Ontology-Based Constraint Recognition for Free-Form Service Requests

Muhammed J. Al-Muhammed*

Department of Computer Science
Brigham Young University
Provo, UT 84602, U.S.A.

David W. Embley*

Department of Computer Science
Brigham Young University
Provo, UT 84602, U.S.A.

Abstract

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

1 Introduction

Allowing users to specify service requests using fully free-form specifications is likely, if successful, to enhance their ability to obtain needed services. Consider, for example, the free-form request for an appointment with a dermatologist in Figure 1: "I want to see a dermatologist between the 5th and the 10th, at 1:00 PM or after. The dermatologist should be within 5 miles of my home and must accept my IHC insurance." To handle this request, a system must somehow recognize the constraints involved and transform them to a formal specification such as the one in Figure 2. If the system can recognize the constraints in Figure 1 and represent them in a predicate-calculus formalism like the one in Figure 2, then servicing this request becomes a matter of instantiating the free variables, the x_i's, such that the constraints are satisfied.

This paper proposes a particular way to recognize constraints from free-form service requests such as the request in Figure 1. Rather than use traditional natural language approaches that depend on syntax analysis (e.g. [8]) or statistical analysis (e.g. [11]), this paper introduces an ontological approach that depends on the long-standing notion of a semantic data model. In our ontology-based approach, a domain ontology encodes information such as applicable object sets, potential constraints over these object sets, and recognizers for instances of these object sets and constraints. The system recognizes the constraints in a service request by two-fold process. (1) It matches a free-form service request against a collection of ontologies that belong to different domains to find the ontology that matches best. (2) It then selects from the given and implied constraints in the matched ontology those that are relevant to the service request to generate the constraints.

The semantic data model of our approach characterizes the type of service requests our system is capable of handling. Specifically, our approach handles service requests whose objective is to instantiate an object set of interest in the domain ontology with a single value such that all applicable constraints are satisfied. The objective of the appointment request in Figure 1, for example, is to instantiate the variable x_0 with a value of type *Appointment* such that constraints on *Date*, *Time*, *Distance*, and *Insurance* are satisfied. This type of service covers a wide range of

* Supported in part by the National Science Foundation under grants 0083127 and 0414644.

Proceedings of the 32nd VLDB Conference, Seoul, Korea, 2006

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

Figure 1: A free-form appointment request.

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

Figure 2: The predicate-calculus formalism for the appointment request in Figure 1.

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

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

Our ontology-based approach also has the interesting advantage of being fully declarative. The algorithms to find the ontology that matches best, generate constraints, and produce a formal representation for the constraints are fixed. They work across domains with no need for recoding or reconfiguration. As a consequence, to produce formal representations for service requests belonging to a new domain, it is sufficient to specify the domain ontology—no coding is necessary.

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

We present our contributions as follows. Section 2 introduces domain knowledge. It describes the explicitly given domain knowledge encoded in terms of a domain ontology (Subsections 2.1 and 2.2) and the knowledge implied by a domain ontology (Subsection 2.3). Section 3 shows how to match a free-form service request to a domain ontology and obtain the ontology that matches best. Section 4 explains how to use the matched ontology to produce formal representations. It shows how to identify the parts of the best matching domain ontology that are relevant to a service request (Subsection 4.1) and how to identify any needed operations (Subsection 4.2). It then shows how to use the ontology, including both given and implied relationships sets and operations, to generate predicates (Subsection 4.3). In Section 5 we evaluate our approach. In Section 6 we compare our approach to other related work, and in Section 7 we give concluding remarks and directions for future work.

2 Domain Knowledge

In this section we describe the knowledge our system needs to generate a formal representation for a service request in terms of a domain ontology. First, the system requires explicit knowledge of basic concepts related to the service request. This explicit knowledge is encoded in terms of a domain ontology, which consists of two major components: (1) a semantic data model declaring sets of objects, sets of relationships, and constraints over the object and relationship sets (Subsection 2.1) and (2) instance semantics declaring recognizers for object set data values as well as operations applicable to these data values (Subsection 2.2). Sec-

