LCW-Based Agent Planning for the Semantic Web

Jeff Heflin and Hector Muñoz-Avila

Dept. of Computer Science & Engineering
Lehigh University
19 Memorial Drive West
Bethlehem, PA 18015
{heflin, munoz}@cse.lehigh.edu

Abstract

The Semantic Web has the potential to allow software agents to intelligently process and integrate the Web's wealth of information. These agents must plan how to achieve their goals in light of the information available. However, because the Web is so vast and changes so rapidly, the agent cannot make a closed-world assumption. This condition makes it difficult for an agent to know when it has gathered all relevant information or when additional searches may be redundant. We propose to use local closed world (LCW) information to assist these agents. LCW information can be obtained by accessing sources that are described in a Semantic Web language with LCW extensions, or by executing operators that provide exhaustive information. In this paper, we demonstrate how two Semantic Web languages (DAML+OIL and SHOE) can be augmented with the ability to state LCW information. We also show that DAML+OIL can represent many kinds of LCW information even without additional language features. Finally, we describe how ordered task decomposition can be used with LCW information to efficiently plan in distributed information environments.

1 Introduction

The World Wide Web has transformed society, changing the way people communicate, learn, and conduct business. However, the dramatic growth of the Web over recent years has made it increasingly difficult for people to take full advantage of its capabilities. One solution is to build sophisticated intelligent agents, programs that can autonomously take actions in order to achieve their owners' goals. These agents could process and integrate the multitude of data available, filtering information for users or acting on their behalf. The Semantic Web (Berners-Lee, Hendler, and Lasilla 2001) is an approach to making the Web's information accessible to intelligent agents that does not require them to understand natural language. Instead, the Semantic Web encodes the content of pages in a machine-readable format and explicitly links this content to machine-understandable semantics in the form of ontologies.

Agents on the Semantic Web will need to plan how to achieve their goals and must use heterogeneous web resources in order to make their decisions. However, most planning methodologies assume that the planner has complete knowledge about the state of the world. On the Web, this is simply impossible: the Web is too large and changes too quickly for any agent to reasonably assume that it has complete knowledge. However, with an open-world, an agent may spend an unbounded amount of time attempting to find an answer to a query when none exists. Golden, Etzioni, and Weld (1994) have proposed that local closed world information (LCW) can be used to handle such problems in incomplete information environments. However, this approach has never been directly applied to the Semantic Web.

In this paper, we propose extensions to two Semantic Web languages that allow Web documents and other resources to express LCW information. We then describe the design of an agent that can integrate Semantic Web information and use it in its planning process.

2 Background

This work builds on research in the Semantic Web and research into the use of LCW information. We will now briefly discuss each of these areas.

2.1 The Semantic Web

The goal of the Semantic Web is to automate machine processing of web documents by making their meanings explicit (Berners-Lee, Hendler, and Lasilla 2001). Semantic web languages do this by allowing users to create ontologies, which specify standard terms and machine-readable definitions. Semantic web documents then *commit* to one or more ontologies, thus stating which sets of definitions are applicable. Over the last ten years, knowledge representation researchers have studied the use of ontologies for sharing and reusing knowledge (Gruber 1993, Guarino 1998, Noy and Hafner 1997). Although there is some disagreement as to what comprises an ontology, most ontologies include a taxonomy of terms (e.g., a Car is a Vehicle), and many ontology languages allow additional definitions using some form of a logic. Guarino (1998) has defined an ontology as "a logical theory that accounts for the intended meaning of a formal vocabulary." A common feature in ontology languages is the ability to extend preexisting ontologies. Thus, users can gain the interoperability benefits of sharing terminology where possible, but can also customize ontologies to include domain specific information.

