Ontology Aware Software Service Agents: Meeting Ordinary User Needs on the Semantic Web

A Dissertation Proposal Presented to the

Department of Computer Science

Brigham Young University

In Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

Muhammed Al-Muhammed

April 20, 2005

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 [SSG04, DW03, KKS+02, Hen01].

In this dissertation we offer a unique approach to turn the vision of semantic web pioneers into reality for everyday tasks such as scheduling appointments, selling, buying, 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 specify their needs. We intend to let users assume the existence of an intelligent agent within the

system and specify their needs textually in any way they wish.

2. *Task Recognition.* After specification of the task, our approach goes through a recognition process of the specified task. This recognition process matches task specifications against a task ontology to identify the goals of the specified task, the processes by which these tasks may be achieved, inputs, outputs, and requirements to be imposed on a task.

3. *Task Execution.* Given a specified task, the system generates a software agent that can do the task. The software agent has the ability to gather the information it needs by interacting with the system or the user or both in case of missing required information in the usual case of incomplete specification. 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.

The chief objective of this dissertation is to build a proof-of-concept prototype system to show that the vision of the semantic web for everyday tasks can be achieved.

2 Thesis Statement

It is possible to automate everyday tasks, such as scheduling appointments, buying and selling, and so forth, by creating systems that build on current advances in both software agents and the semantic web. These systems use task ontologies as their foundational knowledge to identify users' needs and task requirements to meet these needs. Identified tasks are assigned to generated task-specific agents that accomplish the tasks using semantic web facilities. 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 a software agent capable of performing the task.

3 Research Description

The dissertation has three major parts: (1) task specification, which allows users to textually specify tasks, (2) task recognition, which finds the domain of a specified task and the required processes to do it, and (3) task execution. We discuss these topics in the following three subsections. In Section 3.4, we compare our approach to related work.

3.1 Task Specification

We explain task specification using an example, which will be our running 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 must be within 20 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.

3.2 Task Recognition

The main objective of a task recognition is to determine the domain of a specified task. Our approach employs a task ontology for this objective. Therefore, we first introduce the two components of a task ontology, namely a *domain ontology* and a *process ontology*, in Sections 3.2.1 and 3.2.2. In Section 3.2.3, we describe how our approach determines which task ontology to use.

3.2.1 Domain Ontology

A *domain ontology* consists of concepts that can be found in the domain of a task ontology along with relationships among these concepts and constraints over these concepts and relationships. More precisely, 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—in practice, we need a much larger and richer ontology. 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 three types of concepts, namely lexical concepts (enclosed in dashed rectangles), nonlexical concepts (enclosed in solid rectangles), and single concepts (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, such as "2:00 PM" or "2:00 p.m.", represent themselves. A concept is *nonlexical* if its instances are object identi-

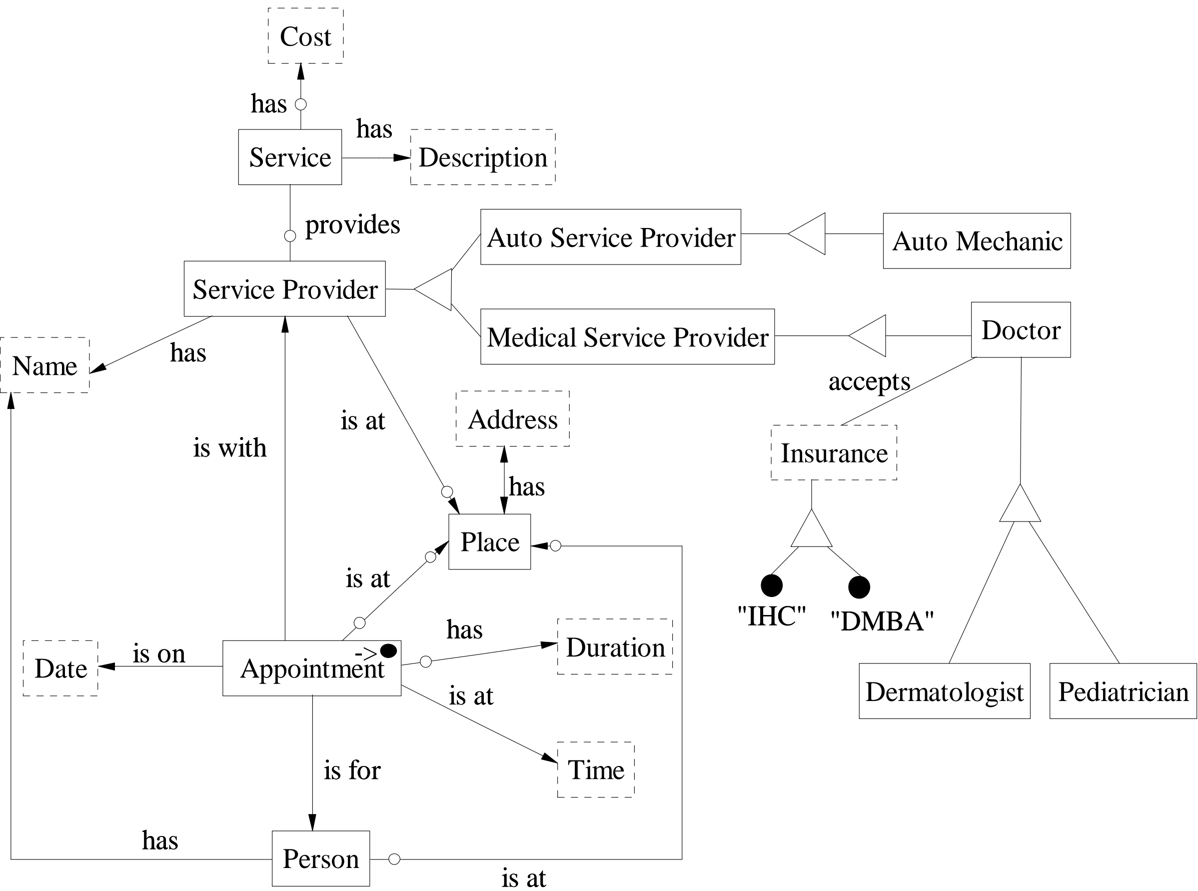


Figure 1: A generic hypergraph representation of a domain ontology for appointments (partial).

fiers (not self identifying representations). *Dermatologist* is an example of a nonlexical concept because its instances are identifiers such as, say, "Dermatologist100". The single object is an object set with a single element in it (for example, the single object "DMBA"). We designate the main concept in a domain ontology by marking it with "–> •" in the upper right corner.[1] For example, we designate the concept *Appointment* in Figure 1 as the main concept because this domain ontology models 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 repre-

[1]This notation denotes that when this ontology is used to create an appointment, the object set *Appointment* becomes ("–>") an object ("•"). A large black dot denotes an object or an object set with a single object in it.

sent 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 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 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*.