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

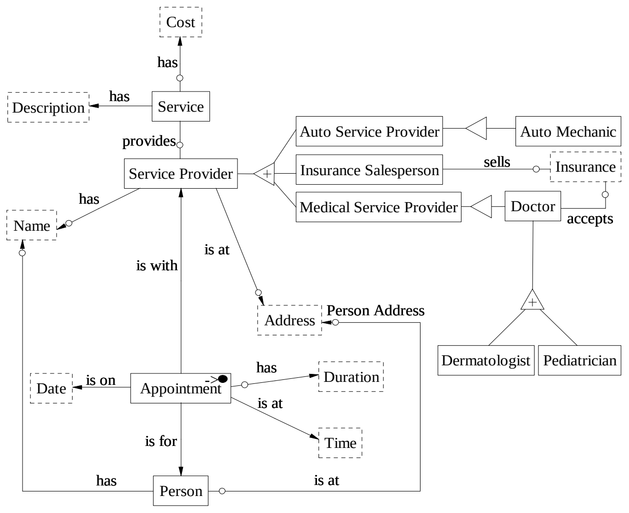


Figure 3: Semantic-data-model view of a domain ontology for appointments (partial).

ond, the system includes implicit knowledge—implied object sets, relationship sets, and constraints, which are based on knowledge explicitly given in the domain ontology (Subsection 2.3).

2.1 Semantic Data Model

A *semantic data model* specifies named sets of objects, which we call *object sets*, named sets of relationships among object sets, which we call *relationship sets*, and constraints over object and relationship sets. Figure 3 shows a small part of a semantic data model representation of a domain ontology for scheduling an appointment. The semantic data model consists of object-set concepts such as *Date*, *Time*, and *Service Provider* that can be used to schedule appointments with service providers such as doctors and auto mechanics. The semantic data model has two types of object sets, those that are lexical (enclosed in dashed rectangles) and those that are nonlexical concepts (enclosed in solid rectangles). An object set is *lexical* if its instances are indistinguishable from their representations. *Time* is an example of a lexical object set because its instances (e.g. "10:00 a.m." and "2:00 p.m.") represent themselves. An object set is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Dermatologist* is an example of a nonlexical object set because its instances are identifiers such as, say, "D_1", which represents a particular person in the real world who is a dermatologist. Each object set maps to a one-place predicate. For instance, the predicate $Date(x)$ is derived from the object set *Date* in Figure 3. The variable x in the predicate $Date(x)$ represents a place holder.

We designate the main object set in a semantic data model by marking it with "–> •" in the upper right corner. This notation,"–> •", denotes that when an ontology is used to satisfy a service request, the main object set becomes ("->") an object ("•"). We designate *Appointment* in Figure 3 as the main object set because this domain ontology is for making appointments. The system satisfies a service request by instantiating the main object set with a single value.

Figure 3 also shows relationship sets among object sets, represented by connecting lines, such as *Appointment is on Date.* The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *Service Provider has Name* is functional from *Service Provider* to *Name* (i.e. a service provider has only one name), and *Service Provider provides Service* is many-many (i.e. a service provider can provide many services and a service can be provided by many service providers). A small circle near the connection between an object set O and a relationship set R represents optional, so that an instance of O need not participate in a relationship in R. For example, the small circle on the *Appointment* side of the relationship set *Appointment has Duration* states that an instance of *Appointment* may or may not relate to an instance of *Duration* (i.e. there need not be a specified duration for an appointment). Each relationship set of arty n ($n \geq 2$) maps to an n-place predicate. For instance, $\mathit{Appointment}(x_0)$ *is with Service Provider*(x_1) is a two-place predicate derived from the relationship set *Appointment is with Service Provider* in Figure 3.

Constraints over unary predicates (object sets) and n-ary predicates (relationship sets) are closed predicate-calculus formulas. Referential integrity holds; thus, for example, for our ontology in Figure 3 we have $\forall x \forall y(\mathit{Doctor}(x)\ \mathit{accepts}\ \mathit{Insurance}(y) \Rightarrow \mathit{Doctor}(x) \wedge \mathit{Insurance}(y))$. Each functional constraint from an object set O to some other object set over a binary[2] relationship set R has the form $\forall x(O(x) \Rightarrow \exists^{\leq 1} y R(x, y))$. For instance, $\forall x(\mathit{Service\ Provider}(x) \Rightarrow \exists^{\leq 1}\mathrm{y}(\mathit{Service\ Provider}(x)\ \mathit{has}\ \mathit{Name}(y)))$ is the functional constraint for the relationship set from *Service Provider* to *Name.* Each constraint for a mandatory object set O for a binary relationship set R has the form $\forall x(O(x) \Rightarrow \exists^{\geq 1} y R(x, y))$. For instance, $\forall x(\mathit{Service\ Provider}(x) \Rightarrow \exists^{\geq 1}\mathrm{y}(\mathit{Service\ Provider}(x)\ \mathit{has}\ \mathit{Name}(y)))$ is the mandatory constraint for *Service Provider* in the *Service Provider has Name* relationship set.