Although there are many Semantic Web languages, this paper will focus on two with very different features. The first is DAML+OIL (Hendler and McGuinness 2000), a language developed by an international committee of researchers interested in the Semantic Web. DAML+OIL has its logical basis in description logics and its syntactic basis is RDF. Description logic systems accept complex expressions as class definitions and provide the ability to

determine subsumption (that is, which classes are necessarily subsets of other classes). Typically, the features of such languages are chosen so that subsumption can be performed efficiently. Class expressions in DAML+OIL can include subclass relationships, boolean combinations of class expressions, and restrictions on properties that a class may have. These property restrictions include minimum and maximum cardinality of a property, restriction of values to a specific class, and specific values that the property must have. The complete description of the language is available on the Web (van Harmelen, Patel-Schneider, and Horrocks 2001).

This paper will also discuss the SHOE language (Luke et al. 1997, Heflin 2001). Like DAML+OIL, SHOE has ontologies which provide definitions of classes and properties (called categories and relations in SHOE). However, SHOE is not based on description logic, instead it is based on datalog, a data model commonly used for deductive databases. The syntax of datalog is basically Prolog without function symbols, but unlike Prolog, no control flow is implied by the ordering of statements and atoms. SHOE does not have as rich expressions for defining classes as DAML+OIL, but does have the ability to express arbitrary Horn clauses, something that DAML+OIL lacks. SHOE data consists of instances that can be found in web documents. These instances commit to one or more SHOE ontologies. The data can specify categories (classes) of which instances are members and relationships between instances (predicates). The complete specification of the language is available on the Web (Luke and Heflin 2000). In this paper, our SHOE examples will use SHOE's XML syntax.

2.2 Local Closed World Information

The closed-world assumption (CWA) is used in the semantics of programming languages like Prolog and most databases. It basically states that if a fact cannot be proven to be true, then the fact is assumed to be false. This assumption is useful in that it allows additional inferences to be drawn from the absence of information. However, this assumption is often inappropriate because knowledge may be incomplete. Local closed-world (LCW) information is an approach to this problem in which closed-world information can be obtained on subsets of the information that are known to be complete, while still allowing other information to be treated as unknown (Golden, Etzioni and Weld 1994). LCW information is given as meta-level sentences of the form LCW(Φ). The semantics of such a sentence is that for all variable substitutions θ, if the ground sentence $\Phi\theta$ is true in the world then $\Phi\theta$ is represented in the agent's knowledge base. Any matching ground sentence that is not in the knowledge base is known to be false.

Golden, Etzioni and Weld originally developed LCW in the context of agent planning, and used it to describe the effects of sensing operations that return exhaustive information. Levy (1996) extended this formalism to deal with obtaining complete answers from partial databases, that is, databases that have incomplete information. Various work in information integration has followed, with an emphasis on using LCW to generate efficient information gathering plans (Friedman and Weld 1997; Duschka 1997; Lambrecht, Kambhampati, and Gnanaprakasam 1999). However, this work assumes a priori knowledge of the local completeness information for each information source. These systems typically have a small number of predetermined information sources, and in such a system, this information could be provided by an administrator whenever a new source was added. However, if this work is to be applied to the Semantic Web, then one must realize that there are potentially millions of information sources, since each web page could be considered a data source. In the next section, we will discuss how Semantic Web languages can be extended to allow web resources to provide LCW information regarding their contents.

3 LCW on the Semantic Web

The closed-world assumption is inappropriate for the Semantic Web due to its size and rate of change. Since the Web, is so large, no single agent could expect to have complete knowledge of its contents, and thus an assumption that any unknown facts must be false will often be mistaken. However, if an open-world is assumed, then an agent's search is unbounded, because if it has not found an answer, it has no way of knowing if the answer may be available elsewhere if it simply continues its search. In this section we propose how two Semantic Web languages can be extended to use LCW information.

3.1 Adding LCW to DAML+OIL

Since all instance information in DAML+OIL is expressed using RDF, all DAML+OIL data either states that an instance is a member of a class or that an object has a specific property value. A natural LCW extension to DAML+OIL would be to allow DAML+OIL document to say that it has complete information on members of a particular class or on the properties of a specific object. We will call our extended version of the language DAML-LCW

We propose that a document can use a new property lcw:hasLcw to state that it has complete information on some subset of information. This property is in a new namespace identified by the lcw prefix,[1] and has rdf:Resource in its domain and daml:Class in its range. As such, it can be applied to any resource. Typically it would

[1] The namespace identifier for this namespace has not been chosen yet.

be used by a DAML+OIL document to describe LCW information about its own contents, but by applying the property to another resource, any document can provide LCW information about any other document.

