Accessing Information and Services on the DAML-Enabled Web*

Grit Denker, Jerry R. Hobbs, David Martin, Srini Narayanan, Richard Waldinger
SRI International
Menlo Park, California, USA
denker@csl.sri.com, {hobbs, martin, narayana, waldinge}@ai.sri.com

ABSTRACT

The DARPA Agent Markup Language (DAML) program aims to allow one to mark up web pages to indicate the meaning of their content; it is intended that the results delivered by a DAML-enabled browser will more closely match the intentions of the user than is possible with today's syntactically oriented search engines.

In this paper we present our vision of a DAML-enabled search architecture. We present a set of queries of increasing complexity that should be answered efficiently in a Semantic Web. We describe several scenarios illustrating how queries are processed, identifying the main software components necessary to facilitate the search. We examine the issue of inference in search, and we address how to characterize procedures and services in DAML, enabling a DAML query language to find web sites with specified capabilities.

Key Words: Semantic Web, DAML, inference, Web services, process modeling.

1. INTRODUCTION

Querying the Web today can be a frustrating activity because the results delivered by syntactically oriented search engines often do not match the intentions of the user. This problem is caused by the Web's lack of semantic structures that could be exploited during the search process.

DARPA's DAML program (`http://www.daml.org`) aims at overcoming this problem. The DARPA Agent Markup Language is intended to allow annotating web pages to indicate the meaning of their content. Thus, search engines are supported in their decisions about the appropriateness of a web page as an answer to a query and are able to extract the most appropriate information.

DAML allows specifying ontologies and annotating Web content with respect to these ontologies. Together with a yet-to-be-defined query mechanism, these DAML annotations are intended to ease and improve searches on the Internet. In this paper we present our vision of a DAML-enabled search architecture by outlining different software components and their functionality, the use of inference during search, and the declarative representation of the capabilities of services and procedures available on the Web.

1.1 Some DAML Queries

To illustrate what is desired on the Semantic Web, we present a set of queries of increasing complexity expressed in natural language, which should be answered efficiently in our framework. The queries, which are concerned with the general area of searching for and obtaining publications via the Internet, illustrate roughly increasing complexity in the requirements for the search process.

1. Find information about a researcher with name "James Hendler". (This can very nearly be accomplished with keyword searches, although for very common names the search engine would return too many hits by not filtering on "researcher".)

2. Find a reference to a paper about SHOE that is co-authored by Hendler. (SHOE was named in part because Web searches for it would also turn up sites on shoes. DAML will restrict the search to references to publications about SHOE.)

3. Find a reference to the most recent paper on SHOE with James Hendler as a co-author. (For this query, the search engine must find all references possible within search constraints, find their dates, and pick the most recent. It must know about the structure of references and about date arithmetic.)

4. Request the Stanford Library to hold a copy of Daniel Dennett's book "Elbow Room" for me. (Here the search engine must deal with service and procedural aspects of web pages. A password is required, and logging in is necessary, as well as searching for the book on the library's web site and executing the process for requesting a hold.)

5. Buy a copy of Daniel Dennett's "Elbow Room" for me from amazon.com. (This again requires executing a procedure, but in addition involves security concerns in sending in credit card information.)

These examples motivate our concerns in the long term. In this paper we will illustrate what is required to handle them.

In Section 2 we describe several scenarios illustrating how queries are processed. Along the way we identify the main software components necessary to facilitate the search, and we summarize the software architecture for DAML-enabled search and outline the main functionalities of its components. In Section 3 we examine the issue of inference in search. Although DAML is still in the process of being defined and its query language has not yet been specified, we have implemented the inference scenarios in first-order

*Supported by the Defense Advanced Research Projects Agency through the Air Force Research Laboratory under Contract F30602-00-C-0168.

Semantic Web Workshop 2001 Hongkong, China

logic on an automatic theorem prover. As DAML develops, we are defining the mapping between it and this notation. Finally in Section 4 we address the issue of characterizing procedures and services in DAML in terms of an existing model of complex processes. This will enable a DAML query language to find web sites with specific capabilities.

2. A DAML-ENABLED ARCHITECTURE FOR QUERYING THE WEB

2.1 Query Processing: Example Scenarios

We present the flow of data and control for DAML query processing, with the help of the first example query of the previous section.

DAML ontologies for publications, researchers, and topics have been built. Here we present only those parts necessary for processing our example queries. More details can be found under

`www.ai.sri.com/daml/ontologies/`

in `Publication.daml` and `Researcher.daml`.

In `Publication.daml` a class `Publication-ref` has two properties among others: `author`, of type `Researcher`, and `topic`, of type `Topic`. The class `Researcher` is declared in `Researcher.daml` along with its properties `lastName` and `firstName` of type `String`. The property `name` as well as `subTopics` and `relatedTopic` are properties of the class `Topic`. Fig. 1 summarizes the classes and their relationships via property declarations, where List is taken to be a way to form lists.

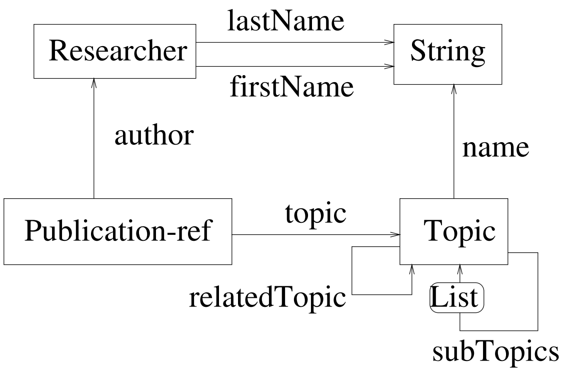

Figure 1: Ontologies

We use an ad hoc notation for DAML queries. Our purpose is not to suggest a particular query language, but rather to identify concepts necessary to support DAML-enabled search.

The first query, "Find information about a researcher with name James Hendler", may be formalized as follows:

```
xmlns: SRI = ``http://www.ai.sri.com/daml/
                      ontologies/Researcher#''
FIND <SRI:Researcher>
SUCH-THAT <SRI:Researcher:lastName>
            Hendler
          </SRI:Researcher:lastName>
          <SRI:Researcher:firstName>
            James
          </SRI:Researcher:firstName>
END
```

The query specifies the ontology (or ontologies) it uses. In our example we search for information about researchers. Therefore, we cite the SRI `Researcher` ontology that has `lastName` and `firstName` as properties. `FIND <SRI:Researcher>` means that we expect the query to return objects of class `Researcher` (including all properties that are defined for this class). We restrict our result to those researcher objects that have "James Hendler" as name.