A triangle in an ontology diagram (see Figure 3) denotes generalization/specialization. The generalization object set connects to the apex of the triangle, and specialization object sets connect to its base. For each generalization/specialization, we write the constraint $\forall x(S_1(x) \vee \ldots \vee S_n(x) \Rightarrow G(x))$, where G is the generalization object set and S_1, ...,

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

S_n are the specialization object sets. If the generalization/specialization has mutual-exclusion constraint (represented by the "+" in the triangle in Figure 3), we also write the constraints $\forall x(S_i(x) \Rightarrow \neg S_j(x))$ for $1 \leq i,\ j \leq n,\ i \neq j$. In Figure 3, for example, the constraint $\forall x(Dermatologist(x) \vee Pediatrician(x) \Rightarrow Doctor(x))$ states that dermatologists and pediatricians are specializations of doctors, and the constraints $\forall x(Dermatologist(x) \Rightarrow \neg Pediatrician(x))$ and $\forall x(Pediatrician(x) \Rightarrow \neg Dermatologist(x))$ state that dermatologists and pediatricians are mutually exclusive.

Every connection between an object set and a relationship set is a role. A role designates the set of objects of an object set that participate in a relationship set. If we wish to name the role, we place the role name near the connection between its object set and its relationship set. For instance, the role *Person Address* in Figure 3 appears near the connection between the object set *Address* and the relationship set *Person is at Address*. A named role is a specialization of the object set to which it connects. *Person Address* thus represents the subset of addresses that associate with persons.

2.2 Data Frames

Each object set (including each named role) in a domain ontology has an associated data frame [6], which describes instances for the object set. Data frames capture the information about object-set instances in terms of their external and internal representation, their context keywords or phrases that may indicate their presence, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the object set along with context keywords or phrases that indicate the applicability of an operation and operands in an operation. Figure 4 shows sample (partial) data frames for *Time*, *Date*, *Address*, *Person Address*, *Dermatologist*, *Appointment*, *Insurance*, and *Distance*.

As Figure 4 shows, we use regular expressions to capture external textual representations. The *Time* data frame, for example, captures instances that end with "AM" or "PM" (e.g. "2:00 PM" and "9:30 a.m."). A data frame's context keywords/phrases are also regular expressions. For example, the *Distance* data frame in Figure 4 includes context keywords such as "miles" or "kilometers". In the context of one of these keywords, if a number appears, it is likely that this number is a distance. A nonlexical object set such as *Dermatologist* has only context keywords or phrases. Figure 4 shows that the *Dermatologist* data frame includes a regular expression, which includes keywords and phrases that could indicate the presence of an instance of a dermatologist.

The operations in data frames manipulate object-set instances. For example, the operation *Distance-*

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

Figure 4: Some sample data frames. (The "..." in the textual-representation part of the *Time* data frame indicates that there are other representations of *Time* such as military time. In general the presence of ellipses show omissions needed to complete the data frames.)

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

The context keywords/phrases for an operation indicate the possible applicability of the operation. The context keywords/phrases are regular expressions that include keywords or phrases and possibly expandable expressions represented by operand names enclosed in braces. The system expands these expressions by finding the types of their operands and substituting the textual representations in the data frames of the types for these expressions. The advantage of marking these expandable expressions with operands is that when context keywords/phrases for an operation match substrings in a service request, the system can record which values are for which operands. For instance, the context keywords/phrases associated with the operation *DateBetween* in Figure 4 has the regular expression *between\s+{x2}\s+and\s+{x3}*, which includes the expandable expressions {*x2*} and {*x3*}. As Figure 4 shows, the operands of these two expressions are of type *Date*. When this regular expression matches a substring in a request such as "make the appointment between the 10th and the 15th," the system can record that the first date value ("the 10th") is for *x2* and the second date value ("the 15th") is for *x3*.

