduced expectation values as domain knowledge and proposed an expectation-based process to elicit or relax constraints respectively for under-constrained and over-constrained requests. We conducted experiments to test our proposed ordering and Pareto optimality techniques and found substantial agreement between the system and human behavior.

Although still preliminary, the results are promising. As future work, we plan to do more user studies on additional domains with a larger number of subjects. In addition, we need to develop a dialog generation system for user interaction and to conduct a field

test for the generated dialog. Finally, we should integrate our resolution techniques into a service request architecture, such as the semantic web.

Acknowledgements

This work is supported in part by the National Science Foundation under grants 0083127 and 0414644.

We appreciate Del T. Scott from the Department of Statistics, Brigham Young University, for his help with our statistical analysis. We also appreciate the help of all the subjects who participated in the experiments.

Bibliography

[AMEL05] M. J. Al-Muhammed, D. W. Embley, and S. W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 288–303, Klagenfurt, Austria, October 2005.

[Arn02] M. T. Arnal. *Scalable Intelligent Electronic Catalogs*. PhD Dissertation, Swiss Federal Institute of Technology in Lausanne (EPFL), 2002.

[CF90] D. Cicchetti and A. Feinstein. High Agreement But Low Kappa. II. Resolving The Paradoxes. *Journal of Clinical Epidemiology*, 43(6):551–558, 1990.

[Fel80] A. M. Feldman. *Welfare Economics and Social Choice Theory*. Kluwer, Boston, Massachusetts, 1980.

[FPTV04] B. Faltings, P. Pu, M. Torrens, and P. Viappiani. Designing Example-Critiquing Interaction. In *Proceedings of the 9th International Conference on Intelligent User Interface*, pages 22–29, Funchal, Portugal, November 2004.

[LHL97] G. Linden, S. Hanks, and N. Lesh. Interactive Assesment of User Preference Models: The Automated Travel Assistant. In *Proceedings of the 6th International Conference on User Modeling (UM 1997)*, pages 67–78, Vienna, New York, June 1997.

[LK77] J. R. Landis and G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, March 1977.

[MA04] R. T. Marler and J. S. Arora. Survey of Multi-Objective Optimization Methods for Engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, March 2004.

[min05] Minitab 14.2 Statitiscal Software. Website, 2005. www.minitab.com.

[Par97] V. Pareto. *Cours d'économie politique*. F. Rouge, Lausanne, Switzerland, 1897.

[PFK03] P. Pu, B. Faltings, and P. Kumar. User-Involved Tradeoff Analysis in Configuration Tasks. In *Proceedings of the 3rd International Workshop on User-Interaction in Constraint Satisfaction*, pages 85–102, Kinsale, Ireland, September 2003.

[SL01] S. Shearin and H. Lieberman. Intelligent Profiling by Example. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 145–151, Santa Fe, New Mexico, January 2001.

Chapter 5

Bringing Web Principles to Services: Ontology-Based Web Services

Abstract

Researchers are beginning to realize the potential of web services that can use the web as a place for information publication and access as opposed to the traditional web-services paradigm that merely uses the web as a transport medium. Traditional web services can be difficult to discover, can have complex invocation APIs, and require strong coupling between communicating applications. In previous work, we presented ontology-based techniques in which users make service requests using free-form, natural-language-like specifications. This paper shows how we can use these ontological techniques to automatically create ontology-based web services that (1) are easy for software agents to discover because they are created based on machine-processable formalisms (ontologies), (2) have invocation APIs requiring only simple read and write operations, and (3) require no a priori agreements regarding types and data formats between communicating applications. Experiments with our prototype implementation in several domains show that our approach can effectively create web services with these characteristics.

Keywords: Web-principled services, ontology-based web services, semantic web services, web service decoupling, data heterogeneity resolution.

5.1 Introduction

Despite the tremendous success of traditional web services, researchers have recently realized that web services can be even more successful by basing them more fundamentally on web principles [KHP+05, Fen04, SKB06, SKMR+06]. To use a traditional web service, a service requester must discover a service of interest and synchronously communicate with it. A requester must also comply with service-specified data formats and data-exchange protocols. In contrast, *web-principled services* (those fundamentally based on web principles, which we describe in this paper) have none of these requirements.

Figure 5.1 shows a traditional web service that returns a weather report for a given place and time. Assuming a service requester has located this service, the requester would provide a latitude, a longitude, and a date, and may choose the number of days for the forecast and whether the results should be returned in 12-hour or 24-hour increments. The latitude and longitude must be real numbers, not, say, degrees and minutes, and the longitude must be negative, not East or West the longitude. The date must be in the format *yyyy-mm-dd*, not in any of many other possible formats. Further, after clicking on "Submit", the calling application must stay coupled with the service until it returns a response. Figure 5.2, on the other hand, shows the interface for a prototype for web-principled services. Without needing to discover a service, a user posts a request. In our example, the requester enters, "What is the weather going to be like tomorrow in Springfield, Illinois?" A service that can appropriately respond observes the posting and responds. The response is twofold: (1) it highlights the part of the query it "understands" (*weather*, *tomorrow*, and *Springfield*, *Illinois* in Figure 5.2) and (2) it answers the query (*MaximumTemperature*=70, *MinimumTemperature*=38, and *PercentChanceOfPrecipitation*=10 in Figure 5.2).

Web-principled services allow decoupling among communicating applications, but, as a result, require heterogeneity resolution.

- *Decoupling among communicating applications.* In our vision of web-principled services, service requesters post their requests to the web without the requesters having to reference any particular service. Requesters, therefore, do not need to discover these services nor to communicate directly with them. As a result, decoupling of ser-

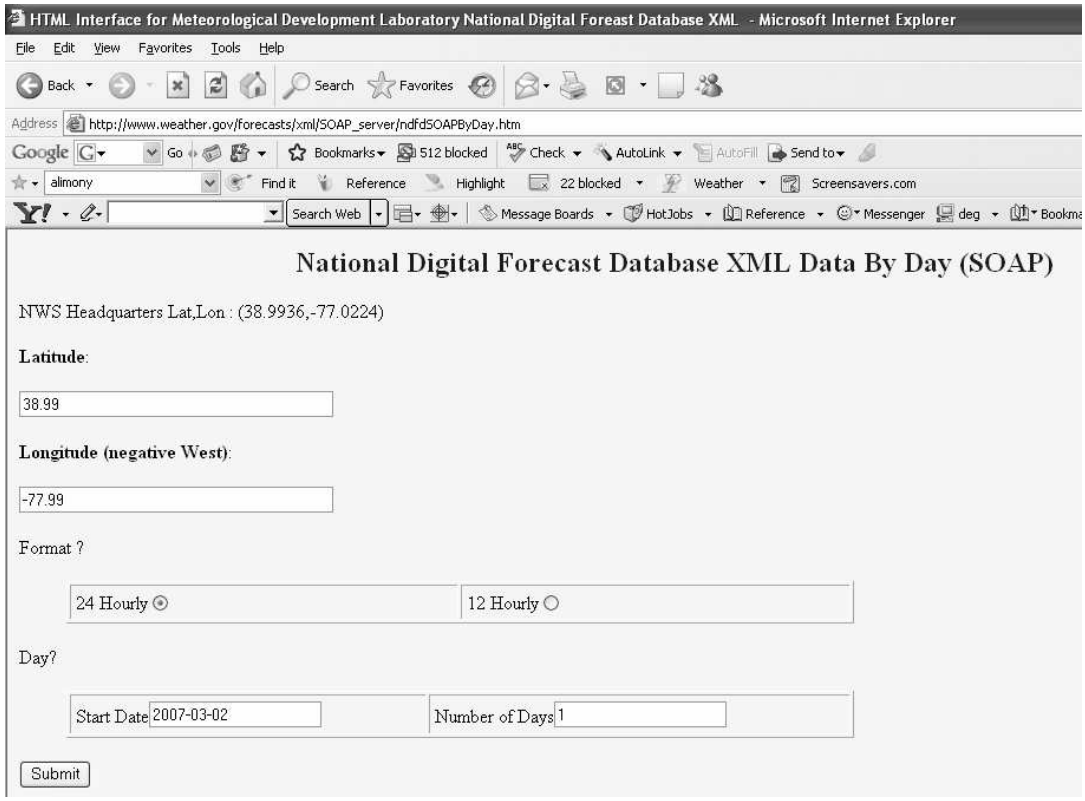


Figure 5.1: A weather report web service.

vices and requesters is achieved. To negate the need for service discovery and direct communication, however, web-principled services must be able to read posted service requests, process them, and return results to requesters.

- *Heterogeneity resolution.* For successful invocation of a service, posted service requests must comply with the service-specified data. Prior agreement between a requester and a service on the data formats, types, and mappings between values in a request and the respective input parameters of a service would enable compliance. Decoupling, however, prevents prior agreement. Therefore, web-principled services must make requests comply with their internal data specification by resolving heterogeneity.

In previous work [AMEL05, AME06, AME07], we have proposed an ontological technique that allows service requesters to invoke certain types of ontology-based services using free-form, natural-language-like specifications. We explain in this paper how these ontology-based services satisfy the decoupling and heterogeneity requirements and thus can

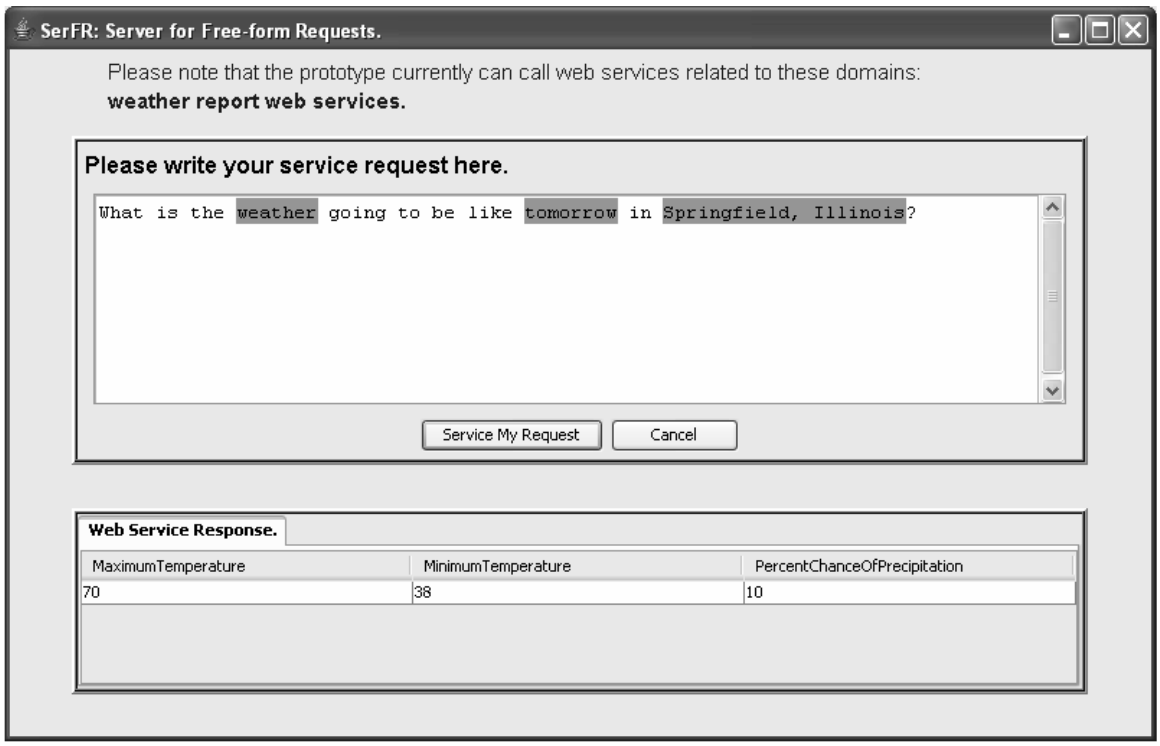


Figure 5.2: A free-form weather report request with all recognized constraints and values highlighted along with the web service response to the request.

enable web-principled services. We call these services *ontology-based web services* (*OBWSs*). OBWSs can interact asynchronously with service requesters. OBWS invocation only requires reading free-form service requests from some resource such as the web. Thus they satisfy the decoupling property because they do not need to have a direct coupling with a service requester. OBWSs require no prior agreement on exchanged data. An OBWS recognizes values in a service request such as the highlighted values in Figure 5.2 and binds them to its input variables. In order for this binding to be successful, the OBWS (1) casts each recognized value to a type of an input variable, (2) transforms the format of each recognized value to an internal format conforming to the format specified by its ontology, and (3) assigns each recognized value to the respective input variable. In addition, an OBWS identifies any missing values necessary for the invocation and obtains them, seeking additional information from the requester, if necessary. All these capabilities of an OBWS in handling a service request enable it to satisfy the heterogeneity property.

We give the details of our contributions to enabling web-principled services via OBWSs as follows. Section 5.2 introduces ontology-based web services. It introduces and gives the necessary details for ontologies, the foundational knowledge for OBWSs, in Sections 5.2.1 and 5.2.2. It then shows how an OBWS services a free-form service request and how it resolves data heterogeneity in Section 5.2.3. Section 5.3 presents the OBWS architecture and shows how this architecture meets the decoupling requirement for web-principled services. Section 5.4 explains how our approach can bring web principles to traditional web services, emphasizing how our techniques provide a way for invoking traditional web services. Section 5.5 compares our approach to related work. In Section 5.6, we give concluding remarks and directions for future work.

5.2 Ontology-Based Web Services

We now discuss the elemental components of ontology-based web services. Section 5.2.1 introduces the semantic data model that we use to represent ontologies, and Section 5.2.2 introduces how we capture the semantics of the instances of different concepts of the semantic data model. Section 5.2.3 shows how OBWSs service requests and how they meet the heterogeneity resolution requirement.

5.2.1 Semantic Data Model

A *semantic data model* specifies named sets of objects, which we call *object sets*, named sets of relationships among object sets, which we call *relationship sets*, and constraints over object and relationship sets. Figure 5.3 shows a domain ontology for providing a weather report. The domain ontology consists of object-set concepts such as *StartDate* and *Place* that can be used to make requests for weather reports. The semantic data model has two types of object sets, those that are lexical (enclosed in dashed rectangles) and those that are nonlexical (enclosed in solid rectangles). An object set is *lexical* if its instances are indistinguishable from their representations. *StartDate* is an example of a lexical object set because its instances (e.g. "May 7, 2007") represent themselves. An object set is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Place* is

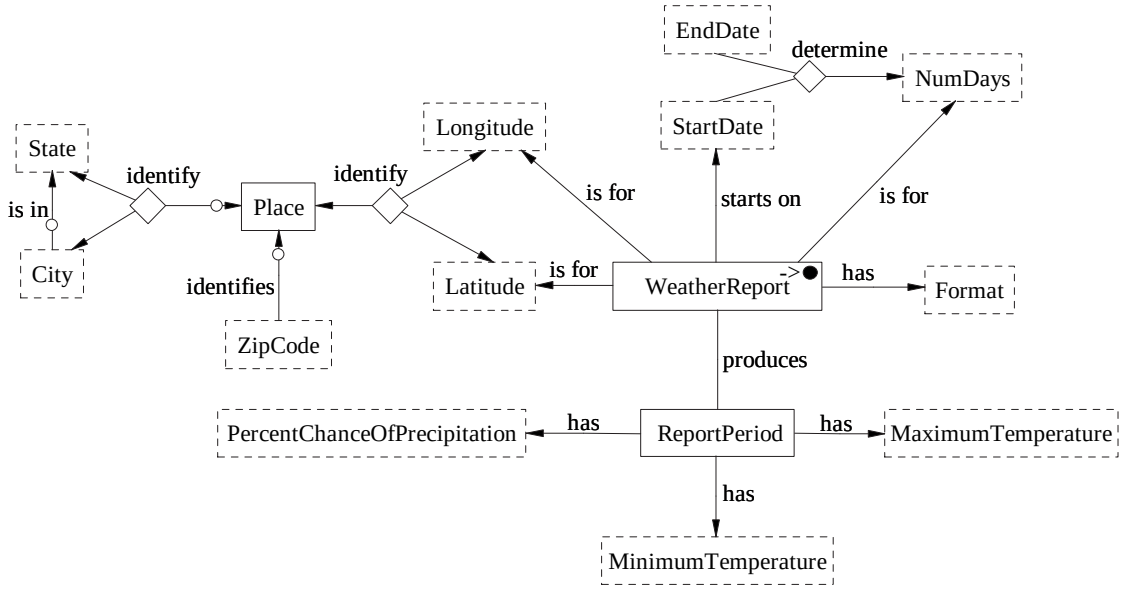


Figure 5.3: A semantic data model for a weather report service.

an example of a nonlexical object set because its instances are identifiers such as, say, "P_1", which represents a particular place in the real world.

We designate the main object set in a domain ontology by marking it with "–> •" in the upper right corner (e.g. *WeatherReport* in Figure 5.3). This notation,"–> •", denotes that when an ontology is used to satisfy a service request, the main object set becomes ("->") an object ("•"). The system satisfies a service request by instantiating the main object set with a single value such that all applicable constraints are satisfied.