The example below shows a DAML+OIL LCW statement which means that the document that contains it has complete information on all instances of the class http://www.faa.org/ont#UsCommercialFlights.

```
<rdf:Description about="">
  <lcw:hasLcw>
    <daml:Class rdf:ID=
   "http://www.faa.org/ont#UsCmrclFlights"/>
  </lcw:hasLcw>
</rdf:Description>
```

This statement is equivalent to LCW(UsCommercialFlights(*x*)).

Due to DAML+OIL's features for composing complex class expressions, even more specific LCW statements could be made. For example, it would be possible to state local completeness for all flights that have a city in the U.S. as their destination.

```
<lcw:hasLcw>
  <daml:Class>
  <daml:intersectionOf rdf:parseType=
       "daml:collection">
    <daml:Class rdf:about=
       "http://www.faa.org/ont#Flight"/>
    <daml:Restriction>
      <daml:onProperty rdf:resource=
       "http://www.faa.org/ont#destination"/>
      <daml:toClass rdf:resource=
       "http://www.faa.org/ont#UsCity"/>
    </daml:Restriction>
  </daml:intersectionOf>
  </daml:Class>
</lcw:hasLcw>
```

Note that this example is equivalent to LCW(Flight(*x*) ^ destination(*x*,*y*) ^ UsCity(*y*)).

What about LCW statements that do not use classes? For example, how can DAML+OIL be extended to represent LCW(prop(*x*,c))? In this case, we can apply LCW to the class of things with a specific hasValue restriction.

```
<lcw:hasLcw>
    <daml:Restriction>
      <daml:onProperty rdf:resource="prop"/>
      <daml:hasValue rdf:resource="c"/>
    </daml:Restriction>
</lcw:hasLcw>
```

However, it is unclear how LCW(prop(c,*x*)) could be represented in DAML+OIL. The language only provides limited facilities for describing properties, and as such, there is no natural way to extend the language to state that there is complete information on an object's values for a specific property.

The semantics of these DAML+OIL expressions is based on the original semantics of LCW sentences by Golden, Etzioni, and Weld. Any DAML+OIL expression can be rewritten as an equivalent first-order logic expression, thus obtaining a standard LCW statement. However, on the Semantic Web, we must describe what it means to have an LCW sentences about a particular resource. Here, the resource takes the role of the knowledge base. A sentence of the form LCW(Φ) means that for all variable substitutions θ, if the ground sentence Φθ is true in the world then Φθ is represented in the resource described by the sentence. Any matching ground sentence that is not represented by the resource is known to be false.

Interestingly, DAML-LCW is no more expressive than DAML+OIL. Using standard DAML+OIL constructs, one can express the same semantics as the hasLcw statement. Consider a document that has some set of resources *R*, whose elements are r_1, r_2, …, r_n all of which have the same class *C* as their type. DAML-LCW can be used to express that the document has LCW information on class C, thus stating that any resources not in R are not members of class C. We can express this same information using the daml:oneOf, daml:complementOf, and daml:disjointWith features of DAML+OIL. First you use daml:oneOf to construct the class of objects in R. Then you take the daml:complementOf this result to represent the class of objects that are not in R. Finally, you say that this class is daml:disjointWith class C. This DAML+OIL syntax for this is shown below:

```
<daml:Class rdf:about="C">
  <daml:disjointWith >
    <daml:Class>
      <daml:complementOf>
        <daml:Class>
          <daml:oneOf rdf:parseType="daml:collection">
            <daml:Thing rdf:resource="r1">
            <daml:Thing rdf:resource="r2">
            …
            <daml:Thing rdf:resource="rn">
          </daml:oneOf>
        </daml:Class>
      </daml:complementOf>
    </daml:Class>
  </daml:disjointWith>
</daml:Class>
```

This means that complete reasoners for DAML+OIL are also complete reasoners for DAML-LCW. One naïve algorithm is to use a preprocessing step that converts all lcw:hasLcw statements to the form described above, and then run the ordinary inferential process.