After parsing the query, it would be handed to a search engine DAML-Q (for "DAML Query engine") that would be capable of processing queries written in the DAML query language. During its search it makes use of the ontology information that is part of the query.

```
xmlns:SRIRes = "http://www.ai.sri.com/daml/
                       ontologies/Researcher#">
  <SRIRes:Researcher>
    <firstName>James</firstName>
    <lastName>Hendler</lastName>
    <title>Dr.</title>
  </SRIRes:Researcher> ....
```

Figure 2: Scenario 1

Assume a DAML-annotated web page as illustrated in Fig. 2. DAML-Q selects this web page for further inspection because the ontology cited in the query matches one of the ontologies in the list of namespaces of the web page. From the xmlns declaration DAML-Q obtains the namespace identifier (here "`SRIRes`") and sequentially searches the content of this web page for a tag `<SRIRes:Researche` according to the query. Whenever it finds an object of this class, DAML-Q will search for tags

```
<firstName>James</firstName>
```

and

```
<lastName>Hendler</lastName>
```

and select objects that satisfy these criteria. Note that DAML-Q needs to be capable of recognizing the various equivalent notations that can be used instead, such as

```
<lastName ID="Hendler"/>
```

or

```
<SRIRes:Researcher:lastName>
  Hendler
</SRIRes:Researcher:lastName>
```

among others. The query result will contain the researcher object (with all its properties) defined in the web page illustrated in Fig. 2.

A slightly more complex situation is one in which a web page does not refer to the ontology stated in the query but is related to that ontology by some means. For example, assume a web page that declares the ontology `www.ai.sri.com/˜hobbs/MyResearcher.daml`. A search through this web page shows that this ontology is defined in terms of the SRI Researcher ontology. More precisely, a subclass `MyResearcher` of `Researcher` is declared in

`www.ai.sri.com/˜hobbs/MyResearcher.daml`

that adds properties to the `Researcher` class of the original SRI ontology. Using this information, DAML-Q can infer that a web page like the one in Fig. 3 is a match for the query. DAML-Q extracts from the ontology `www.ai.sri.com/∼hobbs/MyResearcher.daml` and the given `<subClassOf>` declaration,

```
xmlns:  SRI = "http://www.ai.sri.com/daml/
                       ontologies/Researcher#">
<Class ID="MyResearcher">
   <subClassOf resource="SRI:Researcher"/>
   </Class> ....
----------------
xmlns:  hobbs="http://www.ai.sri.com/ hobbs/
                       MyResearcher#">
  <hobbs:MyResearcher>
   <firstName>James</firstName>
   <lastName>Hendler</lastName>
  </hobbs:MyResearcher> ....
```

Figure 3: Scenario 2

that the class tag to be searched for is `MyResearcher` (the property tag remains `lastName`).

Instead of placing the burden of recursively searching through web pages that have direct or indirect references to ontologies contained in the query on DAML-Q, it seems much more efficient to implement a mapping service for ontologies, which we may call DAML-M. DAML-M would be a service that for a given ontology and a set of classes and properties returns mappings to other ontologies and their classes and properties that are declared to be equivalent. For instance, the Knowledge Systems Lab of Stanford University has declared a class `PERSON` in its ontology `www.ksl.stanford.edu/projects/DAML/ksl-daml-desc.daml`. The property `HAS-FULL-NAME` of type `STRING` is declared for this class. Contacted by DAML-Q with the query at hand, DAML-M would return this information indicating that web pages that refer to the KSL ontologies and that have content marked as an object of class `PERSON` with property `HAS-FULL-NAME` are also possible answers to the query. Fig. 4 illustrates this situation.

```
<rdfs:Class ID="PERSON">
  <rdfs:subClassOf rdfs:resource="#ORGANISM"/>
</rdfs:Class>
<rdfs:Property ID="HAS-FULL-NAME">
  <rdfs:domain rdfs:resource="#PERSON"/>
  <rdfs:range rdfs:resource="#STRING"/>
</rdfs:Property>
----------------
http://www.ai.sri.com/daml/ontologies/Researcher#
  --> http://www.ksl.stanford.edu/projects/
                       DAML/ksl-daml-desc#">
Researcher --> PERSON
firstName lastName --> HAS-FULL-NAME
----------------
xmlns:  KSL="http://www.ksl.stanford.edu/
                       projects/DAML/
                       ksl-daml-desc#">
  <KSL:PERSON>
   <HAS-FULL-NAME>James Hendler</HAS-FULL-NAME>
  </KSL:PERSON> ...
```

Figure 4: Scenario 3

It is obvious that the task of mapping ontologies, classes and properties is nontrivial. Already in our simple example one has to be able to map two properties into one. We envision that DAML would suggest a "DAML mapping" ontology that defines the standard interface for ontology mapping. This ontology has to provide for complex mapping operations such as mapping different kinds of combinations of properties or classes. Given such a "standardized ontology mapping interface", a software module like DAML-M can collect mapping information from different sites that advertise such mappings using the "DAML mapping" ontology. Here trust issues are critical: A search engine like DAML-Q might prefer to get mapping-related information only from specific sites since they have proven reliable with respect to the mapping assertions. Trust and security issues are orthogonal to other parts of the query. The DAML query language also needs to be able to support the specification of such requirements.

Further complexity may be introduced by less-defined DAML queries. For instance, the user may be aware that "James Hendler" is often also referred to as "Jim Hendler". Thus, the formalization of the query may leave the specifics of the first name open, or suggest alternatives that are considered equivalent. For instance, the query "Find information about a researcher with name James Hendler" may be formalized as

```
xmlns: SRI = "http://www.ai.sri.com/daml/
                     ontologies/Researcher#}"
FIND <SRI:Researcher>
SUCH-THAT <SRI:Researcher:lastName>Hendler
          </SRI:Researcher:lastName>
          OR(<SRI:Researcher:firstName>
               James
             </SRI:Researcher:firstName>,
             <SRI:Researcher:firstName>
               Jim
             </SRI:Researcher:firstName>)
END
```

It is also possible that one does not know the exact first name, but believes that it is "James" or something similar. Assume that there exists an Internet service that for a given first name delivers names that are commonly used instead of the given first name or that are considered equivalent or a nickname or an abbreviation. For instance, there might be a "`Name-Match`" service available that given a name like "Jim" returns "James" and "J.". A query that keeps the first name open and rather relies on such a service could look as follows:

```
xmlns: SRI = ``http://www.ai.sri.com/daml/
                      ontologies/Researcher#''
xmlns: S = ``http://www.ai.sri.com/daml/
                    ontologies/Service#''
FIND <SRI:Researcher>
SUCH-THAT <SRI:Researcher:lastName>
            Hendler
          </SRI:Researcher:lastName>
          <SRI:Researcher:firstName>
            USE <S:Service>
                  <S:Service:name>
                    Name-Match
                  </S:Service:name>
                  <S:Service:par>
                    James
                  </S:Service:par>
                </S:Service>
          </SRI:Researcher:firstName>
END
```

The intended meaning of this ad hoc notation is that DAML-Q is supposed to make use of a service in order to determine the possible first names. The name of the service to be used is `Name-Match` and it is to be consulted with the parameter `James`. This service would deliver a set of possible names that are associated with "James" that will be used in processing the query. Service specifications of various kinds are one of the key foci of our research and are described in greater detail below.