Figure 5.3 also shows relationship sets among object sets, represented by connecting lines, such as *StartDate and EndDate determine NumDays*. The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *StartDate and EndDate determine NumDays* is functional from the pair *StartDate* and *EndDate* to *NumDays*, and *WeatherReport produces ReportPeriod* is many-many. A small circle near the connection between an object set O and a relationship set R represents an optional participation, so that an instance of O need not participate in a relationship in R. For example, the small circle on the *Place* side of the relationship set *ZipCode identifies Place* states that an instance of *Place* may or may not relate to an instance of *ZipCode*.

5.2.2 Data Frames

Each object set in a semantic data model has an associated data frame [Emb80], which describes instances for the object set. Data frames capture the information about object-set instances in terms of internal and external representations, context keywords or phrases that may indicate their presence, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the object set along with context keywords or phrases that indicate the applicability of an operation and operands in an operation. Figure 5.4 shows sample (partial) data frames for several object sets in Figure 5.3.

The internal representation specifies the data type for the instances of an object set. The *StartDate* instances, for example, are of type *date*, which must be of the form *yyyy-mm-dd*. We use regular expressions to capture external textual representations. The *StartDate* data frame, for example, captures date instances such as "July 6, 2007". The regular expression can also have lexicon references. For instance, "{stateName}" in the *State* data frame refers to a lexicon of state names. A data frame's context keywords/phrases are also regular expressions. For example, the *ZipCode* data frame in Figure 5.4 includes context keywords such as "zip code" or "zip". In the context of one of these keywords, if a 5-digit number appears, it is likely that this number is a zip code. A nonlexical object set such as *WeatherReport* has only context keywords or phrases. Figure 5.4 shows that the *WeatherReport* data frame includes keywords and phrases that could indicate the presence of an instance of a *WeatherReport*.

Operations in data frames manipulate object-set instances. For example, the operation *getLatitude*(x: *ZipCode*) computes the latitude for a given zip-code argument x, and the operation *toInternalRepresentation*(x: *StartDate*) transforms dates in various formats to the internal format. The context keywords/phrases for an operation indicate the possible applicability of the operation. The context keywords/phrases are regular expressions that include keywords or phrases and possibly expandable expressions represented by operand names enclosed in braces. The system expands these expressions by finding the types of their operands and substituting the textual representations in the data frames of the types

```
StartDate
  internal representation: date -- format: yyyy-mm-dd
  text representation:
    {monthName}\s+([0]?[1-9]|[12]\d|3[01])(\s*\,)?\s+\d{4}|
     (the\s+)?([0]?[1-9]|[12]\d|3[01])\s*(th|nd|rd|st)|...
  toInternalRepresentation(x: StartDate)
    returns (StartDate) -- format: yyyy-mm-dd
  Tomorrow()
    returns (StartDate) --next day with respect to today
    context keywords/phrases: tomorrow|next\s*day|...
  NrDaysBetween(x1: StartDate, x2: EndDate)
    returns (NumDays)
    context keywords/phrases: between\s+{x1}\s+and\s+{x2}|...
  getDefaultStartDate()
    returns (StartDate) -- today's date
  ...
State
  internal representation: string
  text representation:
    {stateName}|{statePostalCode}|{stateAbbreviations}
  ...
ZipCode
  ...
  text representation: [1-9]\d{4}
  context keywords/phrase: zip\s*code|zip
Latitude
  internal representation: real -- positive
  getLatitude(x1: State, x2: City)
    returns (Latitude)
  getLatitude(x: ZipCode)
    returns (Latitude) -- positive real number
  toInternalRepresentation(x: Latitude)
    returns (Latitude) -- positive real number
  ...
Longitude
  internal representation: real -- negative
  getLongitude(x1: State, x2: City)
    returns (Longitude)
  getLongitude(x: ZipCode)
    returns (Longitude) -- negative real number
  toInternalRepresentation(x: Longitude)
    returns (Longitude) -- negative real number
  ...
NumDays
  internal representation: integer
  getDefaultNumDays()
    returns (NumDays) -- 1
  ...
WeatherReport
  internal representation: object ID
  context keywords/phrases:
    (want\s+a\s+)?weather\s+report|weather|forecast|...
  ...

...
```

Figure 5.4: Some sample data frames. The ellipses "..." show omissions needed to complete the data frames. The "--" prefixes a comment that provides more of a clue about an aspect of the data frame specifications.

for these expressions. When context keywords/phrases for an operation match substrings in a service request, the system can record which values are for which operands. For instance, the context keywords/phrases associated with the operation *NrDaysBetween* in Figure 5.4 has the regular expression *between\s+{x1}\s+and\s+{x2}*, which includes the expandable expressions {*x1*} and {*x2*}. As Figure 5.4 shows, the operands of these two expressions are of type *StartDate*. When this regular expression matches a substring in a request such as "What is the weather going to be in Chicago between the 10th and the 15th," the system can record that the first date value ("the 10th") is for *x1* and the second date value ("the 15th") is for *x2*.

5.2.3 Servicing Requests with Ontology-Based Web Services

When an OBWS receives a service request, it applies the recognizers in the data frames of its underlying ontology to the request to extract information necessary for servicing the request. Consider the weather-report request in Figure 5.2 and an OBWS whose underlying ontology is in Figures 5.3 and 5.4. When applying the ontology in Figures 5.3 and 5.4 to the request in Figure 5.2, the OBWS extracts the strings "weather", "tomorrow", "Springfield", and "Illinois"—these strings appear highlighted in Figure 5.2.

The OBWS may, and often does, need to process the extracted information to make it comply with the required data as specified by the ontology. In our weather-request example, some of the extracted values do not comply with the data required by the OBWS. In this case, the request provides a state ("Illinois") and a city ("Springfield"), not a latitude and longitude as required. The OBWS uses the operation *getLatitude*("*Illinois*","*Springfield*") to compute a latitude and assigns the computed latitude value to the object set *Latitude* (similarly for a longitude). Likewise, the request provides "tomorrow" as a start date, which is not in the required format *yyyy-mm-dd*. Since the applicability recognizers of the operation *Tomorrow*() in the data frame recognize "tomorrow" in the request, and this operation returns a start date, the OBWS uses *Tomorrow*() to transform the recognized string "tomorrow" into a properly formatted start date and assigns the transformed value to the object set *StartDate*. Besides not being in the proper form, the information in the request also is incomplete since the request provides no values for *Format* and *NumDays*.

The OBWS, nevertheless, can use its ontology to provide default values for object sets.[1] For our example the OBWS provides the default value "24 Hourly" for *Format* and the default value"1" for *NumDays*. Thus, the OBWS can instantiate all the required data *Latitude*, *Longitude*, *StartDate*, *NumDays*, and *Format*, either by computing or transforming, if necessary, given values or by providing default values.

To service a request, an OBWS must instantiate its output data instances by retriving them from its database or by computing them from other available data instances. For our example, the OBWS retrieves from its database values for *MaximumTemperature*, *MinimumTemperature*, and *PercentChanceOfPrecipitation*. Using a process that is beyond the scope of this paper and is fully described elsewhere [AMEL05], the OBWS generates a database query, executes this query, and returns the query result values.

Observe that OBWSs resolve data heterogeneity. An OBWS extracts values from a request and assigns them to respective object sets of its ontology independently of how these values appear in the request. An OBWS uses the data frames operations to compute required values from given values and to transform recognized values to internal representations when necessary. An OBWS also uses default values defined by its ontology to add missing information to service requests.

5.3 Request-Oriented Architecture

Fensel [Fen04] proposes triple-space computing as a means to provide seamless interoperability between web services in a web-principled manner. In this section we introduce a *request-oriented architecture*, inspired by the feed syndication architecture of XML-based or RDF-based RSS and ATOM feeds [Ham03, Pow05, RSS07], that meets these additional interoperability requirements. Before we describe our architecture, we briefly explain how RSS/ATOM feeds work. In the RSS/ATOM-based feed subscription model, a user subscribes directly to feeds through feed reader clients, sometimes referred to as feed aggregators. Subscription to feeds (e.g. from news sites, blogs, podcasts, and other frequently updated web sites) consists of supplying a feed reader with a link to the feed. The feed

[1]When there are no default values for required information, the ontology lets the system know precisely which values are needed. The OBWS can request these values from the user.

reader then periodically checks for updates to the feed and notifies the user when a change has occurred in one of the subscribed feeds. Often, feeds only include summaries of the changes and the user must click on the feed's hyperlink to view the full change, which normally consists of new content or so-called articles relevant to some particular subject of interest to the user. The proposed request-oriented architecture emulates some of the concepts of the RSS-based feed subscription model, but more significantly it further extends those concepts to broker interactions between service requesters and OBWSs.

The request-oriented architecture provides a mechanism to *dynamically* match free-form service requests with OBWSs capable of servicing those requests. Fundamentally, this architecture is based on a centralized brokerage mechanism that enables OBWSs to subscribe to service-specific ontology feeds to which new service requests are posted. The brokerage mechanism's main functions are to match free-form service requests against available ontologies and to update service-specific ontology feeds with service requests so that OBWSs that subscribe to the feed can receive and service relevant service requests.

In the request-oriented architecture, service requesters make free-form services request such as the weather request in Figure 5.2. The broker matches a request with the available ontologies. For each ontology, the broker applies all the recognizers in the data frames to the request and ranks the ontology according to the number of matches the ontology has with the request. (More details about request-ontology matching and ranking can be found elsewhere [AME07].) The broker then selects the ontologies with the highest rank as a means to find OBWSs that can potentially service the request.

The service brokerage mechanism takes a two-tiered approach to broker requests and responses between service requesters and OBWSs. The first tier consists of mapping a service requester with potential OBWSs associated with the ontology that the broker determined to be most relevant to the service request. In this tier, the request is published as an update to the ontology feed. Then, the OBWSs that subscribe to the feed may choose to respond to the broker based on their availability and other individual criteria. The brokerage mechanism then presents the OBWS response proposals to the service requester as links. In the second tier of brokerage, the service requester selects a response proposal from the list provided by the brokerage mechanism, which indicates to the brokerage mechanism

the acceptance of the proposed service response. At this point, the brokerage mechanism performs the last step of the brokerage process by establishing a direct link between the service requester and the selected OBWS. This step consists of sending the complete service request to the selected OBWS along with the service requester's URL, the web session ID, and other relevant parameters in the POST request necessary for the OBWS to interact directly with the service requester. After the direct link between the service requester and the OBWS is successfully established, the OBWS then provides the complete response to the service requester and opens the door for further direct interactions.

Observe that the proposed OBWS architecture decouples service requesters and OBWSs. Requesters do not have to discover OBWSs that are capable of servicing their requests because the broker selects the capable OBWSs via request-ontology matching. In addition, requesters do not have to reference and establish communication links with the OBWSs. The broker references the OBWSs and passes the necessary information for them to dynamically establish communication links with requesters. OBWSs establish these communication links only when they need information from the requester to process requests or to return response.

The broker scales up to a reasonable size. The average size of the ontologies with which we have been experimenting is 18KB. Therefore, for 10,000 ontologies, the required space is about 180MB, which is relatively small for recent machines. Based on [SP01], for r regular expressions with average length of $\bar{n}$ characters, a text of length t characters, a machine of a clock speed s, the time to process the text using the regular expressions is $r\bar{n}t/s$. Since our ontologies have an average of 90 regular expressions with an average of 60 characters, processing a request of 70 characters, like the one in Figure 5.2, on a machine with a clock speed of 1.8GH would take about 0.20 milliseconds. Thus, for 10,000 ontologies the time is about 2 seconds. We add to this the time for applying lexicons, which each can be done in $(log_2 L)$ time, where L is the length of the lexicon. In any case, the time complexity is still manageable and can be greatly improved by using techniques such as parallel processing, duplicate regular-expression removal, and indexing.

5.4 Web-Principled Traditional Web Services

This section describes how to turn a traditional web service into an OBWS, not only so that it can be invoked via the OBWS architecture, but, perhaps more importantly, so that it can also exploit the decoupling and data heterogeneity resolution capabilities of the OBWS approach.

To create an OBWS from a traditional web service, we must provide an ontology that describes the service's external interface characteristics, and we must specify mappings between the ontology and the service's I/O requirements. A developer can reuse an ontology, if there is one, or create a new ontology that describes the interface layer semantics of the traditional web service. No change to the actual interface and the internal implementation of the service is required. The information required to create an ontological description for a traditional web service includes (1) the name of the service, (2) the names of the input and output parameters along with their types, and (3) the accepted formats for the values of the input parameters. Figures 5.3 and 5.4 show the ontology that describes the weather web service example in Figure 5.1.[2] The semantic model in Figure 5.3 encodes the input parameters of the service in terms of the object sets *Latitude*, *Longitude*, *StartDate*, *NumDays*, and *Format*. The semantic model also encodes the output parameters of the service in terms of the object sets *MaximumTemperature*, *MinimumTemperature*, and *PercentChanceOfPrecipitation*. We define the name of the operation as the main object set (marked with "–>•") in Figure 5.3.

The data frames in Figure 5.4 define the semantics of the possible instances of the object sets. The types of the object sets must comply with the types of the input and output parameters required by the service. As an example, the WSDL document that describes the interface for the weather service in Figure 5.1 defines the input parameter `startDate` to be of type `date` (as defined in XML Schema [W3C06]). Therefore, Figure 5.4 defines the type of the object set *StartDate*, the ontological description for `startDate`, to be of type `date`. The formats of the instances of the object sets must also comply with the accepted

[2] We obtained the information about the web service in Figure 5.1 from the WSDL document at `http://www.weather.gov/forecasts/xml/DWMLgen/wsdl/ndfdXML.wsdl`. We also invoked the service and reviewed its response as the WSDL document does not explicitly specify the names of the output parameters.

formats for the values of the input parameters. Therefore, Figure 5.4 defines the format for the instances of the object set *StartDate*, for example, to be *yyyy-mm-dd*, which is the valid format for values of `startDate` as specified by the weather service interface. In addition, an ontology designer must also decide which input parameters are required from the user and which input parameters can have default values. For instance, in our design for the ontology in Figures 5.3 and 5.4, *Latitude* and *Longitude* are required from the user whereas *StartDate*, for instance, is not required in the sense that, if not provided, it defaults to "today."

The ontology in Figures 5.3 and 5.4 also shows extensions to the input of the original service in Figure 5.1. It declares additional object sets *State*, *City*, and *ZipCode*. With these added object sets, a requester can invoke the service in Figure 5.1 not only using a latitude and a longitude, but also using a zip code, a city, or both a city and a state. It also declares *EndDate*, which provides an alternative way to specify the number of days for the forecast.

The ontological description of the traditional web service in Figure 5.1 enables requesters to invoke this web service using free-form requests. Figure 5.2 shows an example of invoking the service for the free-form weather report request: "What is the weather going to be like tomorrow in Springfield, Illinois?" The requester just specifies the request without referencing any particular service. The request does not need to satisfy any particular data type and format requirements specified by the service nor does it need to be complete. In this example, the request specifies a city and state, but not a latitude and a longitude as required by the service, and it does not provide any values for format and number of days.

To map the request to the input requirements of the service, the developer must specify which input parameter maps to which object set of the ontology. In our ontologies we do this mapping through the data frames by making each input parameter name a synonym for the respective object set in the ontology. In this way, the OBWS can pass the internal representation of an extracted value to the respective input parameter. Figure 5.5(a) shows the results of the mapping for our example. The developer must also map the service response to the ontology. Typically web service responses are XML documents embedded in SOAP messages. The developer therefore must specify which XML tag corresponding to which object set. Then, the developer can use XSLT [W3C07a] or XML parsers [XML06]

```
WeatherRequest(
                getLatitude("Illinois", "Springfield"),
                getLongitude("Illinois", "Springfield"),
                Tomorrow(),
                getDefaultNumDays(),
                getDefaultFormat()
               )
```

(a) An instantiated service request. The arguments are ordered as specified by the WSDL document.

```
<?xml version="1.0" ?>
  <dwml version="1.0" xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  ...
    <temperature ...>
      <name>Maximum Temperature</name>
      <value>70</value>
    </temperature>
    <temperature ...>
      <name>Minimum Temperature</name>
      <value>38</value>
    </temperature>
    <probability-of-precipitation ...>
        <name>Probability of Precipitation</name>
        <value>10</value>
    </probability-of-precipitation>
  ...
  </dwml>
```

(b) The service response to the request in Figure 5.5(a).

Figure 5.5: An instantiated request and the service's response to this request.

to extract values and assign these values to appropriate object sets. Figure 5.5(b) shows an example of the output from the service, which the developer must map to the ontology for our service.

5.5 Related Work

There are several existing projects related to our work. Some frameworks, such as the Semantic Web Service Framework (SWSF) [W3C05], the Web Ontology Language for Services (OWL-S) [MPM+05], and Web Service Modeling eXecution environment (WSMX) [HCM+05] can be used to develop web services from scratch by describing their internal representation with semantic data models (e.g. SWSF uses the Semantic Web Service Ontology and a Semantic Web Service Language for this purpose). Others, such as the

Semantic Annotation for the Web Services Description Language (SAWSDL) [W3C07b], provide a means to create an annotation of an existing web service's interface, while not being constrained to any ontology in particular.