We augment a domain ontology with data frames [Emb80]. A *data frame* defines 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*. *Time*'s 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"). We use regular expressions to capture external representations. The "..." in *Time*'s data frame in Figure 2 indicates that there are other representations of *Time* such as military time that need to be captured. A data frame's context keywords/phrases are also regular expressions (often simple lists of keywords/phrases) separated with "|". For example, *Distance*'s 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,

```
Time
...
textual representation: ([2-9]|1[012]?)\s*:\s*([0-5]\d)\s*[AaPp]\s*[.]?\s*[Mm]\s*[.]? | ...
...
end

Date
...
NextWeek(d1: Date, d2: 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

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

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

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

Figure 2: A sample of data frames.

it is likely that this number is an instance of *Distance*. A data frame's permissible operations can manipulate a concept's instances. For example, *Distance*'s data frame includes the operation *LessThan* that takes two instances of *Distance* and returns a Boolean (true or false). The operations context keywords/phrases indicate an operation's applicability, for example, context keywords/phrases such as "less than" and "<" apply to the *LessThan* operation. A nonlexical concept that has only identifiers such as *Dermatologist*, often only has context keywords or phrases. Figure 2 shows the *Dermatologist* data frame, which includes a regular expression that

consists of its identifying name and its synonyms.

Some constraints in our conceptual model are too rigid. For example, service providers usually only have one place where they work, and thus *Service Provider is at Place* is functional, but a few may have an additional place or two where they work. This needs further research, but we are currently thinking of adding a probability declaration. Thus, for *Service Provider is at Place*, we would attach something like (0.99) as a probability declaration. The conceptual model also lets us declare additional constraints known as "general constraints" because they are declared generally, outside of the provided notation. For example, the constraint $\textit{Appointment has Place} = \pi_{Appointment,Place}(\textit{Appointment is with Service provider} \bowtie \textit{Service Provider is at Place})$ declares that an appointment is at a service provider's place. Like some other constraints, this constraint is mostly true, but may not be always true—the appointment could be at another place agreed upon by a *Person* and a *Service Provider.* Therefore, we also need to investigate the possibility of loosening these constraints, most likely by also adding a probability declaration (e.g. perhaps (0.93) for this example).

The notation we use to declare domain ontologies is based on OSM [EKW92]. Although the web ontology language, OWL [W3C04], is currently in vogue, we use OSM because it is more succinct and readable for presentation purposes. Furthermore, we wish to use OSM in our implementation because we have developed tools to support our work. Indeed, OWL in its current state is unable to satisfy our needs because it does not provide support for data frames. In order to not altogether ignore the current W3C standard, however, we plan to provide a transformation to OWL. Another project in our research group [Din05] has as one of its objectives an augmentation of OWL for data frames, and we thus should be able to fully convert OSM ontologies to augmented OWL ontologies.

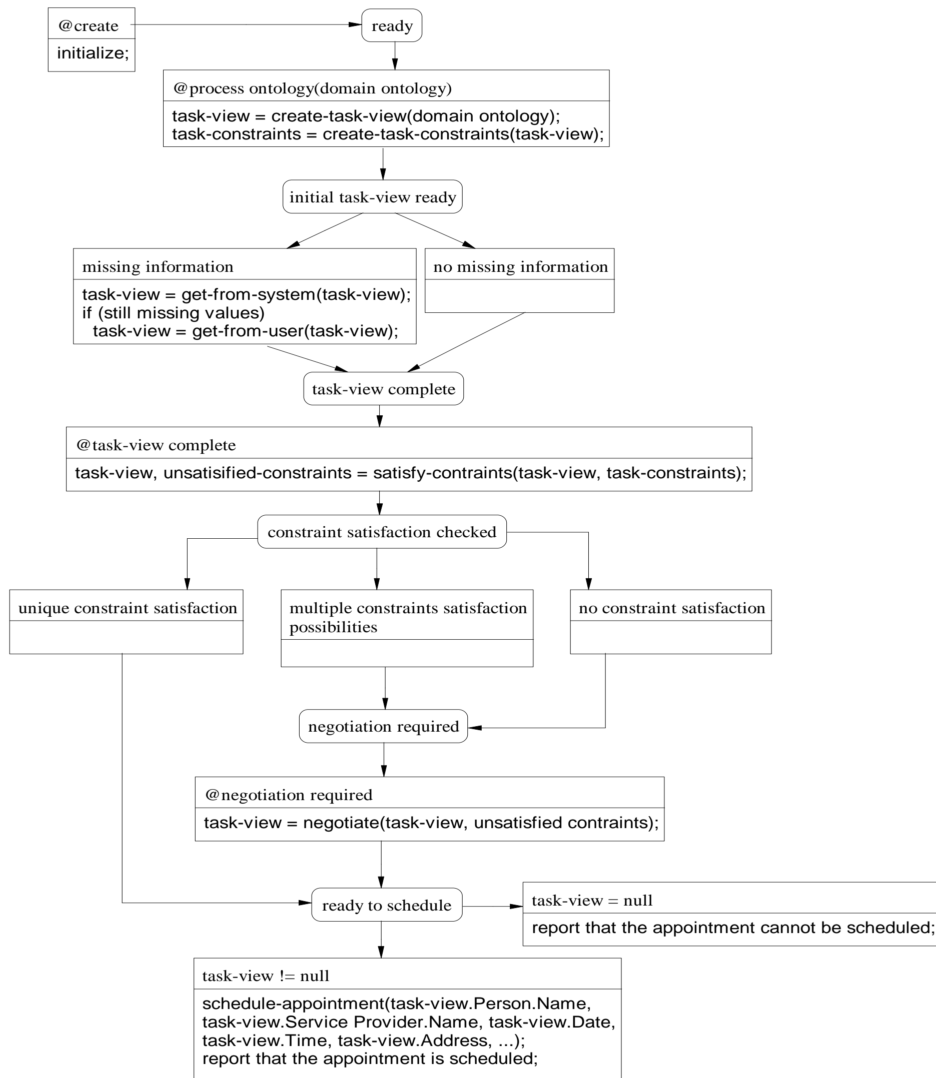


Figure 3: A domain-specific process for scheduling appointment.

3.2.2 Process Ontology

A *process ontology* is a description of a process that executes tasks in a domain. Figure 3 shows the process ontology for scheduling appointments represented as a statenet [EKW92], a representation that lets us specify standard Event-Condition-Action (ECA) rules [WC95, PPW03]. In this section, we provide a high-level description of the process ontology, leaving the details of the subprocesses on which the process ontology depends to be discussed in Section 3.3. As we shall see, all of these subprocesses are domain-independent. Domain-independence makes

it possible to automatically generate process ontologies without having to write any code.

The input for a process ontology is a domain ontology like the domain ontology in Figure 1 except that concepts that match with the task specification are marked, and recognized values are assigned to matching concepts. The process consists of states, such as *initial task-view ready* and *task-view complete*, and transitions represented as divided rectangles. In the top part, we specify triggers, which are events or Boolean 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 becomes active when the event occurs (is called from some other process), and the latter is a non-parameterized event that becomes active when the task-view is complete. In the bottom part of the divided rectangles, we specify actions or subprocesses. The actions in a particular transition execute when the trigger of the transition becomes active. Examples of subprocesses include *create-task-view(domain ontology)* and *get-from-system(task-view).*