2.3 Implied Knowledge

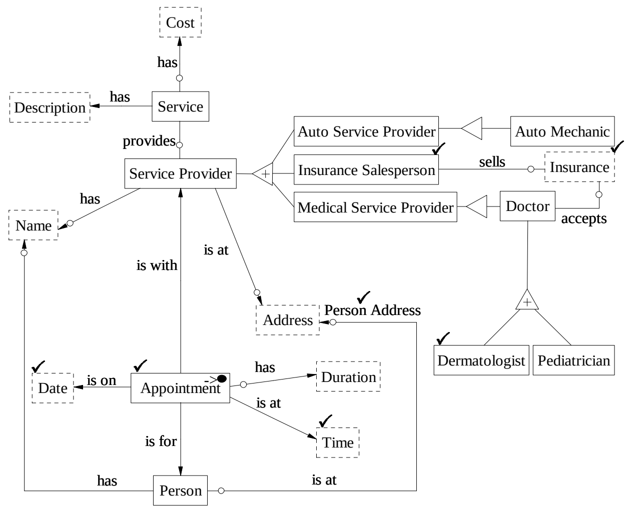
Object sets, relationship sets, and constraints that can be computed from the domain ontology constitute the implied knowledge. For example, the system can derive a relationship set between *Appointment* and *Name* from the given relationship sets *Appointment is with Service Provider* and *Service Provider has Name*. The system can also determine that *Name* mandatorily depends on *Appointment* from the given constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *is with* $Service\ Provider(y)))$ and $\forall$x($Service\ Provider(x) \Rightarrow \exists^{\geq 1} z(Service\ Provider(x)$ *has* $Name(z)))$, where the former states that *Service Provider* is mandatory for *Appointment* and the latter states that *Name* is mandatory for *Service Provider*. Further, the system can determine that *Name* functionally depends on *Appointment* from the given constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\leq 1} y(Appointment(x)$ *is with* $Service\ Provider(y)))$ and $\forall$x($Service\ Provider(x) \Rightarrow \exists^{\leq 1} z(Service\ Provider(x)$ *has* $Name(z)))$. As additional examples, there are many implied generalization/specialization constraints derivable from the constraints in Figure 3. For instance, the system can derive the implied constraint $\forall x(Dermatologist(x) \Rightarrow Service\ Provider(x))$ by transitivity from the following given constraints: $\forall x(Dermatologist(x) \Rightarrow Doctor(x))$, $\forall x(Doctor(x) \Rightarrow Medical\ Service\ Provider(x))$, and $\forall x(Medical\ Service\ Provider(x) \Rightarrow Service\ Provider(x))$.

The connections between operands of an operation in a data frame and the relationship sets of a semantic data model may be implicit. Consider, for example, the operation *DistanceBetweenAddresses* in the *Address* data frame, which computes the distance between two addresses. It is not explicitly given in the domain ontology, Figures 3 and 4, whether this operation computes the distance given two service-provider addresses, two person addresses, or a service-provider address and a person address. The system, however, can reason that for an appointment, if there is a constraint on distance, then it must be between a service-provider address and a person address. The system reasons as follows. The constraints $\forall$x($Appointment(x) \Rightarrow \exists^{\leq 1} y(Appointment(x)$ *is with* $Service\ Provider(y)))$ and $\forall$x($Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)$ *is with* $Service\ Provider(y)))$ allow the system to infer the implicit constraint $\forall$x($Appointment(x) \Rightarrow \exists^{1} y(Appointment(x)$ *is with* $Service\ Provider(y)))$, which states that for any appointment there exists exactly one service provider. The system can derive from the constraints $\forall$x($Service\ Provider(x) \Rightarrow \exists^{\leq 1} y(Service\ Provider(x)$ *is at* $Address(y)))$ and $\forall$x($Service\ Provider(x) \Rightarrow \exists^{\geq 1} y(Service\ Provider(x)$ *is at* $Address(y)))$ the constraint $\forall$x($Service\ Provider(x) \Rightarrow \exists^{1} y(Service\ Provider(x)$ *is at* $Address(y)))$, which states that there is exactly one address for a service provider. Given these two derived constraints, since there is at most one service-provider address for an appointment, the system can certainly exclude the possibility that *DistanceBetweenAddresses* computes distances between two addresses of service providers when it makes an appointment. Likewise, the system can exclude the possibility that *DistanceBetweenAddresses* computes the distance between addresses of persons. Thus, the system can infer that the two operands *a1* and *a2* of the operation *DistanceBetweenAddresses* must obtain their values from addresses in the relationship sets *Service Provider is at Address* and *Person is at Address*.

3 Domain Ontology Recognition

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

For each domain ontology, the system applies all the recognizers in the data frames of every object set in the domain ontology to the service request. It marks every object set whose recognizers match a substring



(a) Matched (✓) object sets in the semantic data model in Figure 3.

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

(b) Matched (✓) object sets and operations in the data frames in Figure 4.