The approaches most similar to ours are (1) architectures that provide a shared space through which web services (requesters and providers) can exchange data and (2) approaches that allow for both internal and external semantic modeling of the web service. The former includes the triple-space computing architecture [RMRD+06]. The latter includes the Web Service Modeling Framework (WSMF) [FB02], the Internet Reasoning Service (IRS) project [MDCG03][CDG+06], and WSMX [HCM+05].

The triple-space computing (TSC) architecture described in [RMRD+06] and further detailed in [Bus05] allows service requesters and providers to exchange data encoded as RDF triples [KC04]. In this architecture, the data exchanged between requesters and providers is written in the form of RDF triples to servers that host the so-called triple space. Web services can read RDF triples of interest by querying the servers that host these triples, process them, and write results as RDE triples to the triple space. Service requesters can also write their requests as RDF triples and read results after some services process their requests. TSC resolves syntactical heterogeneity between requesters and services by imposing a unified language (RDF) that both parties should use for communication. This is radically different from the OBWS approach in the sense that the OBWS architecture imposes no language because requesters can specify their requests in free-form specifications. The OBWS architecture has stronger heterogeneity resolution capabilities than the TCS architecture, which resolves heterogeneity only at the schema level [SSK+06]. Further, while the TSC architecture is passive, as services themselves need to check for new requests of interest, the OBWS architecture is active in the sense that it notifies OBWSs when a request of interest for an OBWS is posted.

The Web Service Modeling Framework, WSMF [FB02], provides decoupling among applications through an ontology modeling language and an ontology. This decoupling, however, is only partial because the mapping between ontologies and the service data needs to be done manually. In contrast, our approach uses data frames to automate the mapping

process. In addition, the WSMF is a framework that provides no web service implementation capabilities, while the OBWS approach does.

The IRS project (with its variations IRS-II [MDCG03] and IRS-III [CDG+06]) and the WSMX [HCM+05] approaches are all reference implementations for the Web Service Modeling Ontology (WSMO) [BBD+]. They enable service developers to describe their services using the WSMO ontology and register these descriptions in the system. Requesters can specify their requests, called goals, also using the WSMO ontology. The server matches goals with WSMO service descriptions and returns matches to users. Users then choose and invoke desired services. Both approaches differ from the OBWS approach in that they do not handle the kind of heterogeneity that our approach handles. They do, however, resolve mismatches between the request ontology and the service description ontology using pre-specified mapping rules, which must be created manually. Further, our approach allows users to make free-form service requests versus using the formal WSMO ontology to create goals.

WSMF, IRS, and WSMX are service-oriented approaches that partially accomplish *decoupling* and *heterogeneity* requirements through the conceptual modeling capabilities of ontologies and logic-based languages used to either describe the interface or define the inner workings of the web services. The OBWS approach described in this paper, on the other hand, is a request-oriented approach, which allows requests to be profiled with an extensional ontology and posted on an ontology feed, so that services subscribed to the feed can service the request. Neither the requester nor the service need to worry about each other's data representation, transport protocols, internal or external representation, or other idiosyncrasies in order to communicate.

5.6 Conclusions and Future Work

We have presented an ontological approach to enabling web-principled services via ontology-based web services (OBWSs). Web-principled services use the web as a place for information publication and access. They communicate asynchronously (and thus resolve coupling problems), and they exchange data without requiring requesters to comply with strict data

format specifications (and thus resolve heterogeneity problems). In addition, we have proposed an architecture for OBWS. Instead of a passive mechanism such as that embodied in UDDI-based brokerage services that require the service requester to find, adapt, and request, our proposed mechanism allows requests to be advertised in a manner that an OBWS can understand them and then propose to service those requests. Instead of a reactive mechanism that depends on human developers to make the necessary adjustments to the interacting requester and responder, the proposed mechanism provides proactive mapping between requests and responses through its relevant ontology-based feeds. As a spin-off of the basic OBWS framework, we have also discussed a way to turn a traditional web service into an OBWS. It suffices to describe a traditional web service with an ontology; it requires no changes to the interface and implementation of the service.

There are three important directions left for future work. First, we need to provide a mechanism for the broker to be able to choose and present the best service, or the best-k services, for a request when there are many relevant service providers that match the request. Authors in [YL04] and [CAH05] suggest criteria for selecting a service from among potential services according to non-functional aspects such as reliability, service cost, and availability. We can adapt these criteria to our approach or use other appropriate techniques. Second, the scalability of the proposed request-oriented architecture is also an important issue. Two important components of the architecture should scale. The broker should scale, theoretically, to an arbitrary number of ontologies and an arbitrary number of service requests. The RSS feeds mechanism should also scale to an arbitrary number of subscribers. One of the ways that we are planning to consider to improve this scalability is to take advantage of parallel processing techniques and indexing. Parallel processing allows the broker to do the matching on more than one processor, which is likely to decrease the processing time. Indexing allows us to apply cross regular expressions only one time instead of applying them for each ontology they belong in, which is also likely to decrease the processing time. Third, we want to extend our approach to handle composite services whose satisfaction requires an instantiation of multiple ontologies. For instance, a vacation planning web service should book an air ticket, reserve a hotel, and rent a car. Handling this type of a service, however, is not just a matter of independently instantiating ontologies.

There are clearly cross constraints that need to be satisfied in order for vacation planning requests to be correctly handled. For instance, the date of the car rental cannot be later than the return date of the air ticket.

By any reasonable measure, web principles have made a significant impact on the way people and machines interact. Ontology-based web services bring these same principles to the world of services. Improved decoupling and heterogeneity resolution will make web services easier to design, implement, and consume.

Acknowledgement

This work is supported in part by the National Science Foundation under Grants 0083127 and 0414644 and by Rollins Center for eBusiness at Brigham Young University under Grant EB-05046.

Bibliography

[AME06] M. J. Al-Muhammed and D. W. Embley. Resolving Underconstrained and Overconstrained Systems of Conjunctive Constraints for Service Requests. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE06)*, pages 223–238, Luxembourg, June 2006.

[AME07] M. J. Al-Muhammed and D. W. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 366–375, Istanbul, Turkey, April 2007.

[AMEL05] M. J. Al-Muhammed, D. W. Embley, and S. W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Conference on Conceptual Modeling (ER 2005)*, pages 288–303, Klagenfurt, Austria, October 2005.

[BBD+] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. Konig-Ries, J. Kopecky, R. Lara, E. Oren, A. Polleres, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). http://www.wsmo.org/TR/d16/.

[Bus05] C. Bussler. A Minimal Triple Space Computing Architecture. In *Proceedings of the 2nd WSMO Implemetation Workshop*, Innsbruck, Austria, June 2005.

[CAH05] D. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition. In *Proceedings of the 2nd International Workshop on Semantic and Dynamic Web Processes (SDWP 2005)*, pages 32–44, Orlando, Florida, July 2005.

[CDG+06] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, and B. Norton. IRS-III: A Broker for Semantic Web Services Based Appli-

cations. In *Proceedings of the 5th International Semantic Web Conference (ICWS 2006)*, pages 201–214, Athens, Georgia, November 2006.

[Emb80] D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[Fen04] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *Proceedings of IFIP International Conference on Intelligence in Communication Systems*, pages 43–53, Bangkok, Thailand, November 2004.

[Ham03] B. Hammersley. *Content Syndication with RSS*. O'Reilly Media, Sebastopol, California, 2003.

[HCM+05] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX - A Semantic Service-Oriented Architecture. In *Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, pages 321–328, Orlando, FL, July 2005.

[KC04] G. Klyne and J. Carroll. Resource Description Format (RDF): Concepts and Abstract Syntax. *http://www.w3c.org/TR/rdf-concepts*, 2004.

[KHP+05] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What Is Wrong with Web Services. In *Proceedings of the 3rd European Conference on Web Services (ECOWS 2005)*, pages 235–243, Växjö, Sweden, November 2005.

[MDCG03] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 306–318, Sanibel Island, FL, October 2003.

[MPM+05] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In J. Cardoso and A. Sheth, editors, *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*, volume 3387, pages 26–42, San Diego, California, July 2005.

[Pow05] S. Powers. *What Are Syndication Feeds*. O'Reilly Media, Sebastopol, California, 2005.

[RMRD+06] J. Riemer, F. Martin-Recuerda, Y. Ding, B. Sapkota, R. Krummenacher, O. Shafiq, D. Fensel, and E. Kühn. Triple Space Computing: Adding Sematics to Space-Based Computing. In *Proceedings of the 1st Asian Semantic Web Conference (ASWC 2006)*, pages 300–306, Beijing, China, September 2006.

[RSS07] RDF Rich Site Summery. Website, 2007. http://xml.coverpages.org/rss.html.

[SKB06] B. Sapkota, E. Kilgarriff, and C. Bussler. Role of Triple Space Computing in Semantic Web Services. In *Proceedings of the 8th Asia-Pacific Web Conference (APWeb 2006)*, pages 714–719, Harbin, China, January 2006.

[SP01] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238, Washington, DC, USA, 2001.

[SSK+06] O. Shafiq, F. Scharffe, R. Krummenacher, Y. Ding, and D. Fensel. Data Mediation Support for Triple Space Computing. In *Proceedings of the 2nd IEEE International Conference on Collaborative Computing (CollaborateCom 2006)*, Atlanta, Georgia, November 2006.

[W3C05] W3C. Semantic Web Services Framework. Website, 2005. http://www.w3.org/Submission/SWSF.

[W3C06] Extensible Markup Language (XML 1.0). Website, 2006. http://www.w3.org/TR/REC-xml/.

[W3C07a] Extensible Stylesheet Language Transoformation (XSLT 2.0). Website, 2007. http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[W3C07b] W3C. Semantic Annotations for WSDL and XML Schema. Website, 2007. http://www.w3.org/TR/sawsdl.

[XML06] Crimson: A Java XML 1.0 parser. Website, 2006. http://xml.apache.org/crimson/.

[YL04] T. Yu and K. Lin. Service Selection Algorithms for Web Services with End-to-End QoS Constraints. In *Proceedings of the IEEE International Conference on E-Commerce Technology (CEC 2004)*, pages 129–136, San Diego, California, July 2004.

Chapter 6

SerFR: Server for Free-form Requests
A Usability Study

In this chapter, we present an end-user-oriented evaluation of the usability of SerFR. SerFR (Server for Free-form Requests) is a proof-of-concept system that is built on the techniques presented in previous chapters—Chapters 2, 3, and 4. The goal is to see whether SerFR, which integrates these techniques, meets end users' satisfaction.

We evaluated the usability of SerFR in three domains: scheduling appointments, purchasing cars, and renting apartments. Subjects tried SerFR on these three domains and then answered a set of usability questions and provided suggestions for potential improvement. To determine the degree of its usability, we analyzed the responses of the subjects along with other information the system automatically collected and logged during the subjects' interaction with the system.

The outline of this chapter is as follows. Section 6.1 gives the details of SerFR usage. It focuses on the interactions that are important from the users' perspective and that are the goal of our usability evaluation. Section 6.2 discusses how we elicited the required information to determine the degree of usability of our system (Subsection 6.2.1) and provides a detailed analysis of the elicited information (Subsections 6.2.2 and 6.2.3). In Section 6.3 we give concluding remarks about the usability of the system. Appendix A contains the test instructions and Appendix B contains the questionnaire for the SerFR usability study.

Figure 6.1: User interface for specifying service requests with a car purchase service request.

6.1 SerFR—Usage Scenario

In this section, we discuss a use case of SerFR. The objective is to provide a clear idea about the functionalities of SerFR we want to test. We do not, however, discuss processes that are not interesting to users such as how the system selects a domain ontology that best matches a request or how the system generates a formalism for a free-form service request. These processes have been thoroughly explained in previous chapters.

SerFR provides users with an interface to specify their service requests. Figure 6.1 shows the interface, which consists of a statement about supported domains (scheduling appointments, purchasing cars, and renting apartments), a text area to specify free-form

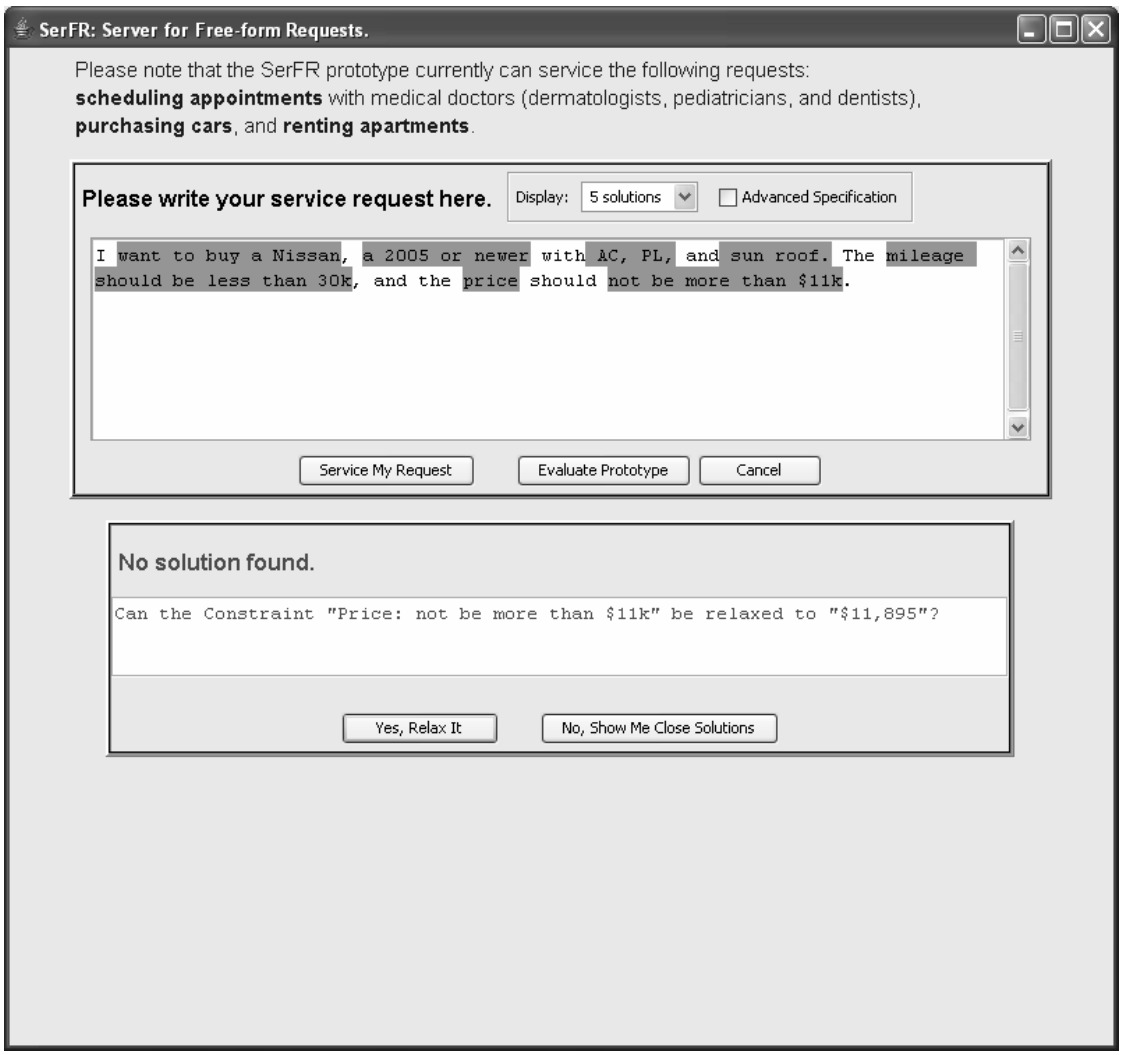


Figure 6.2: Highlighted recognized constraints (top panel) and a constraint relaxation message (bottom panel).

service requests, a choice box labeled with "Display" to select how many solutions to display, and a checkbox labeled with "Advanced Specification" to make advanced service request specifications with conjunctive, disjunctive, and negated constraints. There are also three buttons: "Service My Request" to service a specified service request, "Evaluate Prototype" to show the evaluation sheet that allows users to write their evaluation, and "Cancel" to close the interface. The empty bottom half of the interface is for giving feedback messages from the system, allowing users to respond to the system messages, and showing the results of processing service requests.

To make a request with only conjunctive constraints, a user types the request into the provided text area. Figure 6.1 shows an example of a car purchase request with constraints on make (Nissan), year (a 2005 or newer), features (AC, PL, and sun roof), mileage (less than 30k), and price (not be more than $11k). When a user clicks on the button "Service My Request", SerFR starts the constraint recognition process. The system highlights all recognized constraints in green as Figure 6.2 shows. The highlighting gives users an important clue about the constraints the system recognizes.

After constraint recognition, if the system is able to identify an ontology to use for servicing the service request, it starts the constraint satisfaction process. There are two cases the system handles: over-constrained service requests and under-constrained service requests. If these are insufficient, users may use the advanced-specification feature to specify their requests.