The general control of the process ontology flows as follows. The process ontology uses the subprocess *create-task-view(domain ontology)* to create a task-view, which is the part of a domain ontology that matches with the task, and uses the subprocess *create-task-constraint(task-view)* to enumerate the applicable constraints for the task. If all the task-view concepts have values, the process ontology enters the *task-view complete* state; otherwise it obtains values for these concepts from system repositories using the subprocess *get-from-system(task-view)* and values from the user it cannot obtain from system repositories using the subprocess *get-from-user(task-view).* Next, the process ontology checks the constraints using the process *satisfy-constraints(task-view, task-constraints)* and enters the *constraint satisfaction checked* state. If there is unique constraint satisfaction, the process enters the *ready to schedule* state; otherwise if there are multiple ways the constraints can be satisfied or if there is no way to satisfy the constraints, the process ontology enters the *negotiation required* state. Finally, if the task-view

is not null, the process ontology schedules the appointment.

Although we use an OSM statenet to represent a process ontology, we will provide a transformation from an OSM statenet to OWL-S [MPM+04, W3C04, W3C01]. This will allow us to represent our services in a standardized web service language.

3.2.3 Task Ontology Recognition

The task ontology recognition process determines the domain of a task. The recognition process takes a set of 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 three phases. First, for each domain ontology, the recognition process applies concept recognizers in the data frames of every concept of the domain ontology to the task specification and marks every concept that matches with a substring in the task-specification. Second, the process computes the rank of a domain ontology with respect to a task specification. We are currently thinking of computing the rank based on the number of the marked concepts. We feel, however, that the computation of the rank may require more sophisticated technique such as giving weight to some concepts of a domain ontology depending on their relevancy. For example, the concepts *Appointment*, *Date*, and *Time* of the domain ontology (Figure 1) should probably have higher weight than *Insurance* or *Cost* because they are more relevant to scheduling appointments. This, however, needs more investigation. Third, the recognition process sorts the domain ontologies according to their ranks in descending order and selects the domain ontology with the highest rank. This process assumes nothing about the domains of the ontologies nor about the task specification, and therefore it is domain independent.

Referring to our running example, when the recognition process is called with the domain ontology in Figure 1, the data frames in Figure 2, and the task specification in Section 3.1 as input, it produces the output in Figure 4. The concept recognizers in the data frame for *Derma-*

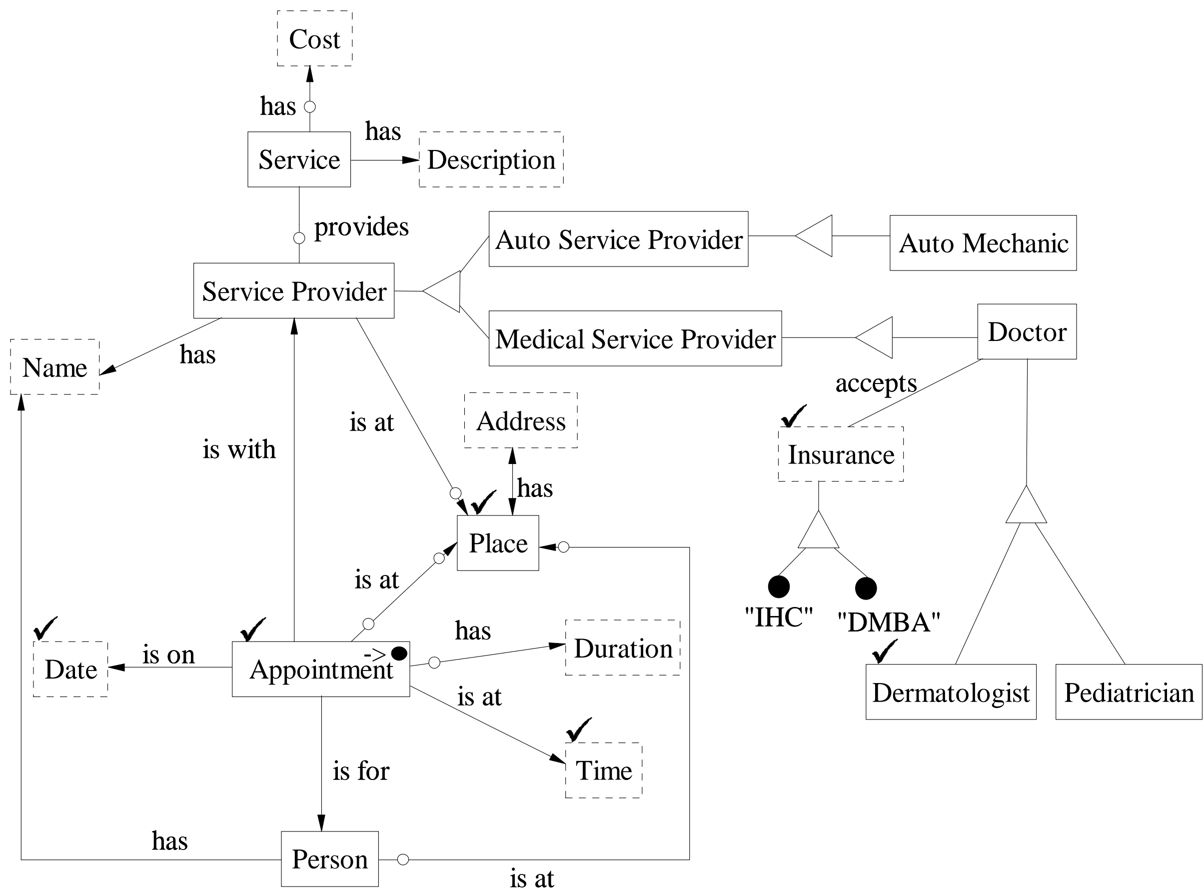


Figure 4: The output of recognition-process.

tologist recognize 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 *Distance* data frame recognizes “within”, “20”, and “miles”; in the *Appointment* data frame recognizes “want to see a”; in the *Place* data frame recognizes “my home”; and in the *Insurance* data frame recognizes “insurance”; and therefore these concepts are marked. It is worth noting that the output in Figure 4 is exactly the domain ontology in Figure 1 except that some of the concepts are marked.

3.3 Task Execution

The process ontology is responsible for executing tasks. As we mentioned in Section 3.2.2, the process ontology depends on domain-independent subprocesses. In this section we discuss these subprocesses, justifying why they are domain independent. We use our running example to illustrate them.

3.3.1 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 iterates over all of the concepts of a domain ontology and keeps the main concept of the domain ontology (the concept marked with "–> •"). It also keeps all the main concept's mandatory concepts as well as the marked concepts, and prunes away all other concepts (along with all their relationships). In addition, the process replaces a generalization concept by its marked specialization and replaces a nonlexical by a lexical concept when there is a one-to-one correspondence. The remaining concepts in the domain ontology are called the task view. Observe that this process is domain independent because it assumes nothing about the domain of its input.

Referring to our running example, the *create-task-view* process in Figure 3 specializes this process by passing to it the domain ontology in Figure 4. The result is the task view in Figure 5. *Appointment* is not pruned because it is the main concept. *Date*, *Time*, *Name*, *Service Provider*, and *Person* are not pruned because they are mandatory concepts for the main concept; *Dermatologist*, *Insurance*, and *Place* are not pruned because they are marked. In addition, the specialization *Dermatologist* replaces its generalization *Service Provider* because this specialization is marked and the lexical concept *Address* replaces the nonlexical concept

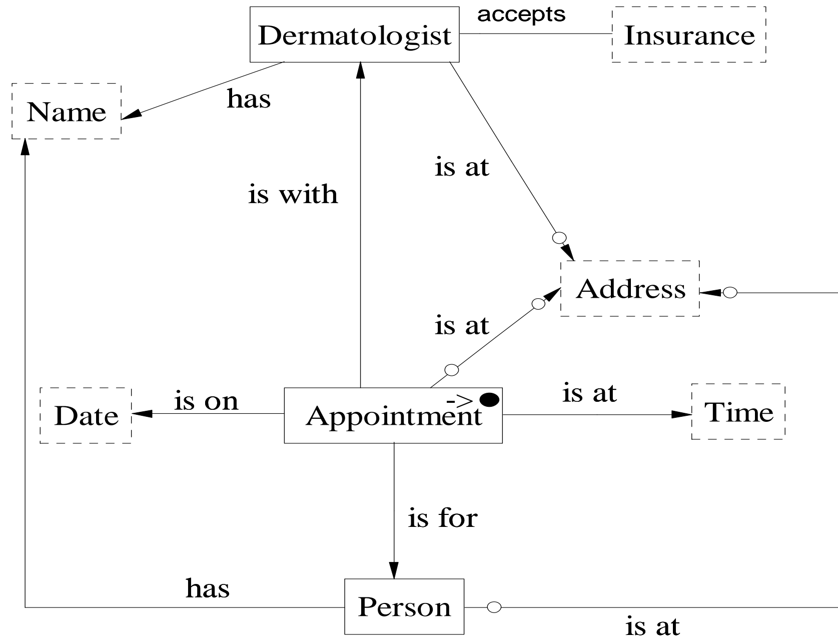


Figure 5: The task view for the specified task in Section 3.1.

Place with which it has a one-to-one correspondence.

We emphasize the need for reasoning to avoid including spurious marked concepts in the task view. For example, suppose *Insurance Salesperson* were included as a specialization of *Service Provider* in Figure 4. If so, it would likely be marked based on the appearance of "insurance" in the task specification in Section 3.1. To decide whether to include or exclude the object set *Insurance Salesperson* in the task view, some reasoning is necessary. The *Appointment* has only one *Service Provider* because the relationship set *Appointment is with Service Provider* is functional from *Appointment* to *Service Provider.* Since the service provider *Dermatologist* is marked and should be included,[2] the process can conclude that including the *Insurance Salesperson* causes the *Appointment* to have two service providers, which violates the constraint. Therefore the system prunes away the object set *Insurance Salesperson.* The