Figure 5: Output of the recognition process—the marked-up domain ontology.

in the service request and every operation whose applicability recognizers match a substring in the service request. The result of the matching is a set of marked-up domain ontologies.[3]

When the recognition process executes for the domain ontology in Figures 3 and 4 and the appointment request in Figure 1, it produces as output the marked-up ontology in Figure 5. Figure 5(a) shows the matched (✓) object sets in the semantic data model in Figure 3, and Figure 5(b) shows the matched (✓) operations and the additional object sets from Figure 4. The recognizers in the data frame in Figure 4 for *Dermatologist* recognize the context keyword "dermatologist" in the service request in Figure 1, and therefore *Dermatologist* is marked (✓). Likewise, as Figure 4 makes evident, recognizers in the *Date* data frame recognize "between the 5th and the 10th"; in the *Time* data frame recognize "at 1:00 PM or after"; in the *Distance* data frame recognize "within 5 miles"; in the *Appointment* data frame recognize "want to see a"; in the *Insurance* data frame recognize the context keyword "insurance" and the constant value "IHC"; and in the *Person Address* data frame recognize the context phrase "my home". Therefore these object sets are marked. Although not included in Figure 4, we assume that the recognizer for context keywords in the *Insurance Salesperson* data frame would recognize "insurance". Therefore *Insurance Salesperson* is marked.

Given the data frames in Figure 4, additional matched operations and object sets may have been expected. For example, the context keywords/phrases for the operation *TimeEqual* in the *Time* data frame would match "at 1:00 PM" and the *Cost* data frame may have recognizers that match "within 5". We eliminate these matches, however, based on a subsumption heuristic. This heuristic uses the positions of the matched substrings in a service request to determine whether a matched substring subsumes another matched substring. The system does not mark an object set or an operation if its matched substring is properly subsumed by another matched substring. We assume that there is only one match for a string and that the subsuming substring is a better match. Thus, although the context keywords/phrases for the operation *TimeEqual* would recognize "at 1:00 PM", the system would not mark the operation *TimeEqual* because it matches with only the substring "at 1:00 PM", which is subsumed by the substring "at 1:00 PM or after", matched by the operation *TimeAtOrAfter*.

To choose the marked-up domain ontology that best matches the service request, the system ranks them. In our approach, the system grants rank values for each marked-up domain ontology based on the marked object sets. The marked main object set of the marked-up ontology has the highest weight for obvious reasons. Marked mandatory object sets contribute with the next highest weight because they represent the necessary requirements to establish the main concept. Marked optional object sets contribute with lower weights because they are not necessary for establishing the main concept.[4] To continue with our running example, we assume that the system selects our appointment ontology as the best matched ontology for the service request in Figure 1.

4 Formal Representation Generation

A formal representation of a free-form service request is a predicate-calculus formula. The system generates the predicates of a formal representation for a free-form service request only from the given and implied knowledge. It cannot generate predicates for con-

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

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

straints in a service request that refer to object sets, relationship sets, constraints, or operations beyond this knowledge. For instance, if the appointment ontology designer leaves out the *Insurance* object set, any constraint in a service request about insurance such as "must accept my IHC insurance" will be ignored.

The input to the formal representation generation process is a marked-up ontology. The output is a predicate-calculus formula. Not all knowledge in a marked-up ontology is relevant. Irrelevant knowledge should be pruned away. Otherwise, the system will generate an overconstrained predicate-calculus formula. The system, therefore, should find the sub-ontology including object sets, relationship sets, and operations that are relevant to the service request. We call this sub-ontology the *service request view.* In Subsections 4.1 and 4.2, we explain how the system generates the components of the service request view. In Subsection 4.3, we show how the system uses the service request view to generate the formal representation.

4.1 Relevant Object Set and Relationship Set Identification

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

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

All other object sets and relationship sets are pruned away.

The system obtains the object sets that mandatorily depend on the main object set from the given and implied relationship sets that involve the main object set and from the given and implied constraints for these relationship sets. In our running example, the given relationship set *Appointment is with Service Provider* shows that *Service Provider* is related to *Appointment*, and the given constraint $\forall x(Appointment(x) \Rightarrow \exists^{\geq 1} y(Appointment(x)\ \textit{is with Service Provider}(y)))$ shows that *Service Provider* is mandatory. Further, as we discussed in Subsection 2.3, there is an implied relationship set between *Appointment* and service-provider *Name*, and an implied constraint for this implied relationship set that makes *Name* mandatorily depend on *Appointment.* Likewise, the system can infer that *Date*, *Time*, *Person*, service-provider *Address*, and person *Name* are all mandatory.

