Conceptual Model Based Semantic Web Services

Muhammed Al-Muhammed* and David W. Embley*
Department of Computer Science
Brigham Young University, Provo, Utah 84602, U.S.A.
mja47@cs.byu.edu, embley@cs.byu.edu

Stephen W. Liddle*,†
School of Accountancy and Information Systems
Brigham Young University, Provo, Utah 84602, U.S.A.
liddle@byu.edu

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.

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

*Supported in part by the National Science Foundation under grant No. IIS-0083127.
†Supported in part by the Rollins Center for eBusiness at BYU under grant No. EB-05046.

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 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 3); task recognition, which finds the domain of a specified task and specializes processes to perform the

task (Section 4); and task execution (Section 5). We give the status of our prototype implementation along with the future work we are considering in Section 6, and we conclude with Section 7.

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, KB02, KB04]. 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.

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.

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 4.1 and 4.2. In Section 4.3, we describe how our approach determines which task ontology to use, among the many we assume exist.[1]

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 1 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

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

[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.

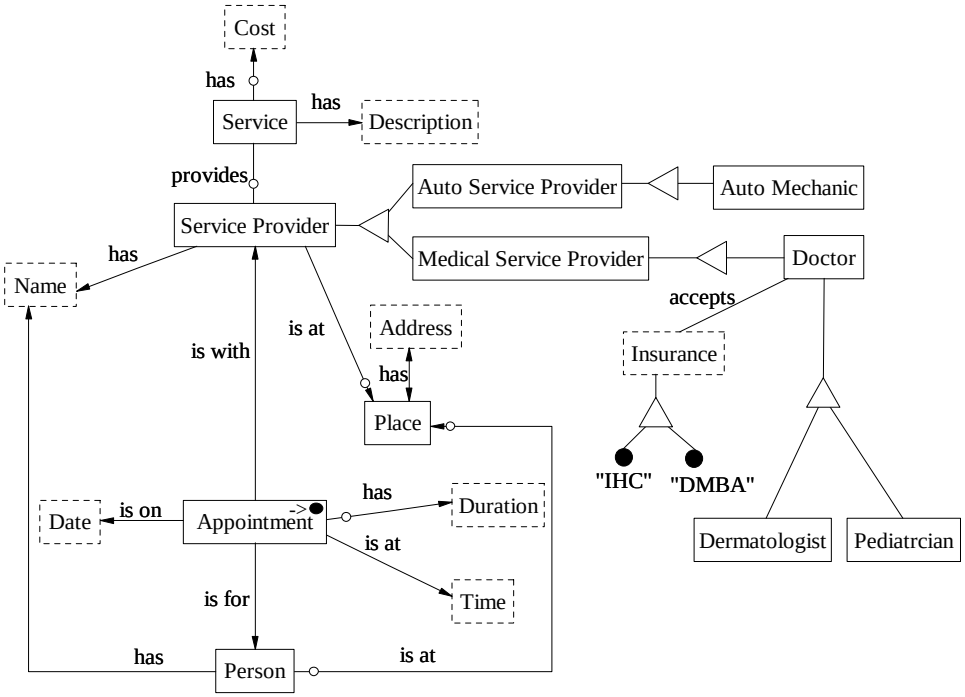


Figure 1: A conceptual-model representation of a domain ontology for appointments (partial).

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 domain ontology by marking it with "–> •" in the upper right corner.[3] We designate the concept *Appointment* in Figure 1 as the main concept because this domain ontology is for making appointments. Figure 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

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

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 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*.

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

Date
...
NextWeek(d: Date)                          Tomorrow (s: String)
returns (Boolean)                          returns (Date)
context keywords/phrases: next week |      context keywords/phrases: tomorrow |next day |...
  week from now |...                       ...
end                                        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)       LessThanOrEqual(d1: Distance, d2: Distance)
returns (Boolean)                          returns (Boolean)
context keywords/phrases: less than |< |... context keywords/phrases: within |not more than |
...                                          ≤ |...
end                                        ...
                                           end