As a last possible scenario we consider the situation where a service on the Web has stored specialty information, the so-called DAML-R (for DAML speciality repository). DAML-R would provide comprehensive information for specific topics. For instance, there may exists a service that has cached links to home pages of researchers in a specific field, or that has cached the most relevant information about those researchers locally (e.g., their names, affiliations, publication references). Such a service can obtain its information offline, and manipulate and store it in a way that allows high throughput of queries.

3. INFERENCE IN QUERIES

We have shown how the differing ways information can be represented on the Web can call for successively more complex processing, even for simple queries. But for complex queries, this is even more true. In our initial list of sample queries, all but the first two require significant inference capabilities.

As part of our research, we have been considering what kinds of inference are necessary to answer plausible queries and to perform typical tasks using Web-based documents and services. We have been considering what constructs will be necessary in the language to allow designers of a web site to advertise its services. Also we have been investigating the representation of the background knowledge necessary to connect queries with the web sites that supply the answers.

We have been using the theorem prover SNARK [11] as a vehicle for our experiments with inference. This is not to say that SNARK needs to be incorporated into a DAML search engine, but rather that by studying the SNARK inferences, we can see what kinds of processing suffice to handle DAML queries.

SNARK is an automatic first-order theorem prover, implemented in Common Lisp, that can be tuned and specialized to exhibit high performance in particular subject domains. It has a highly evolved sort structure that enables us to represent taxonomic information concisely and to infer consequences quickly. It has built-in facilities for fast temporal reasoning. It has answer-extraction facilities, which allow it to answer questions instead of merely proving theorems. These facilities were developed for deductive program synthesis [2], [6], but apply as well to query answering.

SNARK has a procedural-attachment mechanism, which makes it possible to link symbols in the logic with procedures; when the symbol is employed in the proof, the corresponding procedure is executed. This enables us to invoke outside sources, including web sites, while the proof is in progress. The examples in this section are expressed in SNARK notation; we intend that this language be inter-translatable with the logic language of DAML.

3.1 An Experiment with Document Search

As an experiment, we consider what kinds of inference are required to answer queries for searching for documents. We phrase this query in terms of the ontologies and theories we have developed for bibliographical references, names, addresses, topics, and dates. We then use SNARK to find answers, based on theories we have developed for these areas.

In reading the example, one must distinguish between strings, names, and entities. For example, `"James Hendler"` is a string, `(personq "James Hendler")` is a name, and `james-hendler` is a person. While `james-hendler` refers to a unique individual (ultimately a URI), the name `(personq "James Hendler")` is shared by several people. These distinctions seem pedantic, but when we try to ignore them we get into trouble.

The relation `person-val` relates a name of a person with the person itself. Thus, if

```
(person-val ?personq ?person)
```

holds, `?personq` is a name for `?person`. While, by convention, `?person`, `?person0`, `?person1`, etc., are variables that range over people, `?personq`, `?personq0`, etc., range over names of people.

A paper is of sort `paper`; a reference to a paper is of sort `paperq`. Thus we think of a reference as a kind of name for a paper.

Other concepts are best explained in the context of the example. We consider Query 3 from Section 1.1:

> Find a reference to the most recent paper on SHOE with James Hendler as a co-author.

This query may be phrased in the SNARK language as:

```
(find
  ?paperq
 such-that
  (and
   (pub-val ?paperq ?paper)
   (author ?paper ?person)
   (person-val (personq "James Hendler") ?person)
   (about ?paper (topic "SHOE"))
   (= (pub-to-year ?paper) (year-fn ?natural)))
 prefer
  starts-after-starting-of
 on
  (year-fn ?natural)
 time-limit
  10)
```

In other words, we want to find a reference `?paperq` such that

- `?paperq` is a reference to `?paper`—the relation `pub-val`, analogous to `person-val`, relates a publication reference with the corresponding publication.
- `?person` is an author of `?paper`—there may be other authors.
- `(personq "James Hendler")` is a name for `?person`.
- `?paper` is about the topic SHOE; if the topic ontology contained subtopics of SHOE, papers on those subtopics would be acceptable.
- `?paper` was published in year `(year-fn ?natural)`; e.g., while `2000` is a natural number, `(year-fn 2000)` is the year 2000.

Furthermore, if there are several papers found that satisfy the above criteria, we prefer the latest one. The relation

```
starts-after-starting-of
```

is a relation on time intervals such that

```
(starts-after-starting-of
 ?time-interval1
 ?time-interval2)
```

holds if `?time-interval1` starts after `?time-interval2`; thus

```
(starts-after-starting-of
 (year-fn 2000)
 (year-fn 1997))
```

We execute this query by first finding a single paper reference that meets all our criteria. We then look for another one, with a later publication date. We do not ask for the latest of all of Hendler's publications, since we can never be sure if we have seen all of them; instead, we give the search a time limit, and return the latest paper we have found in that time.