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

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

Referring to our example, the implied constraint $\forall x(Appointment(x) \Rightarrow \exists^{1} y(Appointment(x)\ \textit{is with Service Provider}(y)))$ requires exactly one instance value in the is-a hierarchy to be associated with an appointment. Further, the implied mutual exclusion constraint between the marked specializations, *Dermatologist* and *Insurance Salesperson*, allows the system to infer that the single instance must belong to only one of these marked specializations. To determine which one of the marked specializations, the system ranks them. Each marked specialization receives a rank value according to: (1) the number of strings in a service request matched by the data frame recognizers of the specialization, (2) the number of the marked object sets directly related to the specialization, and (3) the distance between the locations of the strings in the service request matched by the specialization and the locations of the strings in the service request matched by the main object set. For the first criterion for our example, *Dermatologist* matches with more strings (two occurrences of "dermatologist") than does *Insurance Salesperson* (matches with the single string "insurance"). For the second criterion, both the marked specializations relate to one marked object set, *Insurance.* (Observe that since *Dermatologist* in Figure 5 is a *Doctor*, it inherits all the relationship sets in which *Doctor* is involved, and thus *Dermatologist* is connected to *Insurance.*) For the third criterion, the

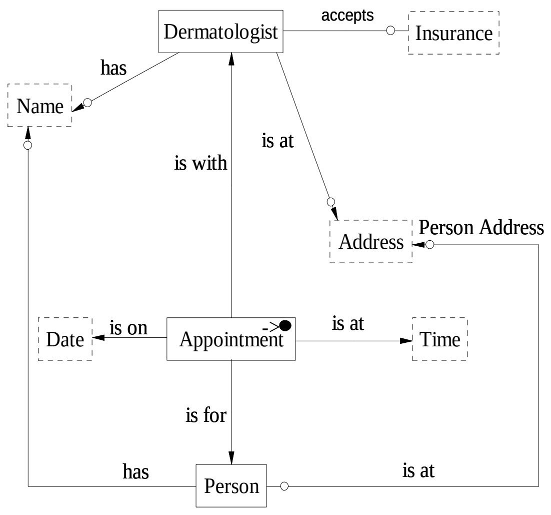


Figure 6: The relevant object sets and relationship sets for the appointment in Figure 1.

location of the first occurrence of "dermatologist" in the service request is closer to the location of the string "want to see a", matched by the main object set than is the location of the string "insurance", matched by *Insurance Salesperson*. Thus, the system keeps only the marked specialization *Dermatologist* in the is-a hierarchy. The system removes all the other specializations and collapses the is-a hierarchy. Figure 6 shows the resulting relevant object sets and relationship sets for the appointment request in Figure 1.

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

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

Finally, if there is no marked specialization in an is-a hierarchy but an element in the is-a hierarchy is mandatory, we keep the root of the is-a hierarchy and prune away all its specializations. We also keep all relationship sets that lead to marked object sets, if any, and optionally connect them to the root. If an element in the is-a hierarchy is not mandatory, we discard the entire hierarchy and all connected relationship sets.

4.2 Relevant Operation Identification

The operations relevant to a service request are the Boolean operations whose applicability recognizers match strings in the service request and operations on which operands of these Boolean operations may depend for values. For our appointment example, the Boolean operations in Figure 5(b) are the relevant Boolean operations.

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

The operand *d1* of the operation *DistanceLessThanOrEqual* is of type *Distance*, which is not involved in any given relevant object set in Figure 6. The system, therefore, must find an operation that depends on the relevant object sets and computes values for this input parameter. If the system cannot find such an operation, the operation is ignored. The operand *d1* can potentially be computed by the operation *DistanceBetweenAddresses*, which depends on the relevant object set *Address*. The system, therefore, binds *d1* to the operation *DistanceBetweenAddresses*. As we discussed in Subsection 2.3, the system can infer from constraints on the relationship sets on which the operation *DistanceBetweenAddresses* depends that the address values *a1* and *a2* come respectively from the *Address* object sets in *Dermatologist is at Address* and *Person is at Address*.

Figure 7 shows the relevant operations for the appointment request in Figure 1. Each input parameter is either instantiated with a value or bound to an operation that yields a value or to a (possibly unknown but nevertheless specific) value in an object set.

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

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

Figure 7: The relevant operations for the appointment request in Figure 1.

4.3 Predicate-Calculus Formula Generation

The system conjoins the predicates generated as described in Subsection 4.1 and Subsection 4.2 to generate the formal representation for a free-form service request. For our running example, the system conjoins the predicates for each relationship set in Figure 6 with the formulas in Figure 7 to produce the formal representation for the service request in Figure 1. After renaming variables, we have exactly the predicate-calculus formula in Figure 2.