Of course, this raises the question whether daml:hasLcw is really necessary. We argue that although it is basically syntactic sugar, that it greatly simplifies the

process of expressing LCW information. Without it, all members of a locally closed class would have to be explicitly listed twice, once to indicate they are a member of the class, and once to define the complement of that set of resources. Furthermore, document maintenance would become much more difficult, because any changes to the classifications of resources would need to be made in both places.

3.2 Adding LCW to SHOE

LCW can be added to SHOE more naturally since its data model is more similar to that used by the information integration approaches that use LCW. In the information integration literature, LCW is used with datalog programs to develop efficient queries for a set of distributed information sources. As mentioned earlier, one of SHOE's chief differences from DAML+OIL is that it is based on datalog.

We can add LCW information to SHOE by introducing a lcw element to instances. That is, a particular SHOE instance can claim to have complete knowledge over some set of information. As with the formulation by Etzioni, Golden, and Weld (1997), we will restrict SHOE LCW sentences to positive conjunctions. In SHOE syntax, this essentially means the child elements of the lcw element are an arbitrary number of category and relation elements. We call the language with this construct SHOE-LCW.

In the previous section, we provided DAML+OIL examples of different LCW statements. The example for LCW(Flight(*x*) ^ destination(*x*,*y*) ^ UsCity(*y*)) can be represented in SHOE-LCW as follows:

```
<lcw>
  <category name="faa.Flight"
            usage="var" for="x" />
  <relation name="faa.destination">
    <arg pos="1" usage="var" value="x" />
    <arg pos="2" value="var" value="y" />
  </relation>
  <category name="faa.UsCity"
             usage="var" for="y" />
</lcw>
```

The use of relations allow SHOE to naturally capture statements such as LCW(prop(*x*,c)) and LCW(prop(c,*x*)). In each case, the lcw element contains a single relation element with one argument that is a constant and one argument that is a variable.

The semantics of SHOE-LCW is based on the semantics of SHOE (Heflin 2001), with one modification to handle the lcw element. This element can be expressed as a standard LCW statement by representing each category as a unary predicate, and representing each n-ary relation as an n-ary predicate. The conjunction of these sentences forms the LCW sentence. The semantics of this sentence are similar to those described in the previous section. The instance element in SHOE represents a web page whose content is described by the SHOE tags. Typically this instance is the page in which the tags occur, but may be another web resource. Thus, SHOE-LCW has the same flexibility to describe different resources as DAML-LCW. However, unlike DAML+OIL, there is no way to express the SHOE-LCW semantics without the language extension. This is because SHOE does not have the capability to express complements and disjointness.

Note that LCW adds implicit negation to SHOE, introducing the possibility of logical inconsistency.[2] That is, if one source claims to have LCW information about a relation, and another source contains an instance of the relation that is not in the former, then the two sources contradict each other. Handling inconsistency on the Semantic Web is still an unsolved problem, so we will assume that information sources only state LCW information when they actually have local completeness information, thereby allowing us to ignore any resources that contradict it.

4 LCW for Agent Planning

In this section we are going to explain how the LCW statements added to the SHOE language can be used for agent planning on the Semantic Web. First we are going to describe the planning formalism known as Ordered Task Decomposition (OTD) (Nau et al, 1999). We have chosen to use OTD planning for two main reasons: first, HTN planning, of which OTD is an special form, has been shown to be strictly more expressive than STRIPS planning, of which partial-order planning is an special form (Erol, Hendler, and Nau 1994). Second, hierarchical task decomposition has been shown to be useful for many real-world domains (Nau et al. 1998).

4.1 Ordered Task Decomposition

An ***HTN*** (hierarchical task network) is a set of tasks and their ordering relations, denoted as $N=(\{t_1,\ldots,t_m\},<)$ $(m\geq 0)$, where < is a binary relation expressing temporal constraints between tasks. Decomposable tasks are called ***compound***, while non-decomposable tasks are called ***primitive***.