Over-constrained service requests. Our car purchase example admits no solution as indicated by the message "No solution found" in Figure 6.2. When SerFR finds no solutions, it prompts users to relax constraints to get near solutions. As Figure 6.2 shows, SerFR prompts users to relax the price constraint from "$11k" to "$11,895" as indicated by the statement: Can the constraint "Price: not be more than $11k" be relaxed to "$11,895"?[1] As Figure 6.2 shows, the system gives a user two possible choices represented by the two buttons. First, a user can click on the button "Yes, Relax It" to see the near solution that results from relaxing the price constraint to be "$11,895". Second, a user can click on the button "No, Show Me Close Solutions" to see near solutions that violate constraints.

For the sake of further discussing the capabilities of SerFR, we assume that a user clicks on the button "No, Show Me Close Solutions", which is likely to produce too many near solutions. SerFR controls the potential drudgery that results from having to evaluate all possible near solutions by displaying near solutions in two different tables. The first table shows the best-k near solutions, where k defaults to 5, but can be changed by users changing the default choice in the box labeled "Display" in Figure 6.2. As discussed in Chapter 4, SerFR produces the best-k near solutions by ordering the near solutions according to their

[1]The text between quotation marks is taken from the service request "not be more than $11k", from the ontology "Price", and from the database "$11,895" (the price of the car closest to satisfying the request). The other part: Can this constraint ... be relaxed to ...? is a general statement hardcoded in the system.

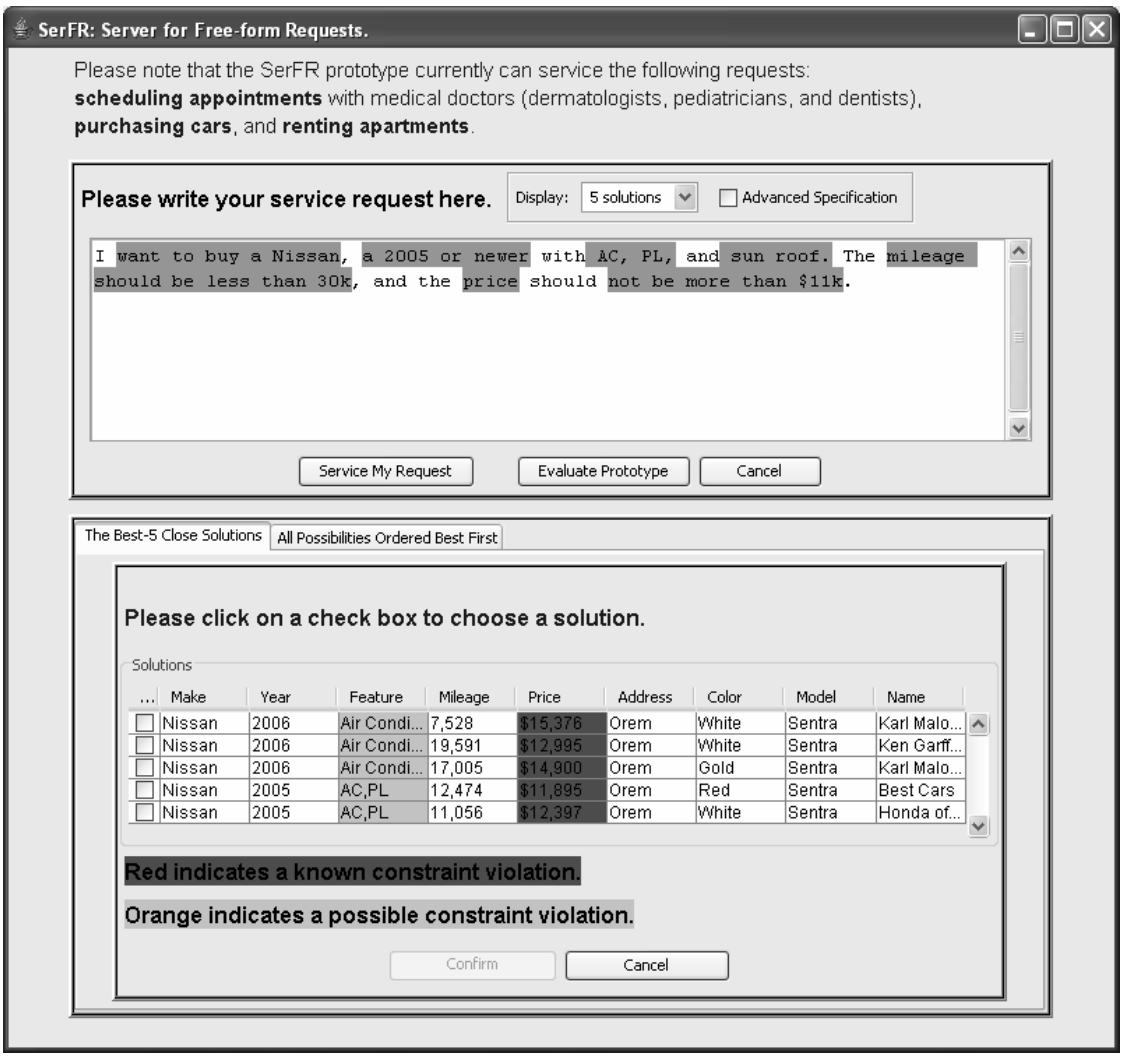


Figure 6.3: The best-5 near solutions.

increasing penalty (the near solution with least penalty first) and using Pareto optimality to select the top-k Pareto optimal near solutions from the penalty-ordering.[2] Figure 6.3 shows the best-5 near solutions under the tab "The Best-5 Close Solutions". A user can also switch to the second table that shows the penalty-ordering for all near solutions by choosing the tab "All Possibilities Ordered Best First" in the result display in Figure 6.3.

The system distinguishes between known constraint violations and possible constraint violations. As Figure 6.3 shows, the system marks all the price values under the column *Price* in red because they definitely violate the price constraint. As Figure 6.3 also

[2] As discussed in Chapter 4, a penalty is a negative real number that represents how much a near solution violates a constraint. Pareto optimality is a selection mechanism that eliminates all near solutions with a greater violation for each individual constraint than other near solutions.

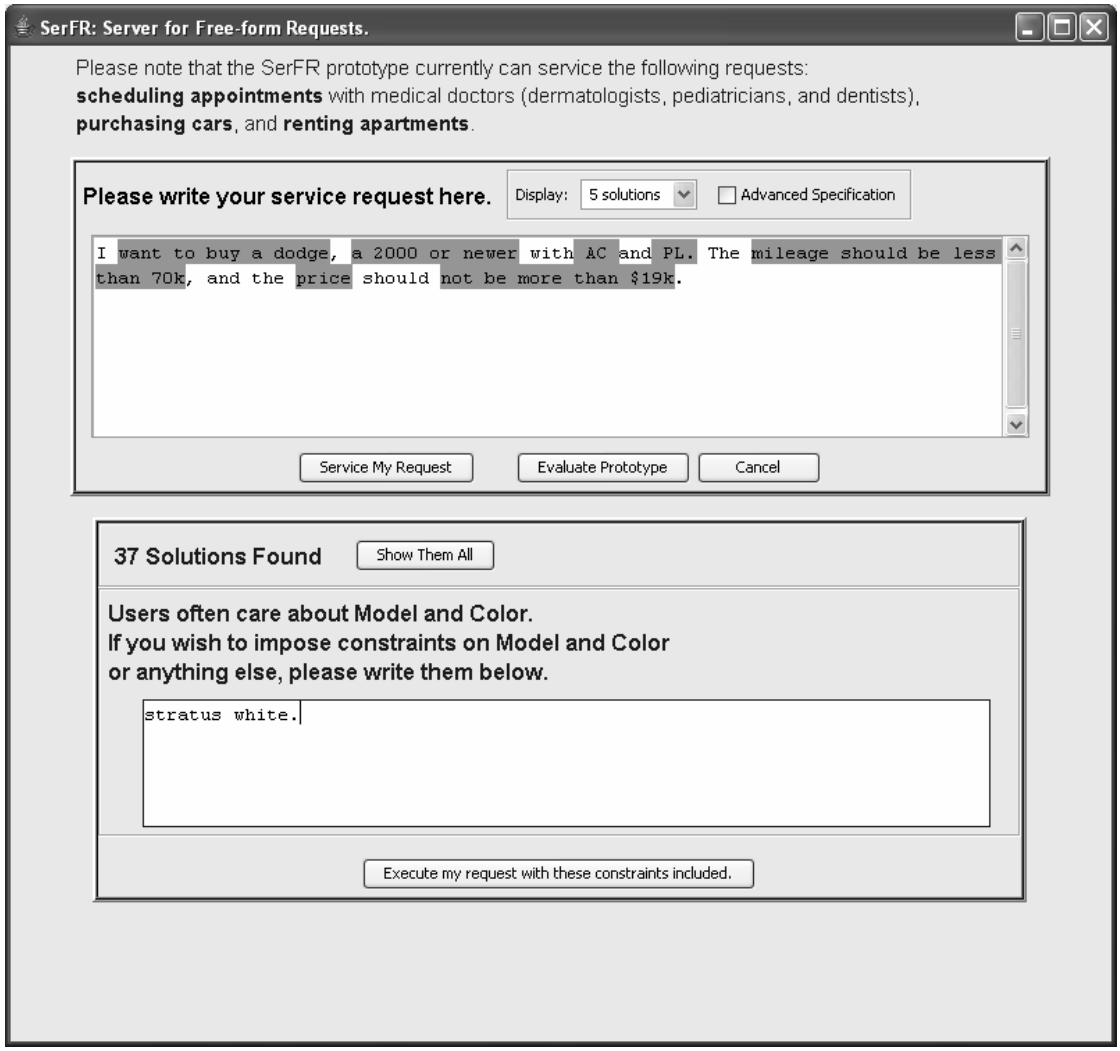


Figure 6.4: Constraint elicitation request.

shows, the system marks the feature values of the car under the column *Feature* in orange because the feature values do not mention "sun roof", which causes a possible but not a definite constraint violation.

As a convention, the system orders the column names in the display table based on the appearance of the constraints in the service request. The system orders alphabetically the remaining names that do not have corresponding constraints in a service request, if any. For instance, as Figure 6.3 shows, the system orders the column names *Make*, *Year*, *Feature*, *Mileage*, and *Price* because they appear in this order in the service request. The system orders alphabetically the remaining column names *Address*, *Color*, *Model*, and *Name* because they do not have constraints imposed on them in the service request.

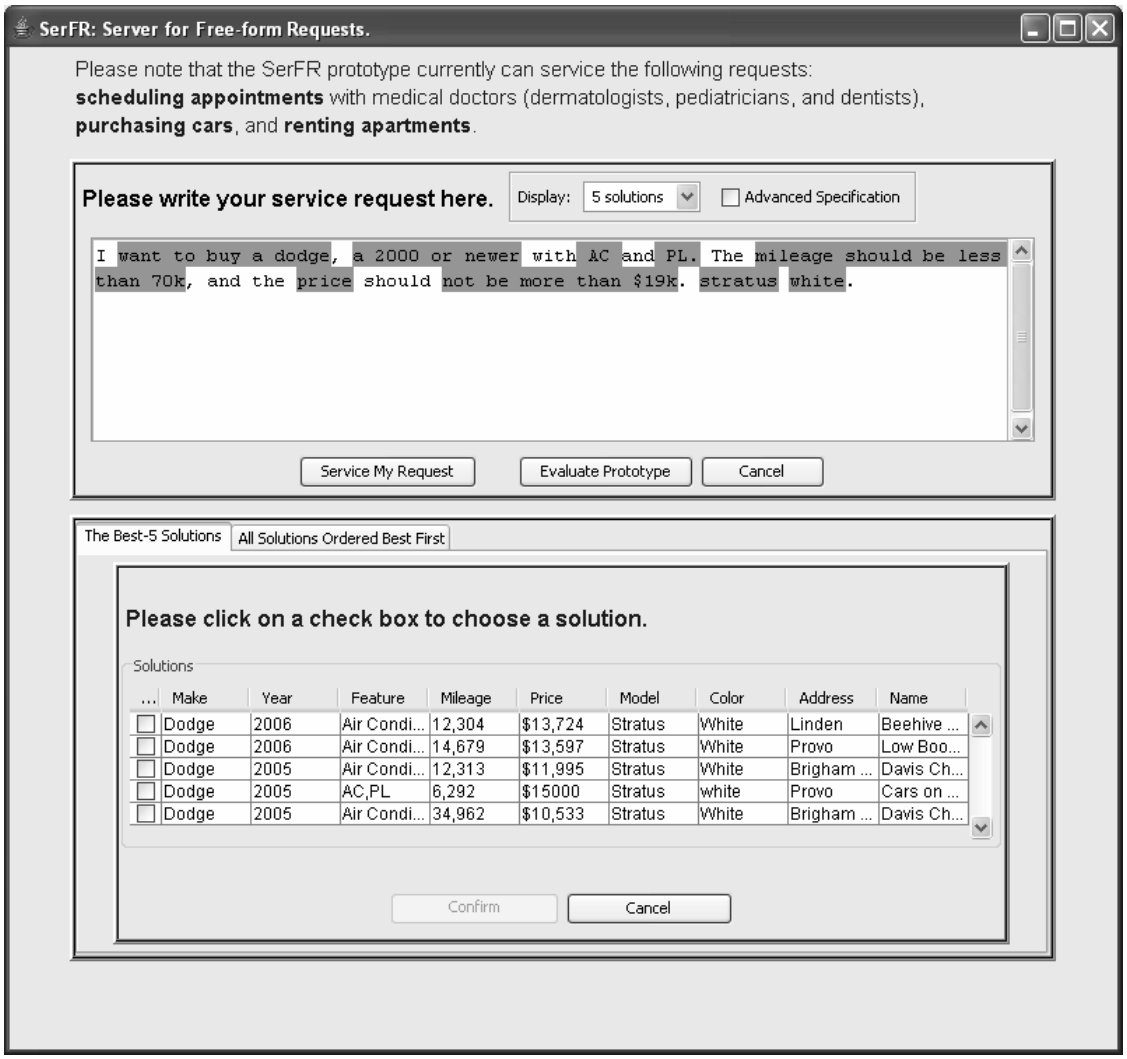


Figure 6.5: The best-5 solutions.

Under-constrained service requests. Figure 6.4 shows an example of a car purchase request that causes the system to find too many solutions. Figure 6.4 (bottom panel) shows that the system has found 37 solutions for this request. The system allows a user to see all the 37 solutions by clicking the button "Show Them All" in Figure 6.4.

However, as we discussed in Chapter 4, adding more constraints makes solutions better satisfy users' preferences and can further reduce the number of solutions. Therefore, SerFR suggests that users impose constraints on selected attributes based on their expectation or any other attributes of a user's choice.[3] As Figure 6.4 shows, the system

[3]An expectation of a concept in a domain ontology is the probability that a constraint on this concept appears in a service request. (See Chapter 4 for more details.)

suggests that the user imposes constraints on *Model* and *Color* or on anything else. Figure 6.4 shows that the user imposes a constraint on the model to be "stratus" and on the color to be "white" by writing them in the provided text area. When the user clicks on the button "Execute my request with these constraints included", the system adds these two constraints to those already specified and attempts to satisfy the constraints.

As in the case for near solutions, SerFR displays solutions in two tables. The first table shows the best-k solutions selected from the reward-ordering using Pareto optimality. The second table shows all the reward-ordering solutions with the highest rewards first.[4] Figure 6.5 shows the best-5 solutions that satisfy all the constraints. The user can also see the reward-ordering by choosing the tab "All Solutions Ordered Best First" in the result display in Figure 6.5.

Advanced specification. Users can also choose the advanced specification functionality of our system by clicking on the checkbox, "Advanced Specification", in the interface. Figure 6.6 shows the user interface for the advanced specification with a car purchase request example. As the figure shows, this functionality provides a text area, labeled "Tell me what you want", for users to specify the goal of the request (e.g. "I want to buy a car") and a group of three other text areas labeled "Tell me your constraints" to specify constraints on the goal. A user can specify (1) conjunctive constraints in the text area labeled "**All** these constraints", (2) disjunctive constraints in the text area labeled "**Any one or more** of these constraints", and (3) a conjunction of negated constraints in the text area labeled "**Not any** of these constraints". We add an illustrative statement to the last label (i.e. "write A B C to mean **not** A **and not** B **and not** C") to help users correctly specify them.

The system interprets the constraints based on the text area in which they are mentioned. The system creates a conjunctive formula by conjoining the recognized individual constraints mentioned in the text area labeled "**All** these constraints" plus any constraints mentioned in the text area labeled "**Tell me what you want**". The system creates a disjunctive formula over the recognized individual constraints in the text area "**Any one**

[4] A reward is a non-negative real number that represents how well a solution satisfies constraints. (See Chapter 4 for more details.)