[2]Giving precedence to *Dermatologist* over *Insurance Salesperson* requires another kind of reasoning. We can reason that because a dermatologist keyword appears next to an appointment keyword or because a dermatologist keyword appears twice, it should have precedence over *Insurance Salesperson.*

required reasoning power needs further investigation, but we believe that the information in a domain ontology along with the information extracted from a task specification can likely provide enough knowledge to do adequate reasoning to resolve these problems.

3.3.2 Task Constraint Creation

Task constraint creation enumerates the constraints imposed on a task using a task view and the operations in the data frames associated with the concepts of the matching ontology. Creating a process to generate the right set of constraints is likely to face several challenging problems such as generating spurious Boolean operations that are likely to match substrings in the task specification, constraining the many degrees of freedom in loose task specification, determining the right sources of values for input parameters of the constraints, and determining free and bound parameters. Although finding the solutions for these problems needs further research, we are currently thinking that the process basically operates as follows.

1. It lists all the operations in the data frames whose recognizers match substrings in the task specification and whose return types are Boolean. In addition, when a recognizer in the data frame of a concept in the task view matches with a value for that concept in the task specification, the process places this value in the concept. Referring to our example, the process lists the Boolean operations: *NextWeek*(d_1*: Date,* d_2*: Date)*, *Within(d: Distance, "20")*, and *Equal*(i_1*: Insurance,* i_2*: Insurance)* as constraints because they match substrings in the task specification. Since the concept recognizer in the *Time* data frame matches the value "4:00" in the task specification, the process places "4:00" in the *Time* concept, which constrains appointment times to 4:00 pm.

2. For each listed constraint, 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

Date(d1: Date) and NextWeek(d1: Date, d2: Date)
Person(x) is at Address(a_1) and Dermatologist(y) is at Address(a_2) and
Within(DistanceBetween(a_1, a_2), "20")
$\exists$ i_2 (Dermatologist(y) accepts Insurance(i_2) and Equal(i_1, i_2))

Figure 6: Constraints added to the task view.

view, the process should 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 and replace the input parameter with this operation. Referring to our example, *Within(d: Distance, "20")* has the input d of type *Distance*. Since *Distance* does not belong to the task view (see Figure 5), the process replaces the input by the operation *DistanceBetween(a_1: Address, a_2: Address)* because this operation returns a value of type *Distance*.

3. To determine the source of values for the input parameters of the constraints, the process can make use of the relationships in the task view. For example, the operation *DistanceBetween(a_1: Address, a_2: 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, therefore, can make use of this information to determine that the value of one of the address parameters comes from the relationship *Dermatologist is at Address* and value of the other comes from the relationship *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 the relationship *Dermatologist accepts Insurance* and leaves the other as a free variable. Also, since the relationship *Dermatologist accepts Insurance* is many-many, the process binds the parameter i_2 with the existential

quantifier to declare that any one value of i_2 that satisfies $\exists$ i_2 *(Dermatologist(y) accepts Insurance(i_2) and Equal(i_1, i_2))* is enough. Figure 6 shows the final set of constraints, which became part of the task view.

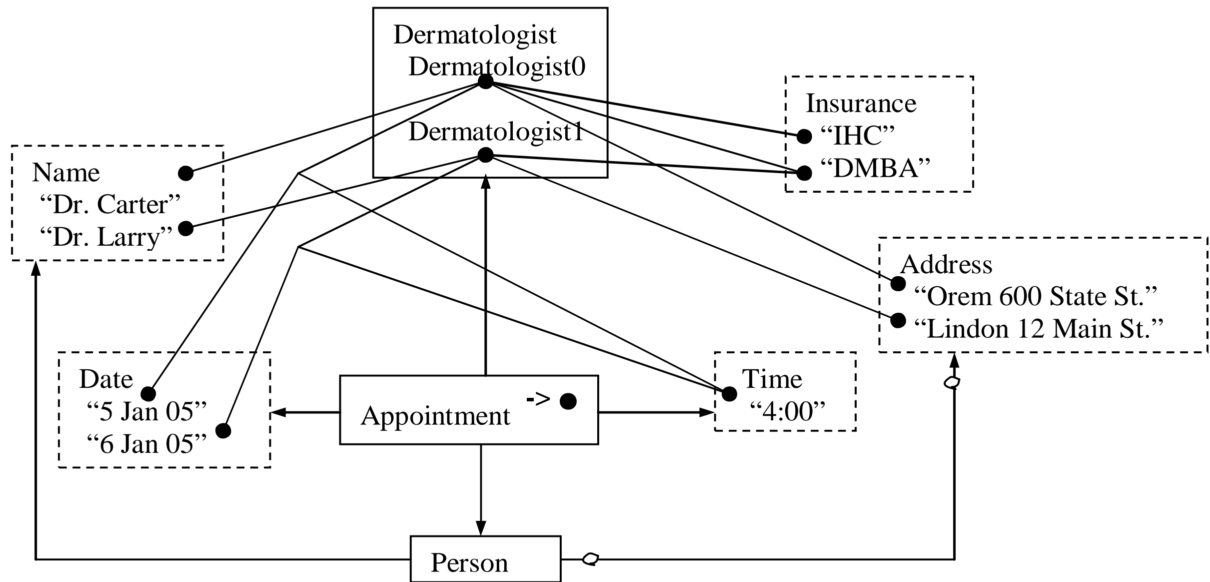
This process is domain independent because its algorithms are the same for all domains. The process makes use of only the information provided by the task view and the associated data frames. Once these are available, the process can discover constraints using fixed algorithms that work for all domains.

3.3.3 Obtaining Information from the System

This process takes a task view, augmented with discovered constraints, as input and uses the system's databases[3] to obtain needed values it can find for concepts. Several problems arise such as determining which concepts in the task view need values from the system databases and which need values from outside sources, limiting the values and value combinations when many are possible, and deciding which constraints to observe while obtaining values. Although the solutions for these problems need further research, we are currently thinking that the process obtains values for each concept in the task view for which it can find values, taking into consideration the task-specification-imposed constraints that can be applied at this stage of task execution. Applicable task-specification-imposed constraints are the discovered constraints for which the process has the required values to instantiate their input parameters. The process returns a partially filled-in task view as output.

Referring to our running example, the *get-from-system* process in Figure 3 takes the task view in Figure 5 combined with the constraints in Figure 6 as input and produces a partially populated task view, such as the one in Figure 7 as output. Since this task view is part of the