We assume we have access to DAML-annotated publication lists; the following is a description of a 1997 paper on SHOE ([5]), of which Hendler is a co-author.

```
<pub:Inproceedings-ref ID= "97:_ont_ba_web_ag">
<pub:author> "Sean Luke" </pub:author>
<pub:author> "Lee Spector" </pub:author>
<pub:author> "David Rager" </pub:author>
<pub:author> "James Hendler" </pub:author>
<pub:title>
 "Ontology-based Web Agents "
</pub:title>
<pub:booktitle>
"Proceedings of the First International
 Conference on Autonomous Agents (Agents97)"
</pub:booktitle>
<pub:year> "1997" </pub:year>
<pub:publisher>
"Association for Computing Machinery"
</pub:publisher>
<pub:address> "New York, NY, US" </pub:address>
<pub:topic> "SHOE" </pub:topic>
</pub:Inproceedings-ref>
```

This may be represented in SNARK notation as:

```
(assert
 '(pub-val
   (inproceedingsq
    author (coq
            (personq "Sean Luke")
            (personq "Lee Spector")
            (personq "David Rager")
            (personq "James Hendler"))
    title (titleq "Ontology-based Web Agents")
    booktitle
    (titleq "Proceedings of the First
             International Conference
             on Autonomous Agents
             (Agents97)")
    year (year-fn 1997)
    publisher
    (publisherq
     "Association for Computing Machinery")
    address
     (cityq "New York" "NY" "US")
    topic (topic "SHOE"))
   97:_ont_ba_web_ag)
 :name 'shoe-acm-paper-reference)
```

Here

```
(coq ?personq1 ?personq2 ...)
```

is a publication list containing names `?personq1`, `?personq2`, ... Names, indicated by the keyword `:name`, have no logical or functional significance; they are used only to make the proof traces easier for us to follow. Our representation for references in logic is derived from one developed in Maude [1] by Meseguer.

The reference indicates that the paper appears in a conference proceedings, gives the title of the paper, the title of the proceedings, the date of publication, the publisher, the address of the publisher, and the topic. The notation for references in this theory is based on that of Bibtex.

Let us also assume that we have the reference to a later SHOE paper ([3]):

```
(assert
 '(pub-val
   (inproceedingsq
    author (coq
            (personq "Jeff Heflin")
            (personq "James Hendler"))
    title (titleq "Searching the Web with SHOE")
    booktitle
    (titleq
     "Artificial Intelligence for Web Search.
      Papers from the AAAI Workshop.")
    year (year-fn 2000)
    publisher (publisherq "AAAI Press")
    number (pubnumber "WS-00-01")
    address (cityq "Menlo Park" "CA" "US")
    topic (topic "SHOE"))
   shoe-aaai-paper)
 :name 'shoe-aaai-paper-reference)
```

Let us follow a few steps of the SNARK inference process by which an answer was found.

We begin with a logical sentence obtained from the query

```
(Row 193
 (or (not (pub-val ?paperq ?paper))
     (not (author ?paper ?person))
     (not (person-val
           (personq "James Hendler")
           ?person))
     (not (about ?paper (topic "SHOE")))
     (not (= (pub-to-year ?paper)
             (year-fn ?integer&nonnegative))))
 Answer (ans ?paperq
             (year-fn ?integer&nonnegative)))
```

Note that, because SNARK is a refutation procedure, queries are negated, and inference proceeds until a contradiction is obtained. Also note that the query, and its logical descendents, is accompanied by an `Answer` expression indicating what answer we expect to obtain from the proof. Because our preference is based on the year, we include the year of publication as part of our answer. The expression `integer&nonnegative` is SNARK's internal notation for the sort of natural numbers.

Formulas and their associated answers are called "rows".

We omit some details of the proof.

Using this assertion, and others from our publication ontology, we obtain the first answer

```
(Row 335
 false
 (resolve 334 name-of-lee-spector)
 Answer (ans
  (assert
   '(pub-val
     (inproceedingsq
      author (coq
              (personq "Sean Luke")
              (personq "Lee Spector")
              (personq "David Rager")
              (personq "James Hendler"))
      title (titleq "Ontology-based Web Agents")
      booktitle
      (titleq "Proceedings of the First
               International Conference
               on Autonomous Agents
               (Agents97)")
      year (year-fn 1997)
      publisher
      (publisherq
```

```
    "Association for Computing Machinery")
   address
    (cityq "New York" "NY" "US")
   topic (topic "SHOE"))
        (year-fn 1997)))
```

Note that, since SNARK is a refutation procedure, obtaining a row `false` indicates that a proof is complete.

Since we want the latest possible publication, we begin a new query, to find a publication later than 1997:

```
(Row 324
     (or (not (pub-val ?paperq ?paper))
         (not (author ?paper ?person))
         (not (person-val
                (personq "James Hendler")
                ?person))
         (not (about ?paper (topic "SHOE")))
         (not (= (pub-to-year ?paper)
                 (year-fn ?integer&nonnegative)))
         (not (starts-after-starting-of
               (year-fn ?integer&nonnegative)
               (year-fn 1997))))
     Answer (ans ?paperq
                 (year-fn ?integer&nonnegative)))
```

This query is the same as the one we started with, except it contains an additional condition:

```
  (starts-after-starting-of
   (year-fn ?integer&nonnegative)
   (year-fn 1997))
```

In other words, we insist that the publication we find is more recent than 1997.

The refutation proceeds similarly to what we have seen already, except this time SNARK finds the second reference, to the AAAI 2000 paper.

SNARK uses a temporal reasoning procedure, based on the Allen temporal calculus, to check temporal constraints, whether they arise directly from the query or are a consequence of our preferences. This allows us to determine that the Year 2000 paper is indeed more recent than the 1997 paper. (It could also discriminate on the basis of publication month and day.) Thus, the AAAI paper satisfies all the constraints and is our next best selection:

```
(Row 421
     false
     (resolve 420 name-of-jeff-heflin)
     Answer
     (ans
      (inproceedingsq
       author
       (coq
        (personq "Jeff Heflin")
        (personq "James Hendler"))
       title
       (titleq "Searching the Web with SHOE")
       booktitle
       (titleq
        "Artificial Intelligence for Web Search.
         Papers from the AAAI Workshop.")
       year (year-fn 2000)
       publisher (publisherq "AAAI Press")
       number
       (pubnumber "WS-00-01")
       address (cityq "Menlo Park" "CA" "US")
       topic (topic "SHOE"))
      (year-fn 2000)))
```

SNARK will continue to look for Hendler SHOE papers more recent than 2000; when the time limit is exceeded, it returns the reference to the best paper we have found.

It is also possible to obtain lists of papers, ordered by our preference, so that the more recent are returned first.

We can also use inference to combine capabilities of multiple web sites. For example, suppose we had forgotten Hendler's name, but remembered he worked for the University of Maryland. Then we could phrase the query

```
(find
  ?paperq
 such-that
  (and
   (pub-val ?paperq ?paper)
   (author ?paper ?person)
   (employed-by
    ?person
    (org "University of Maryland"))
   (about ?paper (topic "SHOE"))
   (= (pub-to-year ?paper)
      (year-fn ?natural)))
 prefer
  starts-after-starting-of
 on
  (year-fn ?natural)
 time-limit
  10)
```

The inference process can then consult the University of Maryland web page to be sure that at least one of the co-authors of the papers it finds in the bibliography is employed there.