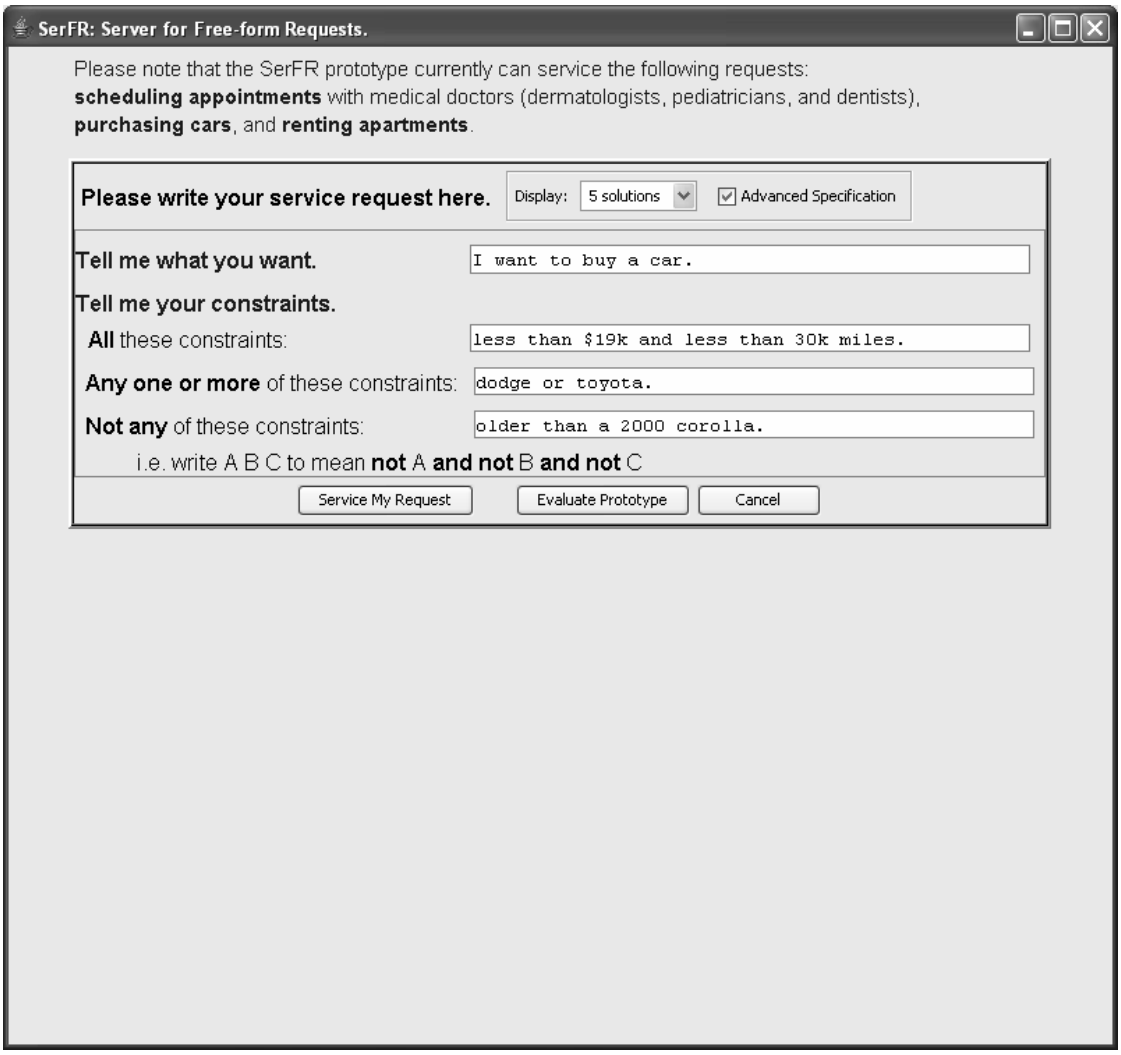


Figure 6.6: The advanced specification interface.

or more of these constraints." The system creates a conjunctive formula with every recognized constraint in the text area "**Not any** of these constraints" negated. For instance, in Figure 6.6, the system negates the year constraint ("older than a 2000") and the model constraint ("corolla") in the text area labeled "**Not any** of these constraints" and conjoins them to yield: *not* older than a 2000 *and not* corolla. The system generates the final formula by conjoining the generated sub-formulas. The system then satisfies the constraints and handles over-constrained and under-constrained service requests as before.

1. **Regular specification**:
   The regular specification is expressive enough to specify all the constraints I needed.
2. **Advanced specification**:
   While using SerFR, I heavily used the advanced specification because I would not otherwise have been able to express most of my requests.
3. **Constraint elicitation**:
   The system suggests the attributes that I would really like to impose constraints on.
4. **Constraint relaxation**:
   The system suggests the constraints that I really like to relax.
5. **Solution ordering**:
   The set of best-k solutions and the set of ordered solutions are helpful.
6. **Near solution ordering**:
   The set of best-k near solutions and the set of ordered near solutions are helpful.
7. **Useful system**:
   The system is helpful as it allows users to obtain and invoke services easily.

Figure 6.7: The tested functionalities (1 through 6) and the overall usefulness of SerFR (7).

6.2 SerFR Usability Study

We asked students at Brigham Young University to try SerFR. We were interested in their opinion about the overall usefulness of SerFR and about six particular functionalities: regular specification, advanced specification, constraint elicitation, constraint relaxation, solution ordering, and near solutions ordering. Figure 6.7 shows our particular functionalities of interest.

To have the subjects focus on the functionalities in Figure 6.7, we provided them with instructions. These instructions include the objectives of the test and when necessary tell them what to do to activate certain interesting cases such as advanced specifications. The complete set of instructions is in Appendix A.

Subjects specified free-form service requests using their own words within the domains: scheduling appointments, purchasing cars, and renting apartments. Altogether, 12 computer science students[5] at Brigham Young University tried the system. Each spent an average of 42 minutes and submitted an average of 13 requests. There were a total of 155 service requests specified in the three domains. After trying SerFR, subjects answered questions about the usability of SerFR and also provided suggestions for further improvement. The complete questionnaire is in Appendix B.

[5]Eight were in a senior-level database course class and four were master's students.

6.2.1 Information Elicitation

We obtained information from users in two ways: (1) by automatically recording information during the subjects' interaction with the system and (2) from answers to usability questions regarding the functionalities.

The system records all the actions of subjects and automatically logs information about performed actions. The system logs all keystrokes, service requests specifications (regular or advanced specification), newly imposed constraints during constraint elicitation, constraints that the system suggests for relaxation, and the chosen solution or near solution by the subject. The system timestamps the time a subject spends on specifying service requests, considering and choosing whether to impose new constraints during constraint elicitation, considering the suggested constraints for relaxation and choosing whether to relax, and considering the displayed solutions or near solutions. From this basic logged information, we can obtain several items of particular interest to our analysis:

- the total time a subject takes from starting the system until the subject starts answering the usability questions;
- the service requests subjects specify as well as the time they spend to specify each one of the service requests;
- the specified constraints SerFR recognizes;
- the attributes on which SerFR suggests imposing constraints and the actual attributes (perhaps not the ones that SerFR suggests) on which the subjects impose constraints as well as the time the subjects take to specify constraints;
- the constraints SerFR suggests for relaxation and whether subjects relax the suggested constraints as well as the time a subject takes to consider the suggested constraints and chooses to relax or not relax them;
- the time a subject takes to consider the displayed solutions until a subject selects a solution or cancels the selection process; and
- the time a subject takes to consider the displayed near solutions until a subject selects a near solution or cancels the selection process.

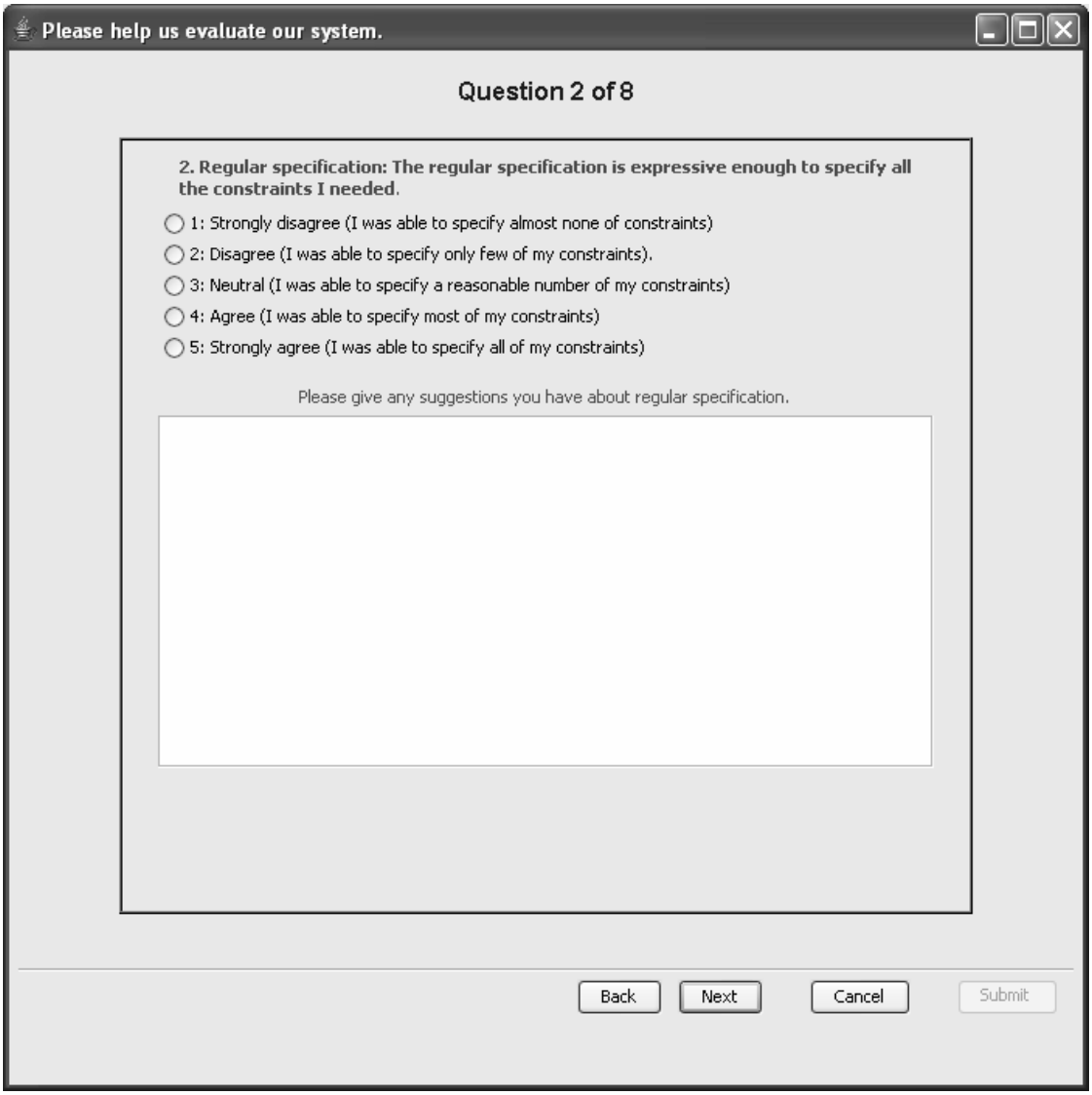
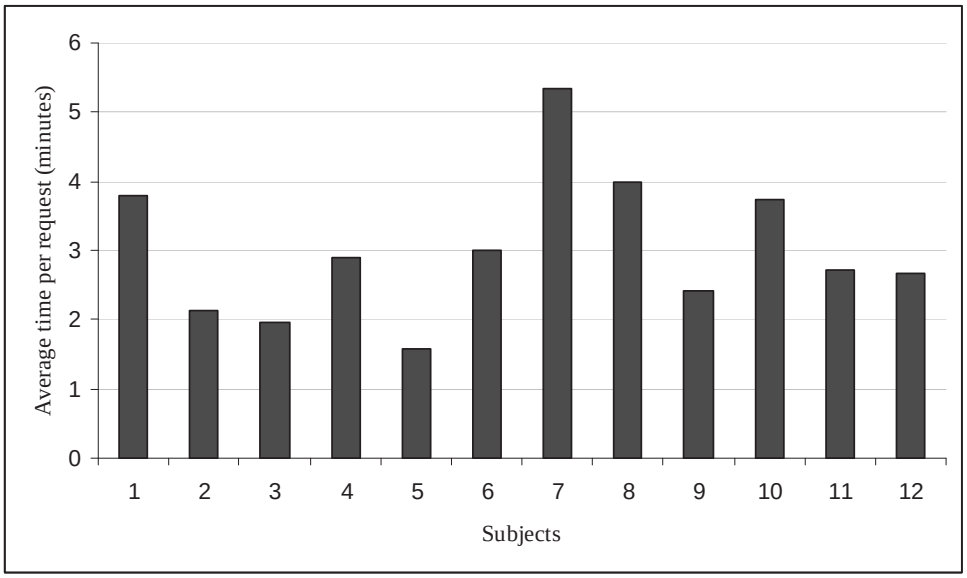


Figure 6.8: An example of a usability question.

Their answers and suggestions provide additional information for studying the usability of SerFR. Each functionality in Figure 6.7 has a corresponding usability question and a request for suggestions about the functionality. We made these usability questions and the requests for suggestions available as part of the system interface so that subjects can electronically submit their evaluations.

For each usability question about a particular functionality, we provided possible answer choices that reflect different degrees of satisfaction with the functionality. Figure 6.8 shows the usability question for the functionality "Regular specification" in Figure 6.7 and



(a) The average time taken by each subject per single request.

Overall average time	Standard deviation	95% Confidence interval
3.2 minutes	1.0 minutes	[2.4, 4.1]

(b) Related statistics.

Figure 6.9: The average time that each subject spent per single request along with the average time overall subjects.

the possible response values "1" ("Strongly disagree") through "5" ("Strongly agree"). It also shows the text area in which subjects can enter suggestions.

In what follows, we present the elicited data about the usability of SerFR and analyze it. In Subsection 6.2.2, we give logged times, responses to the questions and suggestions to show the subjects' opinions of the usability of SerFR, and both logged information and the subjects' responses to correlate opinions with factors such as the time to formulate and reuse queries. In Subsection 6.2.3, we provide a detailed analysis for the results we show in Subsection 6.2.2.

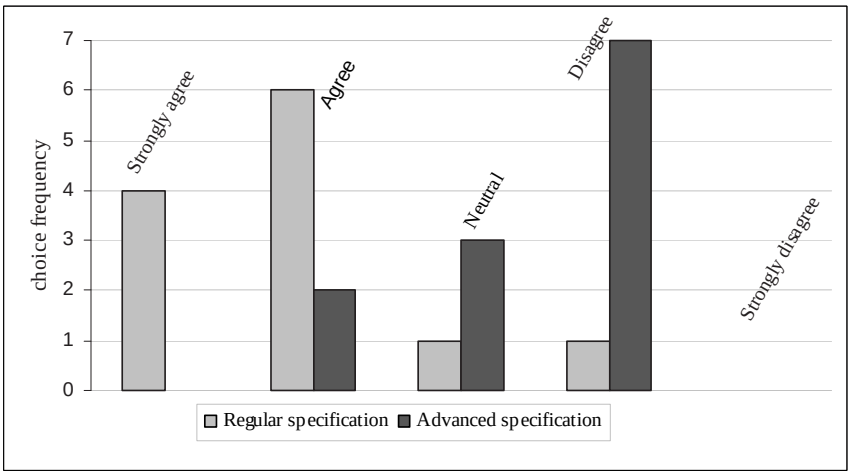
6.2.2 SerFR Usability Results

Figure 6.9 shows the average time from the moment a subject started specifying a service request until the subject chose a solution or a near solution returned by SerFR. Figure 6.9(a) visually shows the average time each subject spent per service request. As the figure shows the maximum average time was 5.3 minutes and the minimum average time was 1.6 minutes.

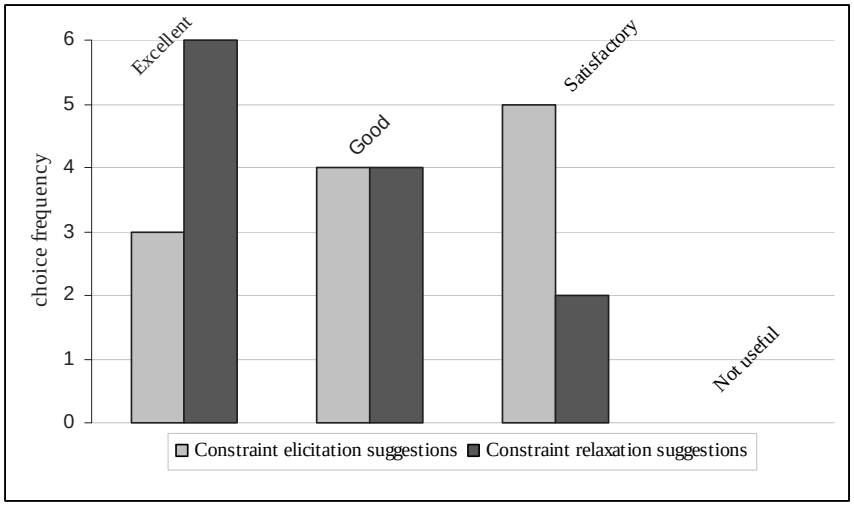
Figure 6.9(b) shows the average time over all subjects, standard deviation, and 95% confidence interval for the average. As Figure 6.9(b) shows, subjects took an average time of 3.2 minutes to perform one request. With 95% confidence, we can be sure that the time required to perform one request is not less than 2.4 minutes and not more than 4.1 minutes. Note that 5.3 minutes is an outlier and that it is possible to be outside the expected range.