Dermatologist                                  Appointment
internal representation: object id             internal representation: object id
...                                            ...
context keywords/phrases: [Dd]ermatologist |skin  context keywords/phrases: appointment |
  doctor |...                                    want to see a[n]? |...
...                                            ...
end                                            end
```

Figure 2: Some sample data frames (partial). (The "..." in the textual-representation part of the *Time* data frame in Figure 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.)

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 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 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 *Distance* data frame in Figure 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 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.

4.2 Process Ontology

A *process ontology* describes an execution pattern in a domain. Figure 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 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 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 3), and transitions, represented as divided rectangles (e.g. the *@cre-*

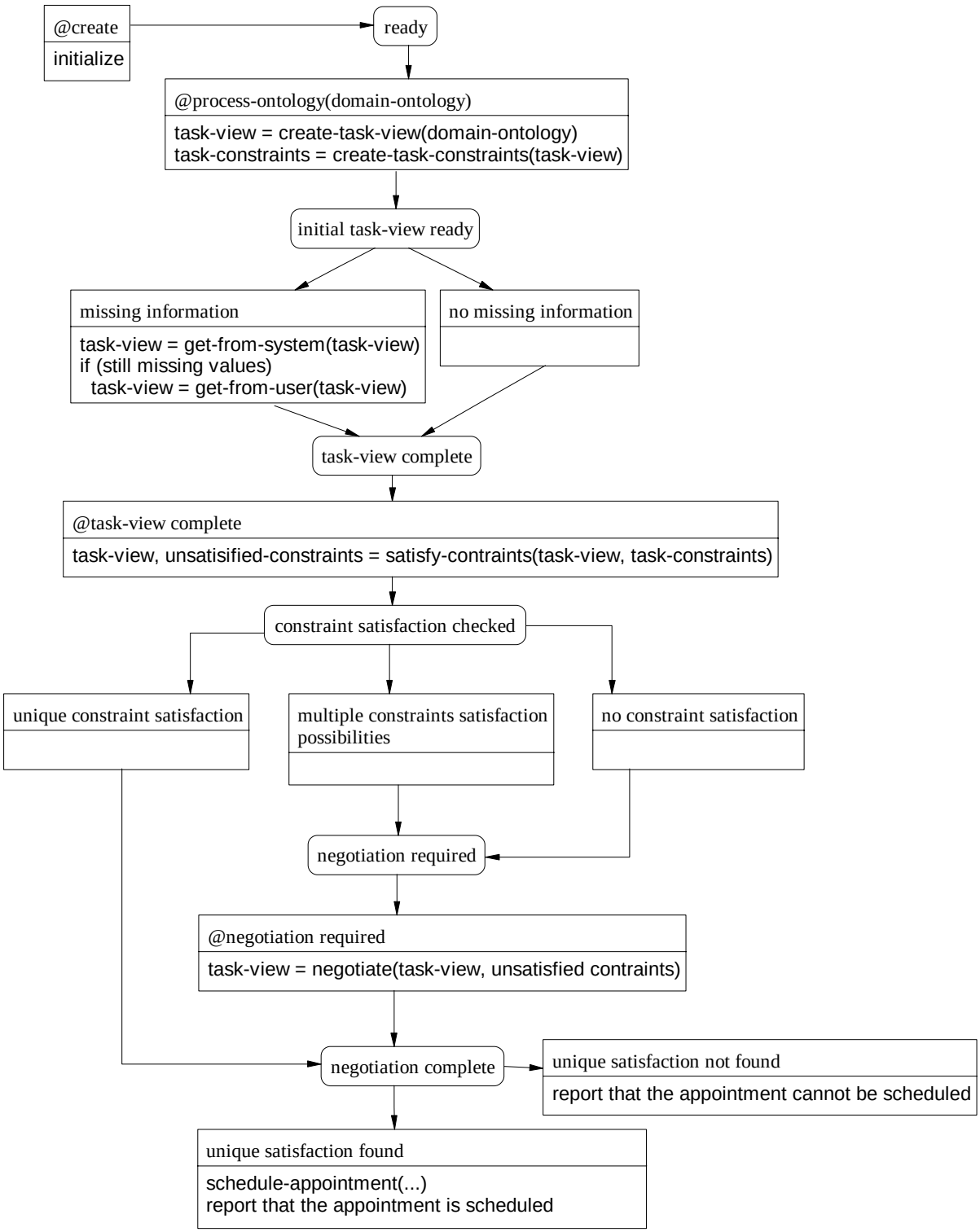


Figure 3: A process ontology specialized for scheduling appointments.

ate/initialize transition in Figure 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 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.

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 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.

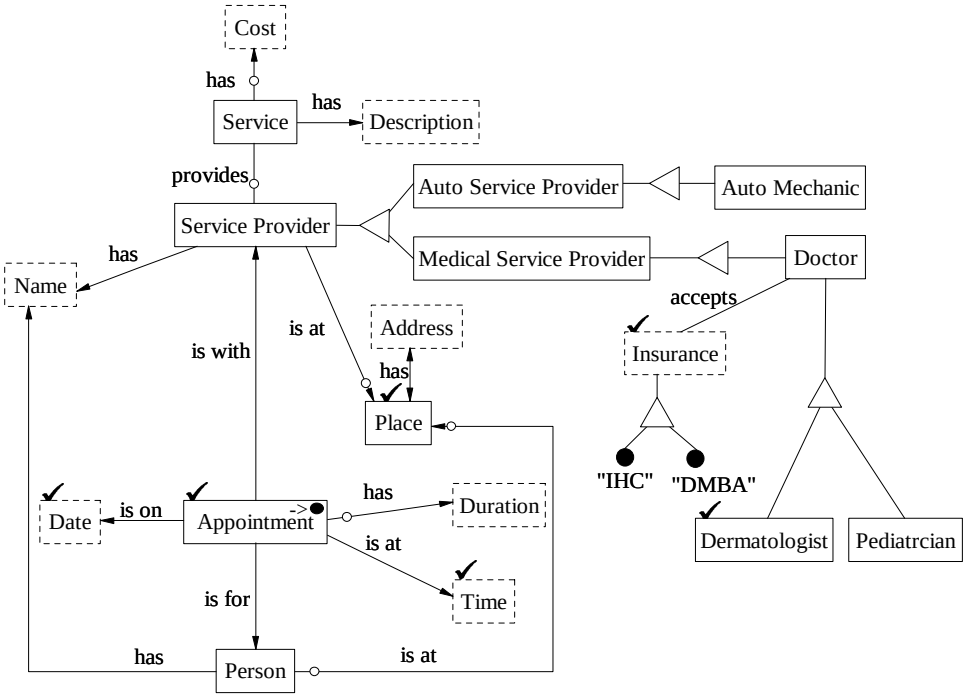


Figure 4: The output of the recognition-process.

Referring to our running example, when the recognition process executes for the domain ontology in Figure 1, the data frames in Figure 2, and the task specification in Section 3, it produces the output in Figure 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]

[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.

5 Task Execution

The process ontology is responsible for executing tasks. As mentioned in Section 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.

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 5. The process does not prune *Appointment* because it is the main concept. It does not prune *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.

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

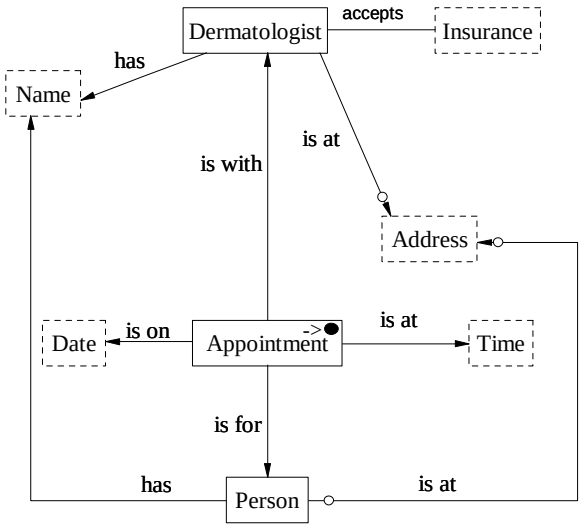