SNARK is limited to first-order logic and has no special facilities for combining theories. We have been working with Jose Meseguer to connect Maude [1], a higher-order language with metalogical capabilities, with SNARK. This would make our presentation of theories simpler and more elegant. For instance, now we have separate sorts, `personq` and `person`, for names and people respectively. Similarly we have separate sorts, `cityq` and `city`, for names of cities and the cities themselves. With Maude we will be able to have a function `name` that maps each sort into the corresponding name sort; thus we would not need new sorts `personq` and `cityq`.

Also Maude would give us more flexible syntax and the ability to develop new theories in a modular way, by instantiating and combining old ones.

4. DEFINING AND ACCESSING WEB SERVICES

In the preceding sections, we have discussed the mechanisms by which DAML will facilitate the representation and discovery of objects containing information on the Web. We turn now to a consideration of services on the Web, with two basic observations: first, it should be possible to use these same mechanisms for representing and discovering services; second, there is a close, interleaved relationship between querying, or searching, the Web, and making use of services on the Web. Queries 4 and 5 of Section 1 are requests for services, and not merely for information.

Today, for the most part, services on the Web are created for human consumption, with interfaces intended for humans, not for software agents. In contrast, a DAML-enabled Web, while still allowing for human consumption of services, will make software agents first-class citizens of the Web, with full access to its services. This, in turn, will allow the number and variety of services offered on the Web, and the efficiency with which they are used, to continue to increase at a dramatic rate.

Web services can be of many types, including but not limited to the types that are already familiar to human users: querying of databases, catalogs, digital libraries, and other types of information repositories, searches and classification services provided by portals, business-to-consumer (B2C) transactions, business-to-business (B2B) transactions, etc. It is worth noting that Web services are not limited to the two-party, client/server approach most commonly used. Services can involve any number of parties, with complex patterns of interaction between them. For example, a business-to-business transaction may well involve a buyer, a seller, a financer, and a shipper.

To make use of a Web service, a software agent needs a formal description of the service, and the means by which it is accessed. An important goal for DAML, then, is to establish a framework within which these descriptions are made and shared.

4.1 Logical Advertisement of Services

The author of a DAML-annotated web page will need to provide sufficient information to link the services provided by the web page with the concepts in an appropriate theory or ontology. Usually there will not be an exact match between the service and a symbol in the theory; rather, the author will need to describe a logical relationship between each service and the concepts of the theory.

For one thing, the web site functions are defined on concrete data types, such as strings or real numbers; the functions and relations in the theory will be defined on abstract entities, such as cities and people, independently of how they are represented. A query will not necessarily be phrased in terms of the same representation selected by a web site. Most people will not even know the concrete functions and relations computed by particular web sites.

For example, the Travelocity site `www.travelocity.com` computes many functions. One of them returns the airports local to a city, with their respective distances from the city. We cannot assume that the user knows that Travelocity provides this service. There may, however, be an ontology that defines an relation `city-airport`, that holds between a city and its local airports, which are abstract entities.

The service that Travelocity provides for finding local airports can then be advertised using the following logical axiom.

```
(assert
 '(implies
   (city-airport-tvl
    ?city-string ?state-abbr ?country-abbr
    ?airport-code ?airport-string ?real ?string)
   (city-airport
    (city ?city-string ?state-abbr ?country-abbr)
    (airport ?airport-code)))

 :name 'travelocity-city-airport
 :documentation
 "Travelocity can determine which airports are
  local to a given city.  The Travelocity
  relation city-airport-tvl implements
  the ontology relation city-airport")
```

While `city-airport` is the abstract relation between a city and its local airports, `city-airport-tvl` is the concrete relation computed by the Travelocity web site that finds airports local to a given city, and their distances in miles from the city. While `city-airport` applies to cities and airports, which are abstract entities, `city-airport-tvl` applies to concrete strings and real numbers, which are a particular representation of the abstract entities. The relationship between the concrete strings and numbers and the abstract entities is expressed by functions and relations from the ontology itself.

Travelocity choses to represent a city with three name-strings, for the city, the state (or region), and the country, respectively. For instance, the function `city` maps these name-strings into the appropriate city, an abstract entity. Thus

```
(city "New York" "New York" "US")
```

is the abstract entity New York City. Similarly, the function `airport` maps a name-string or a code for an airport into the corresponding airport.

This service provided by Travelocity can be advertised in terms of other concepts from the same or other ontologies. For instance, the above assertion does not say anything about the distances that the web site computes, between a city and its local airport. That information is provided by an additional assertion.

```
(assert
 '(implies
   (city-airport-tvl
    ?city-string ?state-abbr ?country-abbr
    ?airport-code ?airport-string ?real ?string)
   (=
    (distance-between
     (city ?city-string ?state-abbr ?country-abbr)
     (airport ?airport-code))
    (miles ?real)))

 :name 'travelocity-city-airport-distance
 :documentation
 "Travelocity can determine the distance between
  a city and its local airports")
```

Here, `distance-between` gives the abstract distance between two places, independent of unit of measurement; `miles` maps a concrete real number into an abstract distance, the corresponding distance in miles. There are other functions, for kilometers, feet, etc. This representation does not identify numbers with distances; rather, it allows web sites that only deal with miles to communicate with web sites that only deal with kilometers.

These examples show the importance of having some logical power to advertise the services of web sites. We can see the value of a sorted logic with equality. Although one could phrase the same assertions in unsorted logic without equality, they would be unwieldy to write and more inefficient to reason with. In particular, without sorts we would need to prefix each assertions with conditions, such as

```
(implies
 (and
  (real ?real1)
  (city-string ?city-string)
  ...)
 ...)
```

Without equality, it is peculiarly awkward to reason about functions, for example to express the uniqueness of their values.

In addition to such linking assertions, we will need other assertions for background inference, to link queries with web sites and to link different web sites together. For example, the following assertion tells us that the time required to travel from one place to another is limited by the distance between them and by the person's best speed.

```
(assert
 '(implies
   (and
    (at ?person ?place1 ?time-point1)
    (at ?person ?place2 ?time-point2))
   (=<
    (distance-between ?place1 ?place2)
    (*-quantity
     (abs-quantity
      (time-minus ?time-point1 ?time-point2))
     (speed-between ?place1 ?place2)
```

```
    )))
:name 'distance-equals-rate-times-time
:documentation
"If a person is going from ?place1 at
?time-point1 to ?place2 at ?time-point2, the
distance between the points is less than or
equal to the product of the difference between
the times and the best speed between the two
places.")
```

While this assertion does not directly refer to any particular web site, it is useful in handling queries that do invoke web sites. For instance, if we are using Travelocity to book a trip, the decision about which flight to book might be determined by the time required to travel between the airport and the ultimate destination city.