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

5 Performance Analysis

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

We asked subjects from Brigham Young University to make free-form, natural-language-like service requests belonging to these domains using their own words. The subjects ranged from savvy computer users and online shoppers to users with limited computer skills. We provided the subjects with no information about the structure of the underlying domain ontologies or the recognizers or operations in the data frames. We asked the subjects to make service requests with only conjunctive constraints and positive literals—the type of constraints that our system is capable of recognizing. To avoid technical terms (e.g. "conjunctive" and "positive literals"), we provided users with illustrative examples to use for formulating their requests. Figure 8 shows the instructions we provided for subjects to write appointment requests for medical doctors.

Table 1 shows the number of service requests and the number of included predicates and values in these requests for each of the three domains. We received a total of 31 requests, which included a total of 548 constraints and a total of 170 constant values. We reviewed all service requests we received, manually extracted the included constraints and constant values in each service request, assigned each constant value to its respective operand, manually generated a formal representation for each request, and stored it in a format similar to the way the system records results. We then fed each service request to the system, which created the formal representation for the request, compared this formal representation against the manually generated request, and automatically computed the recall and precision.

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

Table 1: Number of service requests, predicates, and arguments.

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

Table 2: Recall and precision.

Table 2 shows the performance of the system in finding predicates and constant values in terms of the recall and precision for each one of the three domains along with the overall recall and precision. The system performed surprisingly well.

As Table 2 shows, the recall for predicates was high for all three domains. The recall numbers for constant values was a little lower, but nevertheless quite high. The system did not recognize these variations of date for appointments: "any Monday of this month" and "most days of the week", these features for cars: "power doors and windows" and "v6" (the engine size), and these features for apartments: "a nook", "dryer hookups", and "extra storage". Therefore, the recall for constant values in the appointments, cars, and apartments rental domains dropped off from 100%. Further, missing these constant values caused the sys-

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

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

Figure 8: Instructions for subjects for the appointments domain.

tem to miss the constraints over these values causing the recall for predicates in the appointments, cars, and apartments rental to be lower than they otherwise would have been.

We can fix these recall problems by providing better recognizers that can better cover the space of possible values for the object sets. We recognize, however, that this may not always be easy. In [7], for example, the authors describe an automaton with 1,223 states and 21,006 arcs to correctly recognize strings representing a date. Although not always easy to obtain full coverage, it is possible, with reasonable effort, to obtain near full coverage. The advantage gained may very well be worth the effort.

The precision was near 100% for both predicates and arguments. When the system selects the right ontology for a service request, the system almost cannot obtain irrelevant constraints because our ontology is narrowly focussed on the service. The only way the system can produce an irrelevant predicate is when the system incorrectly marks an operation or an object set based on the appearance of some constant value or a context keyword/phrase and the ontological knowledge is not enough to enable the system to prune it away. Consider, for example, this constraint "I want a Toyota with a cheap price, 2000 would be great ...", which was taken from one of the requests and for which our system incorrectly generated the constraint, *PriceEqual*(*p1*: *Price*, "*2000*"). The appearance of the contextual keyword "price" close to the number 2000 makes our system recognize 2000 as a price value rather than a year value. The type of ambiguity in this constraint is not easy to handle (perhaps not even easy for humans) because it is not so clear whether the subject meant the price to be 2000 or the year to be 2000.[6]

6 Related Work

Some researchers in the natural language processing community work on systems that transform natural language to a formal specification such as predicate calculus, as we do here. These systems, called logic form generation or transformation systems [5, 4, 10], use parsers to parse a syntactically correct sentence and identify its constituents such as nouns, verbs, and adjectives. Each constituent defines a predicate. The syntactic structure of a parsed sentence defines the relationships among the constituents, which are captured through shared arguments among the predicates. Based on reported results in [5, 4, 10] and in [13], which compares the performance of three other approaches, these systems are able to achieve a recall within the interval [78%, 90%] and a precision within [81%, 87%] at the predicate level, and a recall within [65%, 77%] and a precision within [72%, 77%] at the argument level.

For many years, researchers in the database community have also worked on generating constraints from natural language queries. Older approaches, surveyed in [3], parse their input using either syntactic parsers or sematic parsers to produce parse trees. The main difference between syntactic and semantic parsing is that the grammar categories for semantic parsing directly correspond to database elements such as table names and attributes names rather than syntactic concepts such as noun phrases. In both cases, the parse tree is used to generate a database query with the help of mapping rules that specify how each element in the parse tree maps to an element in the database query.