A ***domain theory*** consists of methods and operators for generating plans. A ***method*** is an expression of the form *M*=(*h*,*P*,*ST*), where *h* (the method's ***head***) is a compound task, *P* is a set of ***preconditions***, and *ST* is the set of *M*'s (children) ***subtasks***. *M* is ***applicable*** to a task *t*, relative to a ***state*** *S* (a set of ground atoms), iff matches(*h*,*t*,*S*) (i.e., *h* and *t* have the same predicate and arity, and a consistent set of bindings Θ exists that maps variables to values such that all terms in *h* match their corresponding ground terms in *t*) and the preconditions *P*

[2] DAML+OIL already possesses features that can lead to such inconsistencies, but as yet does not suggest how they should be resolved.

are ***satisfied*** in *S* (i.e., there exists a consistent extension of Θ, named Θ', such that $\forall\ p \in P\ \{p\Theta' \in S\}$), in which case $M(t,S)=ST\ \Theta'$.

An ***operator*** is an expression of the form *O=(h,P,aL,dL)*, where *h* (the operator's ***head***) is a primitive task, *P* is a set of ***preconditions***, and *aL* and *dL* are the so-called ***add-*** and ***delete-lists***. These lists define how the operator's application transforms the current state *S*: every element in the add-list is added to *S* and every element in the delete-list is removed from *S*. An operator *O* is ***applicable*** to a task *t*, relative to a state *S*, iff matches(*h,t,S*) and the preconditions *P* are ***satisfied*** in *S*.

A ***planning problem*** is a triple (*T,S,D*), where *T* is a set of tasks, *S* is a state, and *D* is a domain theory. A ***plan*** is the collection of primitive tasks obtained by decomposing all compound tasks in a planning problem (*T,S,D*).

At any point during the *Ordered Task Decomposition process* (OTD), a task list *T'* is being refined relative to a state *S* and a domain theory *D*. Initially, *T'* is the set of tasks *T* in the planning problem (*T,S,D*). In an ordered task decomposition process, the tasks must be totally ordered (i.e., the < relation on HTNs is a total order). During the OTD process the partial solution plan *p* being derived (i.e., the primitive tasks in *T'*) is maintained. Initially *p* is empty. The OTD process selects the first task *t* in *T'* and continues as follows:

- If *t* is primitive and has an applicable operator *O*, then *O* is applied to *t*, *S* is updated accordingly, *t* is removed from *T'* and added to the end of *p*.
- Else if *t* is compound and has an applicable method *M* (that has not yet been applied to *t*), then *M* is applied, which replaces *t* in *T'* with *M*'s subtasks.
- Else if *T'* is not empty, then backtracking occurs.
- Else the process fails.

The OTD process terminates when *T'* is empty, in which case *p* is the solution, or when trying to backtrack on a compound task *t* whose applicable methods have been exhausted. The OTD process was first implemented in the SHOP planning system (Nau et al., 1999). A variant was created that relaxes the condition requiring the tasks to be totally order (Nau et al., 2001) but for the sake of simplicity we'll concentrate on the original assumption that the tasks are totally ordered.

4.2 LCW Statements in Planning

LCW statements are meta-knowledge about the available facts. There are two sources for LCW statements during planning:

- **LCW information is provided explicitly**. Explicit LCW information could be part of the agent's background knowledge or may be provided by the information sources. The later means that the information sources being accessed know in advance that the information is locally closed. As an example, American Airlines has complete information about all American Airlines flights.

- **LCW is inferred as a result of an action**. This means that the execution an action yields local closed-world information. As an example, the UNIX command *ls*, when executed in a directory */dir*, yields complete information about the files contained and not contained in */dir*.

The first source of LCW information has been explored in work on information integration, where the information is used to generate efficient information gathering plans (Levy 1996, Friedman and Weld 1997, Duschka 1997). In our framework, this information is provided by the LCW statements in the SHOE or the DAML extensions discussed in the previous sections. The second source of LCW was proposed in (Golden, Etzioni, and Weld 1994). It reflects the fact that it is typically assumed that all knowledge about changes in the world is modeled in actions known by the planner.