Figure 6.10 (pages 127 and 128) shows the responses of the subjects for each functionality in Figure 6.7. Figure 6.10(a) shows the response of the subjects to the statement: "The regular specification is expressive enough to allow me to specify all the constraints I needed." As the figure shows, 33% (4/12) of the subjects strongly agreed with this statement in the sense that they were able to specify all their requests; 50% (6/12) agreed with this statement in the sense they were able to specify most of their requests; 8% (1/12) were neutral in the sense that they were able to specify a reasonable number of their constraints; 8% (1/12) disagreed with the statement in the sense that they were able to specify only few of their constraints; and no one strongly disagreed in the sense that they were able to specify almost none of their constraints. Figure 6.10(a) also shows the responses of the subjects to the complementary statement: "I heavily used the advanced specification mode because I would not otherwise have been able to specify all my requests." Only 17% (2/12) of the subjects agreed with this statement in the sense that they used it in most of their constraints; 58% (7/12) disagreed in the sense that they used it for very few of their requests; and 25% (3/12) were neutral in the sense that they used it for some of their requests. No one strongly agreed in the sense that they used advanced specification for all their requests and no one strongly disagreed in the sense that they never needed advanced specification for their requests.

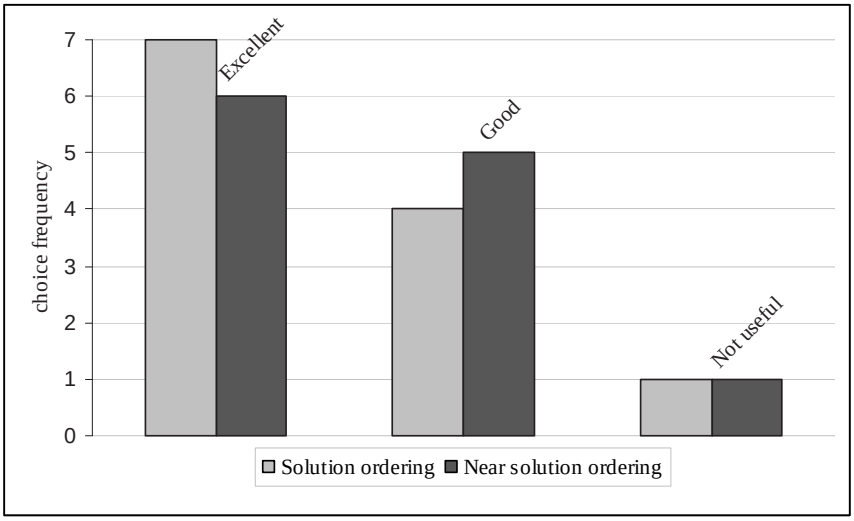
Figure 6.10(b) shows the responses of the subjects to statements about constraint relaxation and constraint elicitation. As the figure shows, 50% (6/12) of the subjects found the constraint relaxation suggestions to be excellent in the sense that the system suggested the constraints that the subjects really would want to relax; 33% (4/12) found them to be good in the sense that they would want to relax most of the suggested constraints; 17% (2/12) found them to be satisfactory in the sense that they would want to relax some the suggested constraints. No one found constraint relaxation suggestions to be not useful



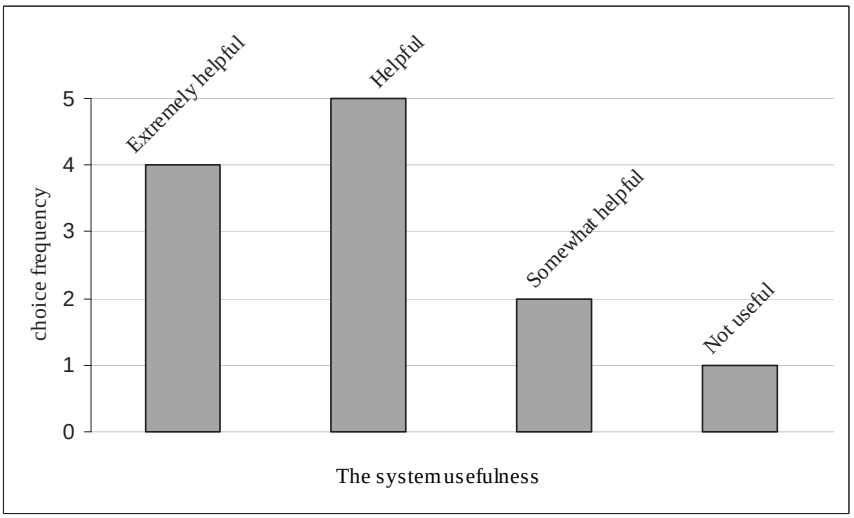
(a) Regular and Advanced specification.



(b) Constraint elicitation and relaxation.



(c) Solution and near solution ordering.



(d) The system usefulness.

Figure 6.10: The subjects selection frequency.

at all in the sense that the system did not provide useful suggestions. As Figure 6.10(b) shows, 25% (3/12) of the subjects found constraint elicitation suggestions to be excellent in the sense that all of the suggested attributes were the ones they would choose to specify constraints on; 33% (4/12) found them to be good in the sense that most of the suggested attributes were the ones they would choose to specify constraints on; while about 42% (5/12) found them to be satisfactory in the sense that some of the suggested attributes were the ones they would choose to specify constraints on. No subject found constraint elicitation suggestions to be not useful at all in the sense that no suggested attribute was satisfactory.

Figure 6.10(c) shows the responses of subjects about solution ordering and near solution ordering. As the figure shows, 60% (7/12) of the subjects found solution ordering to be excellent, and 50% (6/12) found near solution ordering to be excellent both in the sense that the set of best solutions or best near solutions were the best choices. As the figure also shows, 33% (4/12) of the subjects found solution ordering to be good, and 42% (5/12) found near solution ordering to be good both in the sense that the best solutions or best near solutions were useful choices. Only 8% (1/12) found both solution ordering and near solution ordering to be not useful at all in the sense that no solution ordering or near solution ordering was satisfactory.

Table 6.1: The satisfaction degree for the evaluated functionalities.

Functionality	Average response value (StDev; 95% CI)	Subject satisfaction degree*
1. Regular specification	4.1 (0.9; [3.5, 4.6])	Agree$^+$ [Neutral$^+$, Strongly agree$^-$]
2. Advanced specification	2.6 (0.8; [2.2, 2.9])	Neutral$^-$ [Disagree$^+$, Neutral$^-$]
3. Constraint elicitation	2.9 (0.8; [2.4, 3.3])	Good$^-$ [Satisfactory$^+$, Excellent$^-$]
4. Constraint relaxation	3.4 (0.8; [3.1, 3.8])	Good$^+$ [Good$^+$, Excellent$^-$]
5. Solution ordering	2.5 (0.7; [2.2, 2.8])	Good$^+$ [Good$^+$, Excellent$^-$]
6. Near solution ordering	2.4 (0.7; [2.1, 2.7])	Good$^+$ [Good$^+$, Excellent$^-$]
7. Useful system	3.0 (1.0; [2.4, 3.6])	Helpful [Somewhat helpful$^+$, Extremely helpful$^-$]

* A "+" means the satisfaction is higher than the specified degree; a "−" means the satisfaction is lower than the specified degree.

Figure 6.10(d) shows the subject general opinion regarding the overall usability of SerFR. As the figure shows, 33% (4/12) of the subjects found the system to be extremely helpful; 42% (5/12) found it to be helpful; 17% (2/12) found it to be somewhat helpful; and 8% (1/12) found it to be useless.

Table 6.1 shows the average response values across all subjects and the degree of satisfaction with respect to the judgement scale. The second column in Table 6.1 shows the average response across all the subjects, the standard deviation, and the 95% confidence interval for each one of the functionalities.[6] The third column "Subject satisfaction degree" in Table 6.1 shows the satisfaction degree corresponding to the average response value and to the lowest and highest response values as indicated by the 95% confidence interval. We mark the satisfaction degree with "+" if we round off to a lower response value and mark it with "−" if we round off to a higher response value. For instance, in Table 6.1, the first functionality received an average response value of 4.1; since the response value 4 corresponds to "Agree", we give this functionality the satisfaction degree "Agree" and marked it with plus "+".

[6]The average response is the sum of all the values corresponding to the subject-selected responses divided by the number of the subjects. We attached a numeric value to each response based on how positive the response was with respect to the functionality—the more positive the response, the higher the numeric value.

We were interested in investigating any correlations between subjects' response about functionalities and four factors: (1) how often subjects use online services to shop, (2) the number of requests a subject specifies, (3) the number of constraints in these requests, and (4) the average time each subject spends to complete a request. We calculated correlation values between the subjects' response values for each functionality and each of the four factors; we also calculated correlation values between our overall functionality score (sum of all response values of the choices that the subject made for all seven functionalities) and each of the four factors. The objective was to see whether subject responses significantly correlate with the four factors. The results, however, showed that there was no statistically significant correlation between subject responses, both across the functionalities and for each functionality, with any of the four factors.

6.2.3 SerFR Usability Results Discussion

Generally speaking, the results of SerFR usability study presented in Section 6.2.2 reveal that SerFR is usable. As Table 6.1 shows, subjects found the two modes of the specification to be enough to specify their constraints. Usually, regular specification was sufficient for most requests (Agree$^+$) and advanced specification was only used for some or very few requests (Neutral$^-$). As Figure 6.10(a) indicates, when a subject agrees with the statement about the advanced specification, the subject sees that the regular specification is inadequate for specifying all requests the subject wants to specify, while disagreeing means that a subject finds the regular specification sufficient. Table 6.1 also shows that subjects found the constraint handling (relaxation and elicitation) to be effective (respectively Good$^+$ and Good$^-$). Figure 6.10(b) indicates, however, that constraint-relaxation suggestions were better perceived than constraint-elicitation suggestions. Table 6.1 also shows that solution ordering and near solution ordering are helpful (Good$^+$). Figure 6.10(c) indicates that ordering is typically helpful independent of whether it is for solutions or for near solutions. Overall, Table 6.1 shows users found SerFR to be usable (Helpful).

The statistically insignificant correlation between subject satisfaction and how often users use online shopping services indicates that using SerFR was straightforward (and perhaps easy) regardless of whether subjects are savvy online shoppers or not. Likewise, the

statistically insignificant correlation between subject satisfaction and the time, the number of service requests, and the number of constraints in these service requests indicates that the performance of SerFR was satisfying independently of how these factors vary. For instance, SerFR performed well in recognizing constraints regardless of how many constraints a subject specified.

Besides the data, we also collected comments from users. Here are some quotes taken from the log files about the overall usability of SerFR, which provide an indication that subjects were satisfied with SerFR. One subject wrote, "I think the system is practical and excellent in quality." Another subject wrote about SerFR's ability to recognize constraints, "... the system does a great job of working through the English."

Subjects also pointed out some problems and made suggestions for further improving SerFR and enhancing its usability. In what follows, we present and discuss these problems and suggestions, which we divide into four categories.

1. **Free-form specifications versus form-like specifications**. One of the subjects argued for form-like specification: "... people want answers quick. Drop down, selection boxes, and textboxes with labels would more quickly direct people to the results they need than this [free-form specification]." SerFR does not oppose using forms per se. Actually, SerFR interacts with users through forms, albeit more general forms than the specific forms to which our subject was referring. SerFR, however, aims at eliminating the burdens of using typical service-specific forms. In order to use a form for a particular service, a user needs to find the form for this service. SerFR frees users from the burden of finding service-specific forms by providing a general form—not tied with any specific service—that allows users to request any service using free-form specification. Further, assuming that a form for a particular service is located, we argue that although users may be familiar with service-specific forms, service-specific forms can be limiting. A users' ability to specify constraints in a form-like way is largely limited by the pre-specified and prefixed fields and labels provided by the form. Users can specify no additional constraints beyond the ones pre-specified by the form. SerFR specification, on the other hand, let subjects

specify all the constraints they needed, as indicated by subject responses. Besides the subject opinion, we can also, from an analytical stand point, provide indications for sufficiency of SerFR specification. SerFR free-form specifications, both regular and advanced, allow users to specify all conjunctive constraints, disjunctive constraints, and conjunctions of negated constraints.[7] Form specification, however, usually only allows for conjunctive constraints.

2. **User preferences**. Some subjects suggested that SerFR should allow users to provide their preferences because this would allow the suggestions for constraint relaxation or elicitation to be more personalized. SerFR provides a general expectation model to capture user preferences. The expectation model is built on the basis of common practices by users in a domain. This expectation model, although it may not reflect every individual user preference, provides reasonable help over all users and especially for users who do not have knowledge of a domain and therefore do not have prior preferences. In any case, providing users with the capability to specify their preferences, if they wish, is a potentially good extension to SerFR.

3. **Help instead of just suggest**. Some subjects indicated that it would be more helpful if SerFR would not just suggest attributes on which to impose constraints, but would also show what possible constraints apply to each of these suggested attributes: "... it would be extremely helpful to give some sort of explanation as to what constraints can be specified." The subject likely meant that SerFR should associate with each suggested attribute a set of constraints that apply to this attribute so that the subject could see how to apply the constraint appropriately. We think that doing so is a potentially good extension to SerFR and could enable users to specify a constraint for an attribute in typical ways. When the number of constraints applicable to a suggested attribute is reasonable, SerFR could provide this set of constraints. Otherwise, in order to free users from the burdens of having to sort through a potentially lengthy

[7] We have recently extended SerFR with the capability to enable users to specify constraints in conjunctive and disjunctive normal forms. With this extension, SerFR would theoretically have the full power of first-order logic. As a direction for future work, we should conduct an end-user usability study to investigate whether this extension would, in real-world settings, provide any added-value with respect to a users's ability to specify requests and thus further increase the usability of SerFR.

list of constraints, SerFR could order these constraints based on some criterion such as their usage frequency and present the top-k constraints.

4. **Processing time issues**. Two subjects pointed out that SerFR is slow. One subject wrote: "... my number one complaint is that it was slow. Took on average 1 1/2 to 2 minutes to process any single request! My computer is not the top of the line, but it's not SO bad either!" Based on the average times to process a request reported in Figure 6.9(a), the subject likely referred to the time for processing a single request to completion. We admit up front that SerFR is just a proof-of-concept prototype whose main objective is to show that our techniques (Chapters 2, 3, and 4) can produce a useful system. We, therefore, did not make it a top priority to optimize the internal processing time. For instance, ontology indexing techniques, which we did not integrate to SerFR, could allow SerFR to apply identical regular expressions belonging to different ontologies only once rather than to apply them as many times as they appear. Beside this missing time optimization, there are also two other factors that are likely to affect the performance of SerFR in terms of the time required to process a service request. First, the communication with users, to prompt them to provide missing information, to provide new constraints, or to relax constraints, is very likely to increase the processing time for a service request. SerFR does not have complete control over this factor since it is mainly related to individuals. Solutions to this problem rely to a large extent on users: provide service requests with complete information or respond faster in case of requests that are incomplete. Second, SerFR executes on the client side not on the server side. As such, several factors can decrease the time performance of SerFR. The speed of the client machine on which SerFR executes plays a role. The network latency, which results because SerFR queries a database that resides on the server, is likely to increase the processing time. We can eliminate both machine speed variations and network latency to a great extent by making SerFR execute on the server rather than on the client. Finally, the time to execute a query on the database can take longer, depending on the query. Query

optimization techniques (e.g. [SMWM06]), however, can improve the execution time of a query.

6.3 Concluding Remarks

Although preliminary, the end-user-usability study shows that SerFR is usable. Subject evaluations provide reasonable evidence for usability. There were 12 subjects. Each tried SerFR for a reasonably long time (42 minutes on average) and specified a reasonable number of service requests (13 on average). After this effort, 9 out of 12 subjects found SerFR extremely helpful (4/9) or helpful (5/9). Two subjects found it somewhat helpful and only one subject found SerFR not useful, indicating a strong preference for form-like specification.

We believe that SerFR is usable not only because others generally agree, as indicated in the results of this chapter, but also because of its unique capabilities. SerFR allows users to obtain services in an easy and a friendly way. Specifying a needed service using free-form specifications is enough to invoke it. SerFR frees end-users from the burdens typically involved in service invocation (discovery, referencing, and other idiosyncrasies of services). It proactively interacts with end-users when their service requests are incomplete in order to obtain additional information, when their service requests are loosely constrained in order to impose new constraints, or when their service requests are tightly constrained in order to relax constraints. And it finds the best solutions or near solutions based on user-specified constraints, freeing users from having to sort through long lists of solutions or near solutions to find a satisfactory solution or near solution.

Appendix A

SerFR Usability Study Test Instructions

Thank you for being willing to help with the evaluation of the SerFR prototype. SerFR is a server for Free-form Requests.

Running the prototype

1. Go to: `http://www.deg.byu.edu/demos/SerFR.demo/SerFRMain.html`. This link takes you to a page describing the prototype. To run the prototype, click on the link "Launch Demo" in this page.
2. The first window that pops up gives a description of the applications the prototype can currently handle. You need to log in as a user by clicking on the button "User? Please Click Here". In the new window, you need to provide your name and address, store the profile, and click on the button "Start Prototype".
3. The main interface will appear and allow you to specify your service requests.