Newer approaches build on these older approaches by introducing additional techniques that improve results. The approach proposed in [8] uses a dependency parser to determine how the words in a sentence depend on each other. A query is parsed to create the parse tree, which captures the dependencies between the query tokens, and then each node in the parse tree is classified according to XQuery components (e.g. a **return** clause). If the system cannot classify some node in the parse tree, it asks a user to rephrase the query. The **where** clause constraints are created based on patterns that appear in a dependency tree. For instance, the appearance of the pattern "$\langle$*variable*$\rangle$ + $\langle$*constant*$\rangle$"—i.e. a variable followed by a constant value—maps to the constraint "*variable* = *constant*" in **where** clause. Experiments reported in [8] show that this approach is able to achieve 95.1% precision and 97.6% recall. These results are for queries that are

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

correctly parsed and whose resulting parse-tree nodes are correctly classified. With respect to all queries, however, the reported recall and precision were respectively 90.1% and 83%.

The PRECISE system, proposed first in [12] and later enhanced with a semantic model to correct some parser errors [11], uses a statistical parser and lexicons, consisting of names of relations, attributes, and values of the attributes as well as *wh*-designators (what, which, where, who, and when designators) attached to the attributes. A natural language query is parsed with respect to the lexicon that matches each main word in the query to one or more database elements (table name, attribute name, value, and *wh*-designator). The system then constructs attribute-value mappings, which are validated by the relationships produced by the parser. The **where** clause in the generated SQL query is a conjunction of attributes with mapped values along with join conditions that reflect the join paths among tables. The reported results for experiments on three domains show that PRECISE is able to achieve 100% precision and a recall within the interval [∼75%, ∼93%] for "semantically tractable queries." Like our proposed system, PRECISE extensively exploits the schema of the database. Since neither system generates constraints beyond its schema, precision tends to be high. Improper constraints can only be generated by false positives within the purview of the database schema.

The approach described in [9] is quite close to our approach. It uses a semantic model of an underlying database, which is a graph that consists of nodes representing database relations and attributes and edges representing connections among relations. Keywords or keyword phrases are attached as labels to nodes and operators (standard operators such as "<", ">", or "="). Operators are attached to the attributes if these operators apply to values of the attributes. The system matches a natural language query to the keywords attached to semantic-model elements and uses a statistical approach (n-grams) to disambiguate matches. As with our approach, this approach does not seem to require syntactically correct queries. No empirical results are reported in [9], and therefore it is hard to assess its performance.

All these approaches, except [9], expect syntactically correct sentences. We do not. Further, generally speaking, our approach performed with better recall and precision. We believe that our approach has two important novelties that contribute to its performance. First, the semantic data model captures the relationships among objects and constraints over these objects in the domain, and therefore we avoid precision errors introduced when parsers try to determine relationships among constituent parts of the input. Further, as an added benefit of our particular service-oriented paradigm, the semantic data model allows the system to derive relationships that are necessary for satisfying a service request even though the service request does not specify them at all. Second, the semantics associated with the object sets through data frames allow our approach to capture constraints through operations in these data frames. This means that once a constraint in a service request is recognized by the applicability recognizers of an operation, then this constraint is correctly formalized by means of this operation. Our approach, however, does require designers of service-request ontologies to produce a proper semantic data model that appropriately covers the scope of the service and to produce recognizers in data frames that correctly recognize appropriate value and keyword instances. We believe, however, that because of the narrow focus of a particular service, this task is as easy (and possibly easier) than producing required lexicons, parsers, and similar components for alternative approaches.

7 Conclusions and Future Work

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

As future work, we plan to integrate the work reported here with other work we have done [1] to produce the overall system we have envisioned [2]. The system we have envisioned transforms a service request into a predicate-calculus formula as explained here. It uses the predicate-calculus formula to create a query to a databases associated with the domain ontology from which the formula was generated to instantiate as many variables of the formula as possible. The system then discovers the variables in the predicate-calculus formula that are yet to be instantiated and interacts with a user to obtain values for these variables. When all the variables are instantiated, the system checks whether the constraints of the formula are satisfied. Constraint satisfaction can yield too many solutions or no solution. As reported in [1], the system controls the potential overload on users when there are too many solutions by returning the best-m solutions rather than all of them or offers users the best-m near solutions when there is no solution. When a user chooses one of the suggested solutions or near solutions, the system completes the service request by inserting an object of interest (e.g. an appointment) in the main object set of the domain ontology and by inserting other mandatory and optional objects and relationships and thus satisfies the service request.

References

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

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

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

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

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

[6] 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.

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

[8] Y. Li, H. Yang, and H.V. Jagadish. Constructing a Generic Natural Language Interface for an XML Database. In *Proceedings of International Conference on Extending Database Technology (EDBT 2006)*, Munich, Germany, March 2006. (to appear).

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

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

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

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

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