It may not be necessary to express background assertions in the DAML language, but it will be necessary to invoke such assertions in the course of handling queries. It will be important that people writing DAML-annotated web pages should find it easy to write declarative statements that describe how the services offered by that web page are related to some theory. While they need not write them in logic directly, they will need to write them with some tool that translates them into logic. If the logical language is less expressive, that translation will be more difficult. It is not true that using a less expressive language will give us more efficiency of inference. If we paraphrase the above assertions without using equality and sorts, the result will be less efficient inference, not more.

4.2 Toward an Ontology for Describing Services

Web sites should be able to employ a set of basic classes and properties for declaring and describing services, and the ontology structuring mechanisms of DAML provide the appropriate framework within which to do this. In this subsection, we sketch out an initial proposal for these basic classes and properties.

A note on terminology: In what follows, when we refer to the "user" of a service, we are thinking of a software agent, unless stated otherwise. (Of course, a software agent, in most cases, acts on behalf of some human *user*.)

We propose here some general principles by which a DAML ontology of services may be organized, and a few fundamental classes and properties that will provide the backbone of any DAML service ontology. (We will refer to these fundamental classes and properties as "built-in" classes and properties, because we expect they will form a coherent layer of the DAML language, which is generic enough to be used as an upper ontology for all service descriptions.) Note, however, that we are not proposing any particular taxonomy of service types; indeed, we are neutral as to whether there should be one such taxonomy, a few, or a great many. Our concern here is to provide the mechanisms and concepts by which any number of such hierarchies can be constructed and used.

Our proposal begins, naturally enough, with built-in class *Service*. The categories of a taxonomy of services will be subclasses of class *Service*, and will be structured according to the needs to a given domain or application. For example, a bookseller's web site might make use of a toplevel *B2C-Service* class, the immediate subclasses of which might be *B2C-InformationService* (including such things as search and recommendation of books), *B2C-Transaction*, *B2C-AccountMaintenance*, and *B2C-PurchaseTracking*.

4.3 The Fundamental Properties of a Service

Our structuring of the ontology of services is motivated by the need to provide three essential types of knowledge about a service, which may be intuitively characterized by these questions:

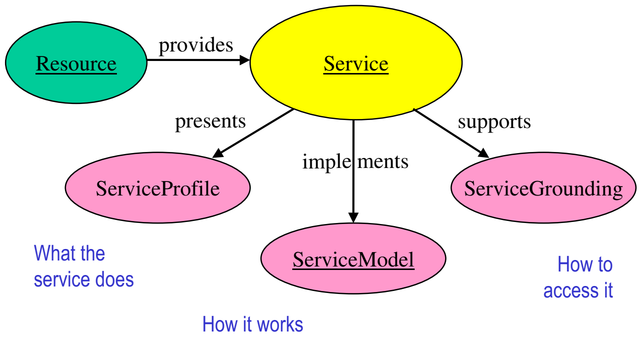

Figure 5: Top level of service ontology

- *What does the service require of the users and provide for them?*
- *How does it work?*
- *How is it used?*

As shown in Fig. 5, these three types of knowledge are captured by the built-in properties *presents*, *implements*, and *supports*, of class *Service*. These properties range over built-in classes *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*, respectively. These are "placeholder" classes; that is, they have no important properties defined. The substantive properties are left to be defined by their various subclasses. For each descendant class S of *Service*, there will exist corresponding subclasses of *ServiceProfile*, *ServiceModel*, and *ServiceGrounding*, containing the properties that can most appropriately represent the required information for class S.

It is important to note that logical rules may be employed in any or all of (the subclasses of) ServiceProfile, ServiceModel, and ServiceGrounding, so as to support the uses of inferential reasoning discussed earlier in this paper.

- **Service profiles.** The specification of what a service requires and provides calls for an *external*, or "black-box", view of the service, giving such things as preconditions that must hold before the service can be requested, required inputs, effects of the service, expected outputs, cost of accessing the service, acceptable forms of payment, and so on. The service profile is meant to provide the essential information by which a potential user of the service (or perhaps a matchmaker agent) can determine if the service meets the user's needs. The role of a service profile is analogous to that of a procedure declaration (header) in a typed programming language, or the schema of a database relation. (Indeed, for many simple, one-step, services, the service profile may be essentially the same as a procedure header or schema.)

- **Service models.** The specification of how a service works calls for an *internal*, or "glass-box" view of the service. The type of representation used to display this information will vary with the nature of the service. For example, a purchase from a Web site may involve a number of steps (specifying quantity, desired color, and other purchase attributes; giving shipping address; securely inputting credit card information; verification of credit card information; final confirmation). The service model describes the shared knowledge of these steps, which is required for the user(s) and service provider to coordinate their activities. For many such services, the representations associated with process modeling will be essential in describing this knowledge. Thus, we expect there to be one or more important subclasses of *ServiceModel* based on process modeling. One such approach, under development at SRI, is described in the following section.

- **Service groundings.** A service supports one or more service groundings, each of which spells out the implementation-specific details by which the user communicates with the service, such as message formatting, transport mechanisms, etc. A service grounding will typically be a well-known set of specifications for exchange and interaction, and will play a role such as is played today by standards such as HTTP forms protocol, CORBA IDL, SOAP, OAA's ICL, KQML, Java's RMI, and the like.

  In addition, the grounding must specify, for each abstract type specified in the ServiceModel, an unambigous way of exchanging data elements of that type with the service (that is, marshalling / serialization techniques employed). The likelihood is that, as a result of market forces, a relatively small set of groundings will come to be widely used in conjunction with DAML services. Groundings will be declared and documented in detail at various well-known URIs.

In a nutshell, the relationship between these three fundamental classes is this: The service model, which provides the fullest description of the service, gives an abstract, semantic description of what the service does. (Thus, a given service model could be reused for a variety of services accessed in widely varying ways.) Each of the service's groundings spells out how the data types and messages, required by the service model, are to be formatted and communicated in using a specific instance of the service. Taken together, the ServiceModel and ServiceGrounding object associated with a Service will give enough information for an agent to make use of a newly discovered service.

The service profile summarizes the service model, by giving an external view of it (inputs, outputs, preconditions, postconditions), and may also provide additional information needed to determine the appropriateness of the service to a potential user. Generally speaking, a potential user of a service (or a matchmaker for the user) examines the service's service profile to determine if the user can provide the service's inputs and make use of its outputs, and examines the service's groundings to determine if it is competent to interact with the service provider (that is, knows how to use one or more of those service groundings).