Figure 5: The task view for the specified task in Section 3.

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 shows, it is Boolean and one of its context phrases is "next week", which appears in the task specification in Section 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

[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.

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*. 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 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 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.

5.3 Obtaining Information from the System

Given the predicate calculus statement in Figure 6, the system can generate a query for the system's appointment databases.[7] Assuming that the appointment database has a view definition that

[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.

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

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

Figure 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.)

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

Figure 7: Generated predicate calculus statement.

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 7. In Figure 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 8, which we assume are the only partial interpretations returned as a result of executing the relational calculus query in in Figure 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 $Dermatologist_0$.

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

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

Figure 8: Partial interpretations.

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

[8]Verbalization according to [Hal04] and verbalization with respect to the model-equivalent language for OSM

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 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 3, then the system could have extracted this information, and could have executed without any need for asking the user for additional information.

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 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 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,

[LEW00] are examples of worked-out verbalizations that could be used in our application.

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.

5.6 Process Finalization

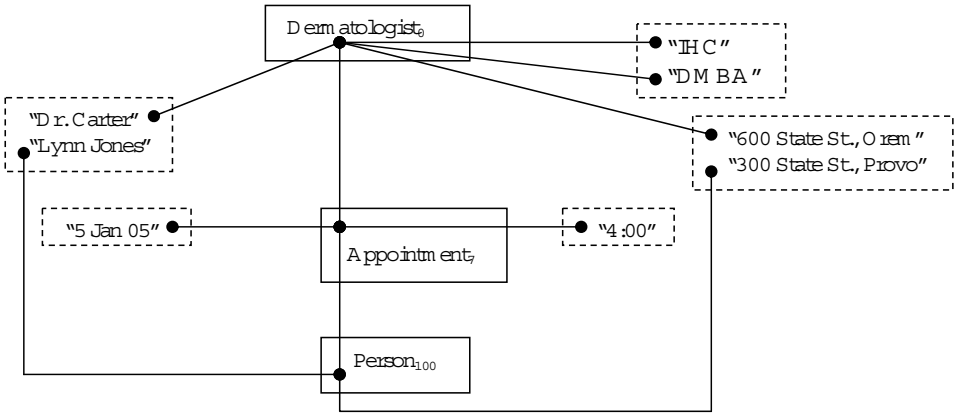