To include these two sources of planning within OTD, we need to (1) cope with the problem of distributed state information (classical planners assume a centralized state that contains all known facts), (2) extend the way methods and (3) operators are used.

4.3 OTD Planning with the Semantic Web

During the OTD decomposition process queries to external information sources need to be performed to evaluate if the preconditions can be satisfied and to take into account LCW statements that had been gathered so far. To handle this situation we created two additional entities external to the OTD Plan Generator: The Knowledge Base (KB) and the Semantic Web Mediator (see Figure 1). The former maintains known facts and LCW statements and the latter mediates between the OTD Plan Generator and the external information sources by accessing and interpreting relevant Semantic Web documents.

The Semantic Web Mediator is based on the concept of mediators proposed by Wiederhold (1992); it is a system that is capable of integrating multiple sources in order to answer questions for another system. Its main function is to evaluate the OTD Plan Generator's preconditions by accessing Semantic Web resources. To accomplish this task the Semantic Web Mediator uses and maintains information about remote sites available, access information, and known ontologies in the KB.

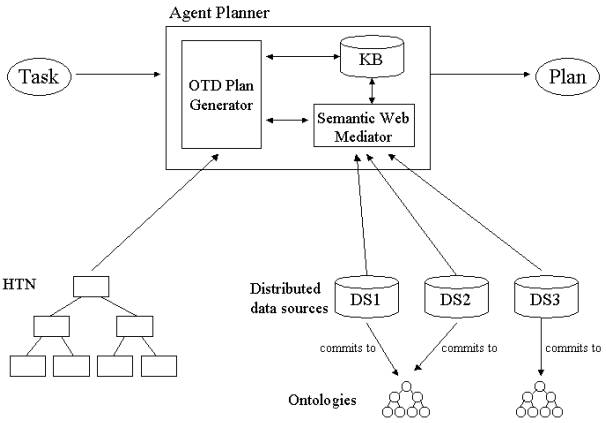

Figure 1

The KB maintains three forms of knowledge:

- Facts that has been gathered so far either through actions of the OTD Plan Generator or through remote access by the Semantic Web Mediator
- LCW statements yielded by the OTD Plan Generator or collected by the Semantic Web Mediator
- Summary of information source contents

It is important to notice that changes resulting from actions taken by the OTD Plan Generator are made in the KB rather than in the actual sources. Consider an action to reserve a seat in a flight that the mediator discovered was free. In the KB we keep track of this action as if the seat was reserved without actually going to the information source and reserving it. We choose this approach because backtracking may occur if we are not be able to satisfy the conditions for other actions later on. Such situations would require the costly process of reaccessing sources in order to tell them to undo previous actions. Thus, we continue planning and once we complete the plan we perform the execution.

For this approach to work we are making the assumption that the content of the information sources (particularly with regards to LCW information) does not change during the planning time. This is a typical assumption made by other systems planning with external information sources (e.g., (Golden, Etzioni, and Weld 1994)) with the rationale being that the small amount of time taken to complete the plan makes it unlikely that this information would change. However, we do acknowledge that in the context of the Semantic Web the validity of this assumption may be questionable and we are currently investigating solutions to the problem.

4.4 Precondition Evaluation

When a precondition *p* is evaluated, four steps need to be performed:

1. Determine if *p* can be satisfied or not by accessing the KB's facts
2. If *p* can be satisfied, the variable bindings satisfying *p* are returned.
3. If *p* cannot be satisfied, the KB's LCW statements are accessed to determine if complete information about p is contained. If this is the case, p is false
4. If p cannot be satisfied and there is no complete information about p in the KB's LCW statements, p is unknown. In this case, *p* is passed to the Semantic Web Mediator.