[3]We assume that the system's databases store up-to-date, real-world instances of concepts of all domain ontologies known to the system.



Task–imposed constraints:
Date(d1: Date) and NextWeek(d1: Date, d2: Date)
Person(x) is at Address(a_1) and Dermatologist(y) is at Address(a_2) and
Within(DistanceBetween(a_1, a_2), "20")
$\exists$ i_2 (Dermatologist(y) accepts Insurance(i_2) and Equal(i_1, i_2))

Figure 7: A partially filled in task view after obtaining information from the system.

scheduling domain ontology, the process uses the database that stores values for this ontology to obtain values for the concepts in the task view. The process populates concepts in the task view with values that satisfy the constraints that can be applied at this stage. Because *Dermatologist0* is available at *"4:00"* on *"5 Jan 05"*, which respectively satisfy the *Time* constraint and *Date* constraint for *NextWeek("28 Dec 04", "5 Jan 05")* assuming today's date is December 28, 2004, the process places them in the task view. Similarly, the process places *Dermatologist1* in the task view because this dermatologist has an opening for an appointment on January 6, 2005 at 4:00 pm. The process obtains and places all *Name*, *Address*, and *Insurance* information for these two dermatologists into the task view. It cannot yet limit these values because the *Distance* and *Insurance* constraints cannot yet be applied since the process lacks the person's address and type of insurance.

The process is domain independent. Deciding which concepts need values depends solely on the mandatory participation constraints declared in the domain ontology. Obtaining values for concepts is a matter of interaction with the system databases. Further, since the types of input parameters of constraints are concepts in the task view, deciding which constraint to observe depends solely on whether the process can obtain values for all input parameters of the constraint, which can be done independently from the domain by matching the input parameters with the these concepts. Thus, we can code the *get-from-system* routine as a fixed routine, parameterized by the task view augmented by the set of constraints whose only free variables can be supplied by the database.

3.3.4 Obtaining Information from a User

At this point in the process, the system has a task view that is partially filled in with zero or more value sets that satisfy the constraints that can be satisfied by obtaining values from the system. In the next part of the process the system requests any remaining, missing information from the user. Determining what information to request is straightforward in the typical case of one or more value sets whose values cover all the same concepts and free variables. For non-typical cases where there are no value sets or where the value sets cover different concepts and free variables, more research is needed to resolve the problem of determining what information to request. In the typical case, the process should interact with a user and obtain a value (possibly values, in the case of a non-functional dependency) for each mandatory lexical concept in the task view that does not yet have a value and for each free variable in the constraints that does not yet have a value. The result is a task view with complete value sets—complete in the sense that there are values for all mandatory concepts and all free variables.

Referring to our running example, when the process is called with the partially filled in task view in Figure 7, it produces the output in Figure 8. Since the nonlexical concept *Person*,

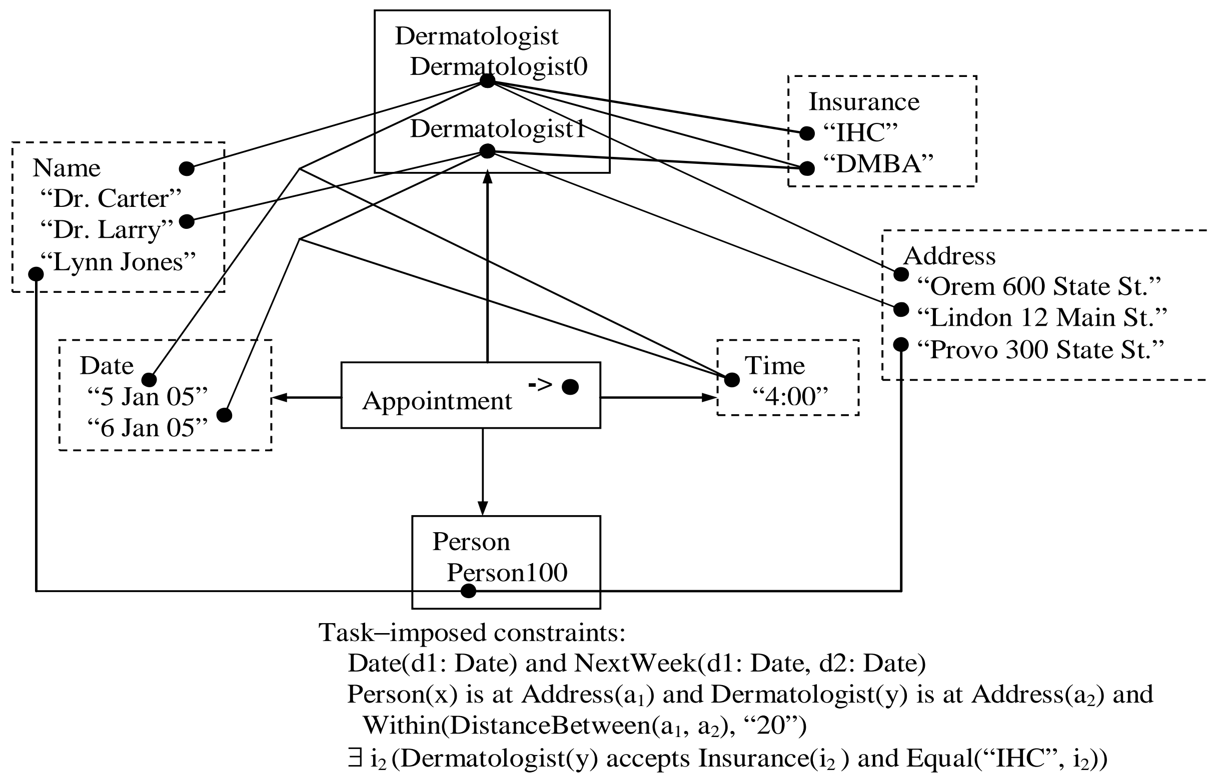


Figure 8: A completed task view after obtaining information from a user.

which is mandatory, does not have value (note, no dot inside the object set *Person*), the process creates a concept instance *Person100.* Because the concept *Name* is mandatory for *Person* and it is a lexical concept, the process initiates an interaction with the user to obtain value for it; we assume that the user enters the value *"Lynn Jones"* for the concept *Name.* Although *Address* is optional for *Person*, the process should obtain a value for it from the user because it is a free variable in a task-imposed constraints; we assume that the user enters the value *"Provo 300 State St."* for the concept *Address. Insurance* is not explicitly related to *Person*, but it is a free variable that does not yet have value in the task-imposed constraints, and therefore should have value; we assume that the user enters the value *"IHC"* for this constraint. The process does not, of course, ask a user for values for the other concepts such as *Dermatologist* because they already have values.

When the system can recognize which concepts and which free variables need values, the process of obtaining values from the user is domain independent. It only makes use of (1) concepts related immediately or transitively to the primary object of interest and whether these concepts are mandatory or optional whether these concepts are lexical or nonlexical, and (2) free variables in constraints. In situations where it may not be so clear which variables the user must enter, it will be important for us not to make the process domain dependent by, for example, listing the object sets that a user typically supplies in an interaction.

3.3.5 Constraint Satisfaction