At this point in the process, the system either has a single solution or has agreed with the user that there is no solution. Hence, it is straightforward to know what to do. What is interesting here is that although this code cannot be written in advance because it does depend on the domain ontology,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

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.



$$
\begin{aligned}
&NextWeek(\text{"5 Jan 05"})\\
&Person(Person_{100})\ is\ at\ Address((\text{"300 State St., Provo"}) \wedge\\
&\quad Dermatologist(Dermatologist_{6})\ is\ at\ Address(\text{"600 State St., Orem"}) \wedge\\
&\quad LessThanOrEqual(DistanceBetween(\text{"600 State St., Orem"}, \text{"300 State St., Provo"}), \text{"6"})\\
&\exists\, x_6\ (Dermatologist(Dermatologist_{6})\ accepts\ Insurance(x_6) \wedge Equal(\text{"IHC"}, x_6))
\end{aligned}
$$

Figure 9: The scheduled appointment.

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].

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.

References

[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. *http://www-128.ibm.com/developerworks/library/specification/ws-bpel*, May 2003.

[AHS03] S. Agarwal, S. Handschuh, and S. Staab. Surfing the Service Web. In *Proceedings of the Second International Semantic Web Conference (ISWC2003)*, pages 211–226, Sanibel Island, Florida, October 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.

[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.

[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. *Available at http://informatik.uibk.ac.at/users/c70385/wese/wsmf.bis2002.pdf*, 2002.

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

[KB02] M. Klein and A. Bernstein. Searching for Services on the Semantic Web using Process Ontologies. In Isabel Cruz, Stefan Decker, Jerome Euzenat, and Deborah McGuinness, editors, *The Emerging Semantic Web-Selected papers from the first Semantic Web Working Symposium*, pages 159–172. IOS press, Amsterdam, 2002.

[KB04] M. Klein and A. Bernstein. 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.

[MDCG03] E. Motta, J. Domingue, L. Cabral, and M. Gaspari. IRS-II: A Framework and Infrastructure for Semantic Web Services. In *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, pages 306–318, Sanibel Island, Florida, 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. *http://www.daml.org/services/owl-s/1.0/*, 2004.

[PPW03] G. Papamarkos, A. Poulovassilis, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In Isabel F. Cruz, Vipul Kashyap, Stefan Decker, and Rainer Eckstein, editors, *Proceedings of the first 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. *http://www.w3.org/2002/ws*, 2005.