Service request specification

The prototype currently handles service requests in three domains: *appointment scheduling* (*only with dermatologists*, *pediatricians*, and *dentists*), *car purchasing*, and *apartment renting*. You can specify service requests within these domains using a free-form, natural-language-like specification. For instance, here is an example of requesting an appointment with a dermatologist: "I want an appointment with a dermatologist next week in the afternoon. The dermatologist must accept my IHC insurance." After you click on the button "Service My Request", the system starts the recognition process (all recognized constraints are highlighted in green), interacts with you if necessary, and displays results.

Interactions

The system interacts with a user in the following cases.

1. It interacts when needed information is missing, SerFR specifically requests it.
2. It interacts when there are too many ways to service your request (too many solutions), serFR tries to provide you with a smaller set of best solutions in two ways. First, it suggests some additional constraints you could impose. Second, based on an optimality criterion, it orders solutions from best to worst.
3. It interacts when there is no way to solve your request, SerFR tries to find near solutions. First, it suggests some constraints to relax. Second, based on an optimality criterion, it orders the best-k near solutions for best to worst.

Test objectives

Please try the prototype with the following objectives in mind.

1. Determine whether requesting service with free-form specification is helpful.
2. Determine the extent to which free-form specification allows you to specify your constraints.
3. Determine the extent to which advanced specification allows you to specify your constraints.

   To use advanced specification you need to (i) switch to the advanced specification mode by checking the box labeled "Advanced Specification" in the interface and (ii) make a request or more that has disjunctive (e.g. "Dodge or Toyota") or negated constraints (e.g. "not corolla") in the respective provided text areas.
4. Determine whether the system gives you satisfactory attributes on which you can impose constraints.
5. Determine whether giving the ordering of solutions is helpful or not.

   To trigger (4) and consequently (5), make the constraints of your request loose. For instance, an appointment request such as "schedule me an appointment with a dentist after October 15th" will most likely cause the system to find too many solutions.
6. Determine whether the constraints suggested by the system for relaxation are satisfactory.

7. Determine whether giving the best-k near solutions is helpful.

   To trigger (6) and consequently (7), make the constraints of your request tight. For instance, a request for a car purchase such as "I want a dodge, a 2005 or newer and less than $6000" will most likely cause the system to find no solution.

Appendix B

SerFR Usability Study Questionnaire

Subject expertise

Please tell us how often you shop online.

(a) On a daily basis.

(b) Several times a week.

(c) Several times a month.

(d) Occasionally.

(e) Never (I have never shopped online).

Service request specification

1. *Regular specification*: The regular specification is expressive enough to specify all the constraints I needed.

   (a) Strongly agree (I was able to specify all of my constraints).

   (b) Agree (I was able to specify most of my constraints).

   (c) Neutral (I was able to specify a reasonable number of my constraints).

   (d) Disagree (I was able to specify only few of my constraints).

   (e) Strongly disagree (I was able to specify almost none of my constraints)

   Please provide any suggestions you have about regular specification.

2. *Advanced specification*: While using SerFR, I heavily used the advanced specification because I would not otherwise have been able to express most of requests I wanted to specify.

   (a) Strongly agree (I used it for all my requests).

   (b) Agree (I used it for most of my requests).

(c) Neutral (I used it for some of my requests).

(d) Disagree (I used it for very few of my requests).

(e) Strongly disagree (I did not need to use it at all).

Please give any suggestion about advanced specification.

Constraint resolution

3. *Constraint relaxation*: The constraints that the system suggested for relaxation are:

   (a) Excellent (the system suggested the constraints that I would really would want to relax).

   (b) Good (I would want to relax most of the suggested constraints).

   (c) Satisfactory (I would want to relax some of the suggested constraints).

   (d) Not useful at all (the system did not provide any useful suggestions).

   Please provide us with the following.

   (a) Some examples of useless suggested constraints for relaxation.

   (b) Why were these suggested constraints for relaxation useless.

      i. You do not wish to relax any constraints.

      ii. The trade off seemed unacceptable.

      iii. Others, please specify.

4. *Constraint elicitation*: The attributes that the system suggested to impose constraints on are:

   (a) Excellent (all of the suggested attributes are the ones that I would choose to impose constraints on).

   (b) Good (most of the suggested attributes are the ones that I would choose to impose constraints on).

   (c) Satisfactory (some of the suggested attributes are the ones that I would choose to impose constraints on).

(d) Not useful at all.

Please provide us with examples of:

i. Useless suggested attributes.

ii. Why were they useless.

iii. Please specify examples of attributes that are more important to you than the ones suggested by the system.

Solution and near solution presentation

5. *Solution ordering*: The set of best solutions provided by the system are:

   (a) Excellent (they are best choices with respect to my constraints).

   (b) Good (they are useful choices, but could have been better).

   (c) Not useful at all (none of them was satisfactory).

   Please give us suggestions to improve our system.

6. *Near solution ordering*: The set of best near solutions (some of the constraints are not satisfied) provided by the system are:

   (a) Excellent (they are best choices with respect to my constraints).

   (b) Good (they are useful choices, but could have been better).

   (c) Not useful at all (none of them was satisfactory).

   Please give us suggestions to improve our system.

System efficiency

7. *System usefulness*: Based on your experience with the system, do you think that the system is helpful? (helpful in terms of simplifying service invocation for users who have limited knowledge in dealing with computers).

   (a) Extremely helpful.

   (b) Helpful.

(c) Somewhat helpful.

(d) Useless.

Please provide us with any suggestions you have about the system in general.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this dissertation, we offered an ontological approach to allow users to invoke a specific type of service request using free-form, natural-langauge-like specifications. Specifically, our approach handles service requests that can be satisfied by instantiating an object set of interest in the domain ontology with a single value and then instantiating additional objects and relationships in the domain ontology such that all applicable constraints are satisfied.

Task ontologies are fundamental to our approach and provide both static knowledge and behavioral knowledge. Service developers manually create the static knowledge, which we call a domain ontology, and which encodes domain information in terms of object sets, relationship sets, operations that manipulate instances of object sets, and constraints appropriate to the domain. The SerFR system developer codes behavioral knowledge, which we call a process ontology, and which consists of generic process to handle service requests. Interestingly, the behavioral knowledge is fixed for all domains so that the SerFR system developer only needs to code it once. This reduces the task of the service developers to only needing to provide the static knowledge in terms of a domain ontology.

Our ontological approach matches a free-form service request with domain ontologies and uses the domain ontology that matches best to generate a formal request as a predicate-calculus formula. The system processes the generated formalism to service the free-form request. Using the formalism, our system can discover missing information in the service request and interact with a user, the system databases, or both to obtain this information, and then satisfy the service request.

Our system controls the potential overload resulting from having too many solutions for a service request when the constraints are too weak, and it helps users obtain the best near solutions when the constraints are too tight. The system uses fundamental ideas including rewards, penalties, and expectations along with Pareto optimality to support our resolution process. The system uses expectations as a mechanism for constraint elicitation when a service request is weakly constrained. The system also orders solutions according to their rewards and uses Pareto optimality as a selection mechanism to choose the best-k solutions from the reward-ordering and show them to a user. When the constraints of a service request are too tight, yielding no solution, the system uses the expectations to suggest constraints for relaxation. Our approach also orders near solutions according to their penalties and uses Pareto optimality to choose the best-k near solutions from the penalty-ordering and show them to a user.

When a user chooses one of the suggested solutions or near solutions, the system completes the service request. The system inserts an object (e.g. an appointment) in the main object set of the domain ontology and inserts other mandatory and optional objects and relationships and thus satisfies the service request.

Our ontology-based techniques also enable web-principled services. (Web-principled services use the web as a place for information publication and access and not merely as a transport mechanism.) Researchers have suggested the notion of web-principled services as a means to achieve more decoupling between communicating applications. Our ontology-based techniques take this notion a step farther. We presented our vision of web-principled services as services that not only can use the web as a place for information publication and access, but also have the capability to resolve data heterogeneity. We showed that our ontology-based services, which we called ontology-based web services (OBWSs), satisfy our vision as they resolve data heterogeneity and decouple requesters and services. Our ontology-based web services inherently resolve data heterogeneity due to their ontological basis.

To satisfy the additional requirement for requester-service decoupling, we proposed a request-oriented architecture that emulates and extends the concepts of RSS feeds. This architecture allows service requesters to specify their requests to the broker using free-form

specifications. The broker then finds the OBWSs capable of processing these requests and lets requesters choose the appropriate OBWS based the requester's own criteria. One of the interesting characteristics of this architecture is that service requesters do not have to discover the OBWSs capable of servicing their requests nor do they have to establish communication links and directly communicate with these OBWSs. The broker finds the OBWSs capable of servicing the request, and the selected OBWS itself establishes a communication link with a requester.

As part of our vision of web-principled services, we showed that traditional web services can be turned into web-principled services. To do this, it is sufficient to reuse or create an ontology that describes the traditional web service and provide mappings between the ontology and the input/output variables of the traditional web service. We showed, as a concrete example, how to turn a weather report service into a web-principled service.

Experiments on our prototype implementation show that our ontological approach is promising. We tested the performance of the system in recognizing constraints in free-form service requests in several domains: car purchase, appointment scheduling, and apartment rental. The system performed surprisingly well. The system achieved a recall of 98% for recognizing constraints and a recall of near 95% for recognizing arguments for these constraints in the requests. The system achieved a precision of near 100% for both constraints and their arguments. We attributed this high recall and precision to our ontological approach, which uses domain ontologies that are narrowly focussed on the services they define and that provide enough information to do reasoning to discard irrelevant information.

We also evaluated our system's performance in choosing the best-k solutions or the best-k near solutions for requests in our car-purchase domain and our appointment-scheduling domain. We let subjects choose the best-k solutions or near solutions from the same sets of solutions or near solutions and compared the system's choices with our subjects' choices. As measured statistically by an inter-observer agreement test, the results of the experiments showed a "substantial agreement" between the system's choices and our subjects' choices in both the best-k solutions and the best-k near solutions.

The end-user usability study shows that our system is helpful in the sense that it provides an easy way to invoke services without having to discover them or to worry about

their idiosyncrasies. Subjects found free-form specifications to be sufficient. Regular specification was sufficient in most cases, and advanced specification was used only for some or very few requests. The subjects found constraint-relaxation and constraint-elicitation suggestions to be effective, although they understood constraint-relaxation suggestions better than constraint-elicitation suggestions. Likewise, the subjects generally found solution ordering and near solution ordering to be equally helpful. Overall, our subjects found the system to be helpful.

As a result of this research, we make the following conclusions.

1. **Easy service invocation**. Our ontological approach provides a way to invoke services using free-form, natural-language-like specifications. This is an alternative to the typical way of finding services and figuring out how to use them.
2. **Knowledge-based formalism generation**. Our ontological approach provides an way to transform a free-form service request to a formal request represented as a predicate-calculus formula. This formalism is machine-processable and allows the system to service the free-form request through satisfying the constraints in the formalism.
3. **Solution selection**. Our ontological approach controls the potential overload on users when a service invocation results in too many solutions and helps users find satisfactory solutions when there is no way to satisfy all the constraints.
4. **Knowledge-based service creation**. Our ontological approach allows service developers to deploy services for new domains by specifying only static knowledge (domain ontologies); surprisingly no behavioral knowledge (algorithms and code) is required. This creates an important implication for service developers. They do not need to write and test code for their services.
5. **Web-principled services**. Our ontological approach contributes to a new vision of web services. This new vision aims at basing web services more fundamentally on web principles. The novelty of our approach is that it produces ontology-based web services that not only comply with this new vision, but also more importantly

that advance this vision a step farther by providing a data-heterogeneity resolution mechanism.

6. **Web-principled traditional web services**. Our ontological approach can turn traditional web services into web-principled services. This allows traditional web services to be discovered and invoked like any web-principled service. This also negates the requirements for prior agreement about data exchanged between the service and its potential requesters.

7.2 Future Work

This dissertation has laid out the foundations for ontology-based service requests and has provided a way for users to invoke services using free-form specifications. It has also provided a way for service providers to deploy ontology-based web services. We believe, however, that there still remains important extensions that can be done as future work.

1. **Conditional constraints**. Conditional constraints can give an added power to users to specify more types of constraints. For instance, a user can schedule the most convenient appointment based on some conditions as in the following partial appointment request: "... if the appointment can be scheduled this week, then schedule with Dr. Adams; otherwise schedule with Dr. Carter." One challenge in correctly recognizing conditional constraints resides in recognizing conditions and consequents and in correctly correlating the conditions with their consequents. We expect that we would have to augment our approach with techniques from the discipline of natural-language processing (especially logic-form transformation) to effectively handle this type of constraint.

2. **Composite service requests**. Experiments in this dissertation only focused on service requests that can be satisfied by instantiating an object set of interest with a single value. We believe, however, that our approach can naturally extend to handle service requests that require using multiple ontologies in a coordinated way. Consider, for instance, a request for planning a vacation, which involves among others booking an airline ticket, renting a car, and making a hotel reservation. To satisfy a request of

this type, the system needs to use three domain ontologies corresponding to these subtasks. To correctly handle the vacation planning request, however, cross constraints among the applicable ontologies need to be satisfied. For instance, we cannot make the date of the car rental after the return date of the airline ticket and other similar kinds of constraints. Further, package deals across multiple vacation services may alter cost values.

3. **Security**. In our approach, users issue service requests and the system returns responses to these requests. This exchange of messages may contain sensitive pieces of information and therefore the communicated messages must be secure so that unauthorized entities cannot access their contents. In our approach, it appears possible to adopt any of the state-of-the-art security techniques such as encryption, signed certificates, or any other standard techniques developed, for instance, by the Web Security Technical Committee [WS06].

4. **Trust**. Trust is an important factor for the success of any service system. Generally, as user trust increases, so does system usage. According to Constantine [Con06], enhancing the system's predictability, transparency, competence, and benevolence highly promotes the user's trust in a system. Although, these concepts are not absent from our current prototype, we have not taken them as fundamental concepts and therefore we need to deal with them as such in our future work.

Bibliography

[ABdB+06] D. Anicic, M. Brodie, J. d. Bruijn, D. Fensel, T. Haselwanter, M. Hepp, S. Heymans, J. Hoffmann, M. Kerrigan, J. Kopecky, R. Krummenacher, H. Lausen, A. Mocan, J. Scicluna, I. Toma, and M. Zaremba. A Semantically Enabled Service Oriented Architecture. In *Proceedings of WICI International Workshop on Web Intelligence (WI) meets Brain Informatics (BI) (WImBI 2006)*, pages 27–42, Beijing, China, December 2006.

[ABH+02] A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web Service Description for the Semantic Web. In I. Horrocks and J. Hendler, editors, *Proceedings of the 1st International Semantic Web Conference (ISWC 2002)*, volume 2342, pages 348–363, Sardinia, Italy, June 2002.

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. Website, May 2003. http://www-128.ibm.com/developerworks/library/specification/ws-bpel.

[AHS03] S. Agarwal, S. Handschuh, and S. Staab. Surfing the Service Web. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 211–226, Sanibel Island, Florida, October 2003.

[AME06] M. J. Al-Muhammed and D. W. Embley. Resolving Underconstrained and Overconstrained Systems of Conjunctive Constraints for Service Requests. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*, pages 223–238, Luxembourg, June 2006.

[AME07] M. J. Al-Muhammed and D. W. Embley. Ontology-Based Constraint Recognition for Free-Form Service Requests. In *Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007)*, pages 366–375, Istanbul, Turkey, April 2007.

[AMEL05] M. J. Al-Muhammed, D. W. Embley, and S. W. Liddle. Conceptual Model Based Semantic Web Services. In *Proceedings of the 24th International Confer-*

ence on Conceptual Modeling (ER 2005), pages 288–303, Klagenfurt, Austria, October 2005.

[AMELT07] M. J. Al-Muhammed, D. W. Embley, S. W. Liddle, and Y. A. Tijerino. Bringing Web Principles to Services: Ontology-Based Web Services. In *Proceedings of the 4th International Workshop on Semantic Web for Services and Processes (SWSP 2007)*, pages 73–80, Salt Lake City, Utah, July 2007.

[AP04] S. Anthony and J. Patrick. Dependency Based Logical Form Transformation. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 54–57, Barcelona, Spain, July 2004.

[Arn02] M. T. Arnal. *Scalable Intelligent Electronic Catalogs*. PhD Dissertation, Swiss Federal Institute of Technology in Lausanne (EPFL), 2002.

[ART95] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural Language Interfaces to Database: An Introduction. *Journal of Natural Language Engineering*, 1(1):29–81, March 1995.

[BBCP05] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE – A Java Agent Development Framework. In *Multi-Agent Programming*, volume 15, pages 125–147. 2005.

[BBD+] J. De Bruijn, C. Bussler, J. Domingue, D. Fensel, M. Hepp, M. Kifer, B. Konig-Ries, J. Kopecky, R. Lara, E. Oren, A. Polleres, J. Scicluna, and M. Stollberg. Web Service Modeling Ontology (WSMO). http://www.wsmo.org/TR/d16/.