The constraint satisfaction process determines which sets of values, if any, satisfy the constraints. We call sets of values that could potentially form a solution *potential solutions* because they are not solutions unless they satisfy the constraints. In our example, *Dermatologist0* with the name "*Dr Carter*" who accepts "*IHC*" and "*DMBA*" insurance, is at *"Orem 600 State St."*, and is available at *"4:00"* on *"5 Jan 05"* is a potential solution. The process basically works as follows. It instantiates the constraints with the values from each potential solution. If a potential solution satisfies all the constraints, the process ranks it with zero, meaning it violates no constraint. If, however, a potential solution violates one or more constraints, the process ranks it according to number of violated constraints and makes a note of the unsatisfied constraints. The process repeats for each potential solution. If a potential solution violates several more constraints than the best solution so far, the process discards it.

Referring to our running example, the process takes as input the value sets in the task-view in Figure 8. Figure 9(a) shows the constraints after the process instantiates them with the values from the first potential solution (*Dermatologist0*), and Figure 9(b) shows the constraints after the process instantiates them with the values from the second potential solution (*Dermatologist1*). The unsatisfied constraints are in bold in Figure 9. As shown, *Dermalogist0* satisfies

Person(Person100) is at Address("Provo 300 State St.") and Dermatologist(Dermatologist0) is at Address("Orem 600 State St.") and **Within(DistanceBetween("Provo 300 State St.", "Orem 600 State St."), "20")**	Person(Person100) is at Address("Provo 300 State St.") and Dermatologist(Dermatologist1) is at Address("Linden 12 Main St.") and **Within(DistanceBetween("Provo 300 State St.", "Lindon 12 Main St."), "20")**
(Dermatologist(Dermatologist0) accepts Insurance("IHC") and Equal("IHC","IHC") or Dermatologist(Dermatologist0) accepts Insurance("DMBA") and Equal("IHC","DMBA"))	(Dermatologist(Dermatologist1) accepts Insurance("DMBA") and **Equal("IHC","DMBA")**)
(a)	(b)

Figure 9: The instantiated constraints.

all of the constraints except the *Distance* constraint because the computed distance by the operation *DistanceBetween(...)* is "22" which is not $\leq$ "20" and thus receives a rank of 1, and *Dermatologist1* satisfies all of the constraints except the *Insurance* and *Distance* constraints and thus receives a rank of 2.

The process is domain independent. Instantiating input parameters of constraints is a matter of finding the concepts of the completely populated task view that match with the types of the input parameters and instantiating them with values from these concepts. Moreover, since all constraints are predicates, or Boolean operations, observing whether a constraint is satisfied only requires a simple check of the returned value.

3.3.6 Negotiation

The negotiation process attempts to reach an agreement with a user when multiple solutions satisfy the constraints or when no solutions satisfy the constraints. The process takes a (possibly empty) set of solutions as input and returns either a single, agreed upon solution or an empty set if the user either refuses all solutions or refuses to relax any constraints when there are no solutions.

The process has two states. It is in the state of negotiating with a user to relax the constraints if the set of unsatisfied constraints is not empty. In this state, we are thinking that the system would display the unsatisfied constraints (with liberal syntactic sugar) and prompt the user to relax a constraint in order to continue executing the task. If the relaxation still results in no solution, the system again prompts the user to relax a constraint or further relax the same constraint. If the relaxation results in a unique solution, this solution is returned and sent to the finalization process. If the relaxation results in several solutions, the negotiation process enters the state of negotiating with a user to select a solution from among the multiple solutions. In this state, the process displays the solutions (or some of the solutions, if there are many) and prompts the user to select a unique solution.

For our running example, there are no solutions, but there is one unsatisfied constraint *Within(DistanceBetween("Provo 300 State St.", "Orem 600 State St."), "20")*. Since there is an unsatisfied constraint, the negotiation process enters the state of negotiating with a user to relax constraints. If the user chooses to relax this constraint by allowing the distance to be within "22 miles" rather than "20 miles", the system has a solution for the task. For the purpose of completing the running example, we assume the user chooses to so loosen the constraint.

The negotiation process works the same for all domains. The set of unsatisfied constraints, if any, are all true/false statements, and each solution in the list of solutions is a set of attribute-value pairs. The system can manipulate both of these sets, as described, independent of the domain.

Several challenges arise that need further research. How does the system negotiate when there are a large number of solutions? Is there some way we can intelligently display just the top k? Exactly how does the system negotiate when there are no solutions? Which constraints should it display as candidates to relax? How can it allow a user to gradually relax constraints (e.g. in our example, relax the distance constraint to be just a little further as opposed to

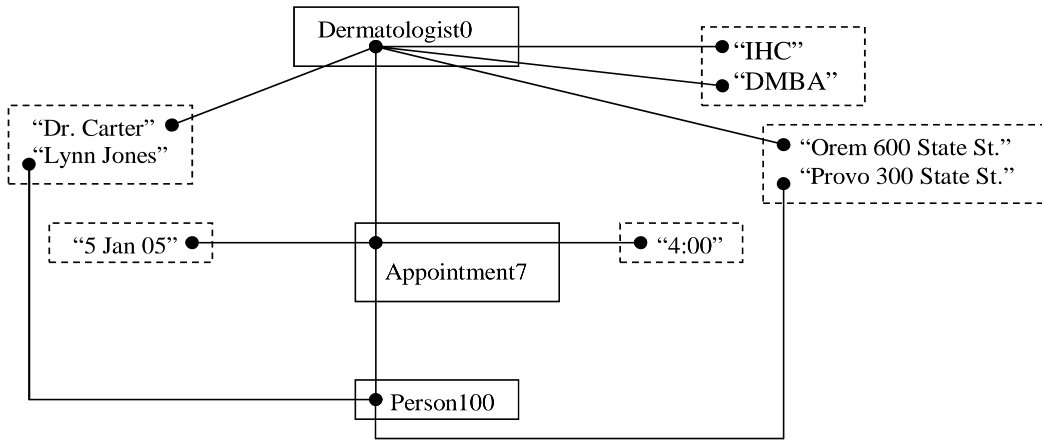


Figure 10: The scheduled appointment.

discarding the constraint altogether)?

3.3.7 Process Finalization

Finally, the process ontology enters the *ready to schedule* state. In our running example, since the negotiation process returns a solution, the condition *task-view != null* is *true* and the process ontology schedules the appointment using the action *schedule-appointment("Lynn Jones", ...)*. Figure 10 shows the scheduled appointment. As shown, *Appointment7* is scheduled for *Person100* whose name is "*Lynn Jones*", with *Dermatologist0* whose name is "*Dr. Carter*" on date "*5 Jan 05*" at time "*4:00*" at address "*Orem 600 State St.*". The process ontology notifies the user that the appointment is successfully scheduled.