In the following section, we discuss how a process-based service model can facilitate the use of interaction sequences for a wide variety of transaction-based services, such as those typical of B2C web sites.

5. MODELING SERVICE PROCESSES

Much of the Web-based ontological content (developed using DAML, OIL and other languages) is based on static attribute-value feature structures arranged in taxonomies. This representation is ideal for describing and representing complex objects (*nouns*) such as documents, organizations, or bibliographies in a manner that supports structured queries such as the first three of our example queries.

A significant missing piece in the currently available ontological content is the representation of the dynamic content of *services* (procedures, protocols, behaviors and transactions). Much of the Web is about providing and using services such as reserving a ticket, interacting with a secure site, buying/selling/trading, and making an appointment with a doctor. Services are distinguished from objects in that they require representing changes in the state of the world and the *dynamics* of interaction. In this way, services are more like *verbs* and actions than like *nouns*. Thus representing services requires tapping into an ontology/theory of actions, processes and events.

5.1 A Core Theory of Processes and Events

Consider the examples of Queries 4 and 5 (Section 1). These queries are information requests that result in actions changing the state of the world in a specific way (one results in a buying action from amazon.com, the other in placing a library hold on a book). In order for these queries to be processed, the Web agent needs to know at least the following: a) *input/output* format/protocol(OAA, Jini), b) *parameters* of interaction (payment method, shipping method, etc., c) *dynamics* of interaction (steps involved and the internal temporal structure of a `buying` action), d) *preconditions/effects* of a specific action (selecting a book results in its being in the shopping cart), e) *resources* consumed (money), produced or locked (credit-card number), f) ways of *controlling* the transaction, such as short cuts (using stored profile information), interrupting or canceling from various stages/states in the interaction (you can cancel an order before shipping).

Continuing with the example, note that buying is not an isolated concept but is integrated with a cluster of concepts in what might be called the *commercial transaction frame*. The commercial transaction frame involves such concepts as possession, change of possession (giving, taking/receiving), exchange (the parties in the exchange accept and are expected by their community to accept the results of the exchange), and money. The basic parameters/roles of this frame, then, will include Money, the Goods (standing for goods or services), the Buyer (the person who surrenders money in exchange for the goods), and the Seller (the person who surrenders the goods in exchange for the money). Further elaborations, needed for describing some of the peripheral terms in this frame, involve certain details of the exchange: in some cases, for example, we need to identify the Price, a ratio between the quantity of money given and the quantity of goods received (e.g. two dollars an ounce), temporal features of the exchange (perhaps the payment is spread over a period of time), the difference between the tender and the price (Change), and so on. Still further elaborations can separate the owner of the goods or the owner of the money from the actual participants in the exchange arrangement.

We have been developing just such a parameterized model of the structure of events and processes, and it can support tools for agents to enter, modify, advertise and communicate the capabilities, types and details of events they can observe, monitor, model and control. The abstract theory of event structure has three major components[1].

1. **Temporal structure and evolution trajectories:** The fine-structure of events comprises of a set of key states (such as enabled, ready, ongoing, done, suspended, canceled, stopped, aborted) and a partially ordered directed graph that represents possible evolution trajectories as transitions between key states (such as prepare, start, interrupt, finish, cancel, abort, iterate, resume, restart). Each of these transitions may be atomic, timed, stochastic or hierarchical (with a recursively embedded event-structure).

2. **Process primitives and event construals:** Events may be punctual, durative, (a)telic, (a)periodic, (un)controllable, reversable, (ir)reversable, ballistic or continuous. Thay may

[1] For further details, comparisons with hybrid system (discrete/continuous) models [4] and an extended treatment of the representation and semantics of verbs and events, see [9, 10].

describe their resource interactions through relations such as locking, producing, creating, transforming, consuming or destroying. They may support defeasable construal operations of shifting granularity (*elaboration* (zoom-in), *collapse* (zoom-out)) and enable *focus*, *profiling* and *framing* of specific parts and participants.

3. **Inter-event relations:** A rich theory of inter-event relations allows sequential and concurrent enabling, disabling, or modifying relations. Examples include interrupting, starting, resuming, canceling, aborting or terminating relations. In each of these cases, we have a precise semantics in terms of the overall structure of the interacting events.

5.2 A DAML Encoding of Processes

We have a preliminary DAML implementation of many aspects of our core theory of processes and events. We are developing a core DAML ontology of services in a larger collaborative effort with Stanford KSL[7, 8], CMU, BBN, and Nokia as part of the DAML service specification language effort. Some aspects of this core ontology are described below.

The basic class that implements process-specific information is appropriately termed the EVENT class. Event specific information includes parameters such as the name of the event, where it is to execute, documents updated, read or written as part of execution. These properties obviously refer to *noun* ontologies like name ontologies or document ontologies (developed by us or other ontology developers).

```
<rdfs:Class rdf:ID="Event">
 <rdfs:comment> A simple event </rdfs:comment>
 <rdfs:subClassOf rdf:resource="ServiceModel"/>
</rdfs:Class>
```

Processes have a name, inputs, outputs, participants, parameters, preconditions, and effects. Each input, output, parameter, precondition or effect is a property of event left unrestricted at this level (it ranges over ”Thing”). Collections of input, output, etc are are unrestricted bags (items are anything). Shown below is an illustrative sample from our existing ontology. Complete versions can be found at `http://www.ai.sri.com/ontologies/services/Process.daml`.

```
<rdf:Property rdf:ID="name">
  <rdfs:domain rdf:resource="#Event"/>
  <rdfs:range rdf:resource="rdfs:Literal"/>
</rdf:Property>

<rdf:Property rdf:ID="precondition">
<rdfs:domain rdf:resource="#Event"/>
<rdfs:range  rdf:resource="daml:Thing"/>
</rdf:Property>

<rdf:Property rdf:ID="parameter">
<rdfs:domain rdf:resource="#Event"/>
<rdfs:range  rdf:resource="daml:Thing"/>
</rdf:Property>

<rdf:Property rdf:ID="effect">
<rdfs:domain rdf:resource="#Event"/>
<rdfs:range  rdf:resource="daml:Thing"/>
</rdf:Property>
```

Parameters are further typed into input, output and participants.

```
<rdf:Property rdf:ID="input">
 <rdfs:subPropertyOf rdf:resource="#parameter"/>
</rdf:Property>

<rdf:Property rdf:ID="output">
  <rdfs:subPropertyOf rdf:resource="#parameter"/>
</rdf:Property>

<rdf:Property rdf:ID="participant">
  <rdfs:subPropertyOf rdf:resource="#parameter"/>
</rdf:Property>
```