[BBGW04] S. Bayer, J. Burger, W. Greiff, and B. Wellner. The MITRE Logical Form Generation System. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 69–72, Barcelona, Spain, July 2004.

[BKF05] A. Bernstein, E. Kaufmann, and N. E. Fuchs. Talking to the Semantic Web – A Controlled English Query Interface for Ontologies. *AIS SIGSEMIS Bulletin*, 2(1):42–47, January-March 2005.

[BLHL01] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.

[BN03] F. Baader and W. Nutt. Basic Description Logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 2, pages 43–95. Cambridge University Press, Cambridge, UK, 2003.

[Bus05] C. Bussler. A Minimal Triple Space Computing Architecture. In *Proceedings of the 2nd WSMO Implemetation Workshop*, Innsbruck, Austria, June 2005.

[CAH05] D. Claro, P. Albers, and J. Hao. Selecting Web Services for Optimal Composition. In *Proceedings of the 2nd International Workshop on Semantic and Dynamic Web Processes (SDWP 2005)*, pages 32–44, Orlando, Florida, July 2005.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL), W3C Note 15. Website, 2001. http://www.w3.org/TR/wsdl/.

[CDG+06] L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, V. Tanasescu, C. Pedrinaci, and B. Norton. IRS-III: A Broker for Semantic Web Services Based Applications. In *Proceedings of the 5th International Semantic Web Conference (ICWS 2006)*, pages 201–214, Athens, Georgia, November 2006.

[CDM+04] L. Cabral, J. Domingue, E. Motta, T. R. Payne, and F. Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. In *Proceedings of the 1st European Semantic Web Symposium (ESWS 2004)*, pages 225–239, Heraklion, Crete, Greece, May 2004.

[CF90] D. Cicchetti and A. Feinstein. High Agreement But Low Kappa. II. Resolving The Paradoxes. *Journal of Clinical Epidemiology*, 43(6):551–558, 1990.

[CGGS07] M. Cossentino, S. Gaglio, A. Garro, and V. Seidita. Method Fragments for Agent Design Methodologies: From Standardisation to Research. *International Journal of Agent-Oriented Software Engineering*, 1(1):91–121, 2007.

[CMRW07] R. Chinnici, J. Moreau, C. A. Ryman, and S. Weerawarana. Web Services Description Language (WSDL), version 2.0. Website, 2007. http://www.w3.org/TR/wsdl20/.

[Con06] L. Constantine. Trusted Interaction: User Control and System Responsibilities in Interaction Design for Information Systems. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE 2006)*, pages 20–30, Luxembourg, June 2006.

[CPC+04] H. Chen, F. Perich, D. Chakraborty, T. Finin, and A. Josh. Intelligent Agents Meet Semantic Web in a Smart Meeting Room. In *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 2, pages 854–861, New York, July 2004.

[dSdL07] V. T. da Silva and C. J. de Lucena. Modeling Multi-Agent Systems. *Communications of the ACM*, 50(5):103–108, May 2007.

[DW03] I. Dickinson and M. Wooldridge. Practical Reasoning Agents for the Semantic Web. In *Proceedings of the 2nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2003)*, pages 306–318, Melbourne, Australia, July 2003.

[ECJ+99] D. W. Embley, D. M. Campbell, Y. S. Jiang, S. W. Liddle, D. W. Lonsdale, Y. K. Ng, and R. D. Smith. Conceptual-Model-Based Data Extraction from Multiple-Record Web Pages. *Data & Knowledge Engineering*, 31(3):227–251, November 1999.

[EKW92] D. W. Embley, B. K. Kurtiz, and S. N. Woodfield. *Object-Oriented Systems Analysis: A Model Driven Approach*. Yourdon Press, Englewood Cliffs, New Jersey, 1992.

[Emb80] D. W. Embley. Programming with Data Frames for Everyday Items. In D. Medley and E. Marie, editors, *Proceedings of AFIPS Conference*, pages 301–305, Anheim, California, May 1980.

[Erl05] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall, New Jersey, 2005.

[FB02] D. Fensel and C. Bussler. The Web Service Modeling Framework (WSMF). *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

[Fel80] A. M. Feldman. *Welfare Economics and Social Choice Theory*. Kluwer, Boston, Massachusetts, 1980.

[Fen04] D. Fensel. Triple-Space Computing: Semantic Web Services Based on Persistent Publication of Information. In *Proceedings of IFIP International Conference on Intelligence in Communication Systems*, pages 43–53, Bangkok, Thailand, November 2004.

[FPTV04] B. Faltings, P. Pu, M. Torrens, and P. Viappiani. Designing Example-Critiquing Interaction. In *Proceedings of the 9th International Conference on Intelligent User Interface*, pages 22–29, Funchal, Portugal, November 2004.

[GBR05] B. Gold-Bernstein and W. Ruh. *Enterprise Integration*. Addison Wesley, Boston, Massachusetts, 2005.

[Hal04] T. Halpin. Business Rule Verbalization. In *Proceedings of the 3rd International Conference on Information Systems Technology and its Applications*, pages 39–52, Salt Lake City, Utah, July 2004.

[Ham03] B. Hammersley. *Content Syndication with RSS*. O'Reilly Media, Sebastopol, California, 2003.

[HCM+05] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. In *Proceedings of IEEE International Conference on Web Services (ICWS 2005)*, pages 321–328, Orlando, Florida, July 2005.

[Hen01] J. Hendler. Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30–37, January 2001.

[KA04] M. Klein and B. Abraham. Towards High-Precision Service Retrieval. *IEEE Internet Computing*, 8(1):30–36, January 2004.

[KC04] G. Klyne and J. Carroll. Resource Description Format (RDF): Concepts and Abstract Syntax, 2004. http://www.w3c.org/TR/rdf-concepts.

[KCGS96] L. Karttunen, J. P. Chanod, G. Grefenstette, and A. Schille. Regular Expressions for Language Engineering. *Natural Langauge Engineering*, 2(4):305–328, December 1996.

[KG03] J. Kim and Y. Gil. Toward Interactive Composition of Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida, 2003.

[KHP+05] R. Krummenacher, M. Hepp, A. Polleres, C. Bussler, and D. Fensel. WWW or What Is Wrong with Web Services. In *Proceedings of the 3rd European Conference on Web Services (ECOWS 2005)*, pages 235–243, Växjö, Sweden, November 2005.

[KKS+02] S. Kumar, A. Kunjithapatham, M. Sheshagiri, T. Finin, A. Joshi, Y. Peng, and R. S. Cost. A Personal Agent Application for the Semantic Web. In *Preceedings of AAAI 2002 Fall Symposium Series*, pages 43–58, North Falmouth, Massachusetts, November 2002.

[KSB04] D. Krafzig, D. Slama, and K. Banke. *Enterprise SOA: Service Oriented Architecture Best Practices*. Prentice Hall, New Jersy, 2004.

[Kun04] A. Kunjithapatham. Personal Agents on Semantic Web. Master thesis, University of Maryland Baltimore County, Baltimore, Maryland, January 2004.

[LEW00] S. W. Liddle, D. W. Embley, and S. N. Woodfield. An Active, Object-Oriented, Model-Equivalent Programming Language. In M. P. Papazoglou, S. Spaccapietra, and Z. Tari, editors, *Advances in Object-Oriented Data Modeling*, pages 333–361. MIT Press, Cambridge, Massachusetts, 2000.

[LHL97] G. Linden, S. Hanks, and N. Lesh. Interactive Assesment of User Preference Models: The Automated Travel Assistant. In *Proceedings of the 6th International Conference on User Modeling (UM 1997)*, pages 67–78, Vienna, New York, June 1997.

[LK77] J. R. Landis and G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, March 1977.

[LYJ06] Y. Li, H. Yang, and H. V. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT 2006)*, pages 737–754, Munich, Germany, March 2006.

[MA02] K. Mark and B. Abraham. Searching for Services on the Semantic Web using Process Ontologies. In I. Cruz, S. Decker, J. Euzenat, and D. McGuinness, editors, *The Emerging Semantic Web-Selected papers from the first Semantic Web Working Symposium*, pages 159–172. Amsterdam, Netherlands, August 2002.

[MA04] R. T. Marler and J. S. Arora. Survey of Multi-Objective Optimization Methods for Engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, March 2004.

[MC99] F. Meng and W. W. Chu. Database Query Formation from Natural Language using Semantic Modeling and Statistical Keyword Meaning Disambiguation. Technical Report CSD-TR 990003, University of California, Los Angeles, California, 1999.

[MDCG03] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the 2nd International Semantic Web Conference (ISWC 2003)*, pages 306–318, Sanibel Island, Florida, October 2003.

[min05] Minitab 14.2 Statitiscal Software. Website, 2005. www.minitab.com.

[MMP04] A. Mohammed, D. Moldovan, and P. Parker. Sensevale 3 Logic Form: A System and Possible Improvements. In *Proceedings of the 3rd International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, pages 163–166, Barcelona, Spain, July 2004.

[MPM+05] D. Martin, M. Paolucci, S. McIlraith, M. Burstein, D. McDermott, D. McGuinness, B. Parsia, T. Payne, M. Sabou, M. Solanki, N. Srinivasan, and K. Sycara. Bringing Semantics to Web Services: The OWL-S Approach. In

J. Cardoso and A. Sheth, editors, *Proceedings of the 1st International Workshop on Semantic Web Services and Web Process Composition*, volume 3387, pages 26–42, San Diego, California, July 2005.

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

[MTI95] R. Mizoguchi, Y. Tijerino, and M. Ikeda. Task Analysis Interview Based on Task Ontology. *Expert Systems with Applications*, 9(1):15–25, 1995.

[OWL04] OWL-S Coalition, OWL-S 1.0 Release. Website, 2004. http://www.daml.org/services/owl-s/1.0/.

[PAE04] A. M. Popescu, A. Armanasu, and O. Etzioni. Modern Natural Language Interfaces to Databases: Composing Statistical Parsing with Semantic Tractability. In *Proceedings of the 20th International Confereence on Computational Linguistics*, pages 30–39, University of Geneva, Switzerland, August 2004.

[Par97] V. Pareto. *Cours d'économie politique*. F. Rouge, Lausanne, Switzerland, 1897.

[PEK03] A. M. Popescu, O. Etzioni, and H. Kautz. Toward a Theory of Natural Languages Interfaces to Databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, pages 149–157, Miami, Florida, January 2003.

[PFK03] P. Pu, B. Faltings, and P. Kumar. User-Involved Tradeoff Analysis in Configuration Tasks. In *Proceedings of the 3rd International Workshop on User-Interaction in Constraint Satisfaction*, pages 85–102, Kinsale, Ireland, September 2003.

[PKC+01] F. Perich, L. Kagal, H. Chen, S. Tolia, Y. Zou, T. Finin, A. Joshi, Y. Peng, R. Scott, and C. Nicholas. ITTALKS: An Application of Agents in the Semantic Web. In *Proceedings of the 2nd International Workshop on Engineering Societies in the Agents World*, pages 175–194, Prague, Czech Republic, July 2001.

[PKCH05] J. Pathak, N. Koul, D. Caragea, and V. Honavar. A Framework for Semantic Web Services Discovery. In *Proceedings of the 7th ACM International Workshop on Web Information and Data Management (WIDM 2005)*, pages 45–50, Bremen, Germany, November 2005.

[PL04] T. R. Payne and O. Lassila. Semantic Web Services. *IEEE Intelligent Systems*, 19(4):14–15, January/February 2004.

[Pow05] S. Powers. *What Are Syndication Feeds*. O'Reilly Media, Sebastopol, California, 2005.

[PPW03] G. Papamarkos, A. Poulovassilis, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In I. Cruz, V. Kashyap, S. Decker, and R. Eckstein, editors, *Proceedings of the 1st International Workshop on Semantic Web and Databases (SWDB 2003)*, pages 309–327, Humboldt-Universität, Berlin, Germany, September 2003.

[PSS02] T. R. Payne, R. Singh, and K. Sycara. Calendar Agents on the Semantic Web. *IEEE Intelligent Systems*, 17(3):84–86, May-June 2002.

[RMRD+06] J. Riemer, F. Martin-Recuerda, Y. Ding, B. Sapkota, R. Krummenacher, O. Shafiq, D. Fensel, and E. Kühn. Triple Space Computing: Adding Sematics to Space-Based Computing. In *Proceedings of the 1st Asian Semantic Web Conference (ASWC 2006)*, pages 300–306, Beijing, China, September 2006.

[RSS07] RDF Rich Site Summery. Website, 2007. http://xml.coverpages.org/rss.html.

[Rus04] V. Rus. A First Evaluation of Logic Form Identification Systems. In *Proceedings of the 3rd International Workshop on Evaluation of Systems for Semnatic Analysis for Text*, pages 37–40, Barcelona, Spain, July 2004.

[Shi06] A. Shia. A Novel Personal Agent Framework for Web Services and Commercial Systems. In *Proceedings of 2006 International Conference on Semantic Web & Web Services (SWWS 2006)*, pages 143–148, Las Vegas, Nevada, June 2006.

[SHP03] E. Sirin, J. Hendler, and B. Parsia. Semi-Automatic Composition of Web Services using Semantic Descriptions. In *Proceedings of the 1st Workshop on Web Services: Modeling, Architecture and Infrastructure (WSMAI 2003), In conjunction with ICEIS 2003*, pages 17–24, Angers, France, April 2003.

[SHS+02] T. Sollazzo, S. Handschuh, S. Staab, M. R. Frank, and N. Stojanovic. Semantic Web Service Architecture – Evolving Web Service Standards toward the Semantic Web. In *Proceedings of the 15th International Florida Artificial Intelligence Research Society Conference*, pages 425–429, Pensacola Beach, Florida, May 2002.

[SKB06] B. Sapkota, E. Kilgarriff, and C. Bussler. Role of Triple Space Computing in Semantic Web Services. In *Proceedings of the 8th Asia-Pacific Web Conference (APWeb 2006)*, pages 714–719, Harbin, China, January 2006.

[SKMR+06] O. Shafiq, R. Krummenacher, F. Martin-Recuerda, Y. Ding, and D. Fensel. Triple Space Computing Middleware for Semantic Web Services. In *Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference Workshops (EDOCW 2006)*, pages 15–18, 2006.

[SL01] S. Shearin and H. Lieberman. Intelligent Profiling by Example. In *Proceedings of the 6th International Conference on Intelligent User Interfaces*, pages 145–151, Santa Fe, New Mexico, January 2001.

[SMWM06] U. Srivastava, K. Munagala, J. Widom, and R. Motwani. Query Optimization over Web Services. In *Proceedings of the 32nd International Conference on Very Large Databases (VLDB 2006)*, pages 355–366, Seoul, Korea, September 2006.

[SP01] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In *Proceedings of the the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2001)*, pages 227–238, Washington, DC, USA, 2001.

[SPW+04] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN Planning for Web Service Composition using SHOP2. *Journal of Web Semantics*, 4(1):377–396, 2004.

[SSK+06] O. Shafiq, F. Scharffe, R. Krummenacher, Y. Ding, and D. Fensel. Data Mediation Support for Triple Space Computing. In *Proceedings of the 2nd IEEE International Conference on Collaborative Computing (CollaborateCom 2006)*, pages 2–28, Atlanta, Georgia, November 2006.

[UDD03] The Universal Description, Discovery and Integration (UDDI) protocol, version 3, 2003. http://www.uddi.org/.

[VSS+05] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and J. Miller. METEORS WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.

[W3C03] SOAP 1.2, W3C Recommendation. Website, 2003. http://www.w3.org/TR/soap12-part0/.

[W3C05] W3C. Semantic Web Services Framework. Website, 2005. http://www.w3.org/Submission/SWSF.

[W3C06] Extensible Markup Language (XML 1.0). Website, 2006. http://www.w3.org/TR/REC-xml/.

[W3C07a] Extensible Stylesheet Language Transoformation (XSLT 2.0). Website, 2007. http://www.w3.org/TR/2007/REC-xslt20-20070123/.

[W3C07b] W3C. Semantic Annotations for WSDL and XML Schema. Website, 2007. http://www.w3.org/TR/sawsdl.

[WC95] J. Widom and S. Ceri. *Active Database Systems*. Morgan–Kaufmann, San Mateo, California, 1995.

[WCRS04] J. Wohltorf, R. Cissée, A. Rieger, and H. Scheunemann. BerlinTainment: An Agent-Based Serviceware Framework for Context-Aware Services. In *Proceedings of the 1st International Symposium on Wireless Communication Systems (ISWCS 2004)*, pages 245–249, Piscataway, New Jersey, September 2004.

[WS05] W3C. Web Services Activity home page. Website, 2005. http://www.w3.org/2002/ws.

[WS06] OASIS Web Services Security Technical Committee. Website, 2006. http://www.oasis-open.org.

[XML06] Crimson: A Java XML 1.0 parser. Website, 2006. http://xml.apache.org/crimson/.

[YL04] T. Yu and K. Lin. Service Selection Algorithms for Web Services with End-to-End QoS Constraints. In *Proceedings of the IEEE International Conference on E-Commerce Technology (CEC 2004)*, pages 129–136, San Diego, California, July 2004.