The process *schedule-appointment(...)* is domain dependent because it needs knowledge about what object sets should be filled in with objects to schedule an appointment. However, since the required knowledge to do scheduling is already in the defined domain ontology, we can use this knowledge to automatically generate code for this process. Observe that this is true

for any domain so long as the objective is to insert an object into a single 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.

Moreover, we argued in the previous subsections (3.3.1 through 3.3.6) that all of the other processes in the process ontology are domain independent. As a result, we can make our general conclusion that the process ontology in Figure 3 is general and can be specialized automatically to any domain we wish by instantiating it with domain knowledge supplied as a domain ontology (as we did for our example for our appointment scheduling domain). The domain ontologies with which the systems works, however, must have a single primary object set (e.g. *Appointment* in our example), and the objective of the task ontology must be the insertion of an object into this primary object set.

3.4 Related Work

We know of no related work like the work we propose here. Researchers have, however, proposed systems similar to some aspects of the work we are proposing. We are not the first to have suggested the idea of a task ontology, nor the first to have proposed ways to let end users derive web services, nor the first to have thought of letting personal agents serve end-user needs.

Task Ontologies. 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." Although our approach also includes a task ontology, it differs significantly from this approach in that it does not have domain experts nor does it require users to compose components to do their tasks. [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 this approach because it does not require users to compose services. Both [KA04] and [MA02] describe 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 (PQL) 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.

Web Service Derivation. 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. [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 to the system and assists them to semi-automatically compose them to do some task. [SPW+04] describes a system that lets people select from a list of available services, and the system will execute the service and possibly decomposing this service into atomic executable services. Our approach differs significantly from these approaches in that our system does not require users to find or compose services from among a set of available services, which is difficult, even as acknowledged by the authors of the cited papers themselves, nor does it have the notion of pre-specified tasks. In fact, in our approach the only requirements are that users specify their tasks, provide information in a typical case of incomplete task specification, and negotiate when no solution because of strong imposed constraints or when many possible solutions due to the existence of many ways to satisfy the imposed constraints.

Personal Software Agents. Researchers have described implementations of personal agents that operate on behalf of their owners to do useful tasks. [CPC+04] describes 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 speakers' sound and the level of lights, and attempts to prepare the meeting room in an optimal fashion. [PSS02] describes an implementation for conference personal assistant agents that, given a URL, they bring scheduling information within conference schedules to its user's calendar. [PKC+01] describes an implementation of ITTalk, integrated applications using agent mediated services to disseminate event announcement. Users can provide their profiles, encoded in DAML, about presentations they are interested in and about which they 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 two approaches in that while these agents in these two approaches are preprogrammed to do pre-specified tasks, our system is more general in a sense that the tasks are defined in terms of knowledge rather than preprogrammed.

4 Research Plan

We plan to create a prototype system, test this prototype, and publish the results. Specifically, we plan to use the prototype to show that the techniques described in Section 3 can enable users to accomplish everyday tasks. We will conduct both black box testing and white box testing.

4.1 Black Box Testing

We will conduct several experiments on our prototype. The prototype will include task ontologies pertaining to different domains such as scheduling appointments, scheduling meetings, buying, selling, hiring, and so forth. The tests will be run by independent users who will specify tasks by writing them textually using the prototype interface. The performance of the prototype

will be measured based on the following.

For task recognition, we will measure the performance based on the accuracy of concept and constraint recognition and on recognition of the proper domain for the task. In all cases, we can measure based on recall and precision. Let N be the total number of task concepts (or constraints) that should have been recognized given a set of task specifications S. Then for concept recognition (constraint recognition),

$$recall = \frac{N_C}{N} \text{ and } precision = \frac{N_C}{N_C+N_I}$$

where N_C is the number of correctly recognized and N_I is the number of incorrectly recognized. Further, for domain recognition

$$recall = \frac{|S_C|}{|S|} \text{ and } precision = \frac{|S_C|}{|S_C|+|S_I|}$$

where S_C is the set of task specifications for which the prototype system correctly identified the domain and S_I is the set of task specifications for which the prototype system incorrectly identified the domain.