A theory of events has to link to a theory of time. We have been developing a fairly rich theory of time [1, 11] that includes a theory of time intervals and a theory of time points with varying densities (qualitative, integer, reals, etc.). Our DAML encoding of the start time of a process may thus use the point theory while the duration of a process may be specified using the interval theory.

```
<Property ID="startTime">
 <comment> Start time for the process </comment>
  <domain resource="#Process"/>
  <range resource="sri-time:Time"/>
</Property>

<Property ID="duration">
 <comment> Duration of the process </comment>
  <domain resource="#Process"/>
  <range resource="sri-time:TimeInterval"/>
</Property>
```

Processes are complex events that have additional properties with respect to the ordering and conditional execution of individual events. The attempt here is to come up with a minimal set of process classes that can be specialized to specify a variety of web services.

```
<rdfs:Class rdf:ID="Process">
<rdfs:comment>
A process is a composite event.
</rdfs:comment>
<rdfs:subClassOf rdf:resource="#Event"/>
</rdfs:Class>
```

There are two fundamental relations between events and processes. One pertains to *expanding* an event to its underlying process (zoom-in) and the other corresponds to *collapsing* a process into an atomic event (zoom-out).

```
<rdf:Property rdf:ID="expand">
  <rdfs:domain rdf:resource="#Event"/>
  <rdfs:range rdf:resource="#Process"/>
</rdf:Property>

<rdf:Property rdf:ID="collapse">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="#Event"/>
</rdf:Property>
```

Often we want to define operations on multisets of processes (buying involves simultaneous transfers of money (from credit card bank to seller) and goods (from seller to buyer). To do this we define a PROCESS BAG class that is a subclass of the RDF (`http://www.w3.org/TR/REC-rdf-syntax`) BAG class.

```
<rdfs:Class rdf:ID="ProcessBag">
<rdfs:comment> A multiset of Events </rdfs:comment>
  <rdfs:subClassOf rdf:resource="rdfs:Bag"/>
    <rdfs:subClassOf>
     <daml:Restriction>
```

```
      <daml:onProperty rdf:resource="#item"/>
      <daml:toClass rdf:resource="#Process"/>
     </daml:Restriction>
    </rdfs:subClassOf>
</rdfs:Class>
```

Now we are able to define more complicated assemblages of processes. For instance, SEQUENCE is a process that is comprised of a list of subprocesses, while CONCURRENT is a process which comprises a multiset of subprocesses.

```
<rdfs:Class rdf:ID="Sequence">
  <rdfs:subClassOf rdf:resource="#Process"/>
  <rdfs:subClassOf rdf:resource="#ProcessList"/>
</rdfs:Class>

<rdfs:Class rdf:ID="Concurrent">
  <rdfs:subClassOf rdf:resource="#Process"/>
  <rdfs:subClassOf rdf:resource="#ProcessBag"/>
</rdfs:Class>
```

We have also implemented an algorithm that can recursively construct executable process models of the type described in (Narayanan 1999) given DAML descriptions of the dynamic content of services. Such models can be constructed automatically by agents that encounter specific services. Agents can then use these models to track execution of service requests and responses, monitor requests and task performance and to plan and schedule specific service related tasks. While the details of the construction algorithm and the process modeling environment are outside the scope of this paper, the reader can obtain further details from `http://www.ai.sri.com/daml/services`.

A rich and structured core theory of service and an automatic construction algorithm that compiles into an executable model should provide us with motivated constraints to design interfaces that allow service providers to describe their services at a high level using domain specific vocabulary. We envison this interaction environment to be a guided core-theory based graphical and natural language dialog. Building such a semantically grounded service authoring environment is a current focus of our work.

In summary, an ongoing project addresses a correlated set of essential missing components in the current state of the Web: theories, languages and authoring environments for describing the dynamic content of services. Such models require a rich ontology and theory of events and processes and will be crucial in enabling and coordinating the activities of autonomous agents. This holds true regardless of whether the underlying context is one of controlling devices, carrying out complex tasks, or obtaining information. Indeed, in a Web enabled for intelligent agents, all of these contexts will become intertwined.

5.3 Status of Service/Process Ontology

We are developing DAML declarations for the fundamental classes and properties mentioned above for describing services and process. The latest versions of these may be found at the URL

http://www.ai.sri.com/daml/services.

6. CONCLUSION

We have presented our vision of a DAML-enabled search architecture, describing several scenarios for how queries will be processed amd identifying the main software components necessary to facilitate the search. We examined the issue of inference in search, and we address the issue of characterizing procedures and services in DAML, enabling a DAML query language to find web sites with specified capabilities. These are, we believe, some of the most critical components of the language required for enabling the Semantic Web.

7. REFERENCES

[1] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. Quesada. *Maude: Specification and Programming in Rewriting Logic*. SRI International, Computer Science Laboratory, Menlo Park, CA, January 1999. `http://maude.csl.sri.com/manual/`.

[2] C. C. Green. Application of theorem proving to problem solving. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 219–239, May 1969.

[3] J. Heflin and J. Hendler. Searching the web with shoe. In *Artificial Intelligence for Web Search. Papers from the AAAI Workshop.*, number WS-00-01, Menlo Park, CA, US, 2000. AAAI Press.

[4] T. A. Henzinger. The theory of hybrid automata. In *LICS 96 (expanded version)*, pages 278–292, 1998.

[5] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based web agents. In *Proceedings of the First International Conference on Autonomous Agents (Agents97)*, New York, NY, US, 1997. Association for Computing Machinery.

[6] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming, Languages, and Systems*, 2:90–121, 1980.

[7] S. McIlraith, T. Son, and Z. H. Mobilizing the semantic web with daml-enabled web services. In *Proceedings of the Second International Workshop on the Semantic Web (SemWeb'2001)*, May 2001.

[8] S. McIlraith, T. Son, and Z. H. Semantic web services. In *IEEE Intelligent Systems*, March/April 2001.

[9] S. Narayanan. *Knowledge-based Action Representations for Metaphor and Aspect (KARMA)*. PhD thesis, Computer Science Division, EECS Department, University of California at Berkeley, 1997.

[10] S. Narayanan. Reasoning about actions in narrative understanding. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI-99)*. Morgan Kaufmann Press, 1999.

[11] M. Stickel, R. Waldinger, and V. Chaudhri. *A Guide to SNARK*. SRI International, Artificial Intelligence Center, Menlo Park, CA, 2000. `http://ai.sri.com/snark/tutorial.html`.