For task execution, we are interested in measuring the performance of the prototype based on the number of tasks that the prototype executes to completion. Let T be the number of tasks and T_a be the number of the completely accomplished tasks. We define prototype utility as

$$utility = \frac{T_a}{T}.$$

4.2 White Box Testing

We plan to develop white-box test cases designed in such a way that they force the system to go through all control paths and features in which we are interested. For example, there will be test cases that trigger negotiations with users to relax one constraint or several constraints.

Also, we will have test cases that force the system to negotiate with users to choose one solution from among several or many by gradually loosening constraints. We will observe user behavior and determine whether they can successfully use the system.

4.3 Delimitations

The following are beyond the scope of this dissertation.

1. Recognition and execution of a sequence of tasks.
2. Recognition and execution alternative tasks.
3. Recognition and execution of conditional tasks.
4. Vocal specification of tasks.

5 Research Papers

1. Ontology-Based Task Specification and Recognition
2. Ontology-Based Task Execution
3. Domain-Ontology Constraints Loosening to Meet Real-World Application Needs
4. Ontology-Based Automatic Code Generation for Domain-Specific Processes
5. OSM-OWL Domain Ontology Transformation
6. OSM-OWL Process Ontology Transformation

6 Contribution to Computer Science

As the number of services on the web continues to increase, users will increasingly suffer from issues such as finding and using appropriate services. Our approach uniquely addresses these

issues by turning their essence from finding and using to only assuming the existence of an intelligent agent and textually specifying tasks. Moreover, our approach depends only on both domain-independent processes that can be automatically specialized to a domain and a domain dependent process that can be automatically generated for a domain. This makes our approach work across domains without need for manual configuration. Finally, since our approach centers around task ontologies, the behavior of the prototype is only limited by the richness of the task ontologies, which can be independently enriched by the system specialist.

7 Dissertation Schedule

The tentative schedule of the dissertation will be as follows.

1. Task-Specification and Recognition (by August 2005)
    - (a) Domain Ontologies Creation
    - (b) Domain Ontologies Constraint Loosening
    - (c) Task Ontology Recognition
2. Process Ontologies (by March 2006)
    - (a) Domain-Independent Processes
        - i. Task View Creation process
        - ii. Task Constraints Creation process
        - iii. Obtaining Information from the System Process
        - iv. Obtaining Information from a User Process
        - v. Constraint Satisfaction Process
        - vi. Negotiation Process

(b) Ontology-Based Automatic Code Generation for Domain-Dependent Processes

3. Prototype System Experiments (by September 2006)

References

[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.

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

[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 Third International Joint Conference on Autonomous Agents and Multi Agent Systems*, volume 2, pages 854–861, New York, July 2004.

[Din05] Y. Ding. *Web Semantic Annotation: Ontology Based Approach.* PhD thesis, Brigham Young University, 2005. to appear.

[DW03] I. Dickinson and M. Wooldridge. Practical Reasoning Agents for the Semantic Web. In *Proceedings of the Second International Conference on Autonomous Agents and Multiagent Systems (AAMAS-03)*, pages 306–318, Melbourne, Australia, July 2003.

[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.

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

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

[KG03] J. Kim and Y. Gil. Toward interactive composition of semantic web services. In *Proceedings of the Second International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida, 2003.

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

[MA02] K. Mark and B. Abraham. 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.

[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.

[MPM+04] 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 *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)*, San Diego, California, July 2004.

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

[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 Second International Workshop on Engineering Societies in the Agents World*, pages 175–194, Prague, Czech Republic, 2001.

[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.

[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.

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

[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.

[SSG04] M. Sheshagiri, N. M. Sadeh, and F. Gandon. Using Semantic Web Services for Context-Aware Mobile Applications. In *Proceedings of the Second International Conference on Mobile System, Applications, and Services*, Boston, Massachusetts, June 2004.

[W3C01] W3C. The DARPA Agent Markup Language (DAML). Website, 2001. `http://www.daml.org/`.

[W3C04] W3C. Web Ontology Language (OWL). Website, 2004. `http://www.w3.org/2004/OWL`.

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

Brigham Young University

Graduate Committee Approval

of a dissertation proposal submitted by

Muhammed Al-Muhammed

This dissertation proposal 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
Date	Charles D. Knutson
Date	Michael A. Goodrich
Date	Mark J. Clement
Date	Bryan S. Morse
Date	David W. Embley, Graduate Coordinator