The first two steps mimic the standard precondition evaluation; in OTD the current state is locally maintained and to determine if the preconditions are satisfied or not, one looks if p is matched in the current state. The third and fourth steps are necessary to handle LCW information. An example of such a situation can be illustrated with the following method, which is the knowledge unit describing the conditions (called *preconditions*) under which a *task* can be decomposed into *subtasks*:

Task:
Get a direct flight Ticket from *start* to *destination* at *date*
Preconditions:
1. Airline(*aline*)
2. DirectFlight(*f*, a*line*, *start*, *destination*)
3. SeatFree(*f, date, s)*
Subtasks:
1. Buy ticket for *s* on *f* at *date*
Orderings:
{}

where *start*, *destination*, *aline*, *f*, *date*, and *s* are variables.

If the system is trying to accomplish the task: Get a direct flight Ticket from *Atlanta* to *Allentown*, and when solving the first precondition, *aline* is instantiated to American Airlines, SHOE will also indicate that local closed world information has been yielded (American Airlines is the only carrier that offers direct flights between Atlanta and Allentown, PA). The LCW statement has the form LCW(DirectFlight(*f*, *aline*, Atlanta, Allentown)), indicating that we have complete information about the direct flights from Atlanta to Allentown. Thus, if no seats were available in any of the flights between these two locations (preconditions 3 and 4), there is no need to search for another airline and check direct flights (preconditions 1 and 2).

The operators, which are the knowledge units describing actions changing the world, may yield or remove LCW information. As an example of an operator yielding LCW information, consider the following operator that uses the same variable names as before:

Task:

Get all free seats available in *f* at *date*

Preconditions:

1. Flight(*f*, a*line*, *start*, *destination*)

Add:

1. ∀*s* SeatFree(*f, date, s)*

Delete:

()

This operators collects all free seats available in flight *f* at *date*. Thus we yield complete information since any seat that was not collected must be occupied. The corresponding statement that is added to KB is: LCW(SeatFree(*355, 1/2/2002, s*) assuming that the operator was executed with the bindings: *f* → *355* and *date* → *1/2/2002*.

4.5 Preconditions and the Semantic Web Mediator

If p is unknown, the OTD Plan generator queries the Semantic Web Mediator for p. This subsytem executes the following steps:

1. It determines an information source to access
2. It accesses the source and parses the SHOE associated with it
3. If the *p* can be satisfied from the KB, the source, and associated ontologies, the variable bindings satisfying *p* are returned.
4. If *p* cannot be satisfied, the source's and the KB's LCW statements are accessed to determine if complete information about p is contained. If this is the case, p is false
5. Otherwise, the system chooses another source and repeats the process.

Any information that is gathered by the Semantic Web Mediator while performing these steps is passed to the KB. Notice that the fifth step implies that if LCW information is not available, the search may be unbounded. In practice, resource-bounded constraints such as time limits or maximum number of accesses may be used to terminate the search.

5 Related Work

XII is the planner that first introduced LCW to reduce the planning time when dealing with external information sources (Golden, Etzioni, and Weld 1994). This and other works show that LCW information can dramatically reduce the planning time by avoiding redundant access to external remote information sources. XII follows the partial-order planning paradigm instead of the ordered task decomposition paradigm discussed in this paper. As a result of this difference XII uses a very different mechanism to handle LCW information.

Another important difference is that in XII, only LCW information yielded by actions is accounted for whereas in our work we also allow LCW information to be defined as meta-knowledge. This difference is crucial to reflect the fact that no system has centralized knowledge about all possible inferences. In the semantic web, inference information is distributed in the ontologies of the remote information sources.

Our work is also similar to the Ashop planner (Dix et al 2002) in that OTD was extended for accessing external information sources for a multi-agent system called IMPACT. In our work, however, we extend OTD to take advantage of local closed-world information in the context of the semantic web.

6 Conclusions and Future Work

In this paper we discussed how many Semantic Web applications will need access to closed-world information, but the nature of the Semantic Web makes it naturally an open-world. To overcome this problem, we presented extensions to the DAML+OIL and the SHOE languages for representing LCW statements. We saw that due to limitations in the facilities for describing properties in DAML+OIL, LCW statements like LCW(prop(c,x)) are difficult to represent. However, a limited form of LCW can be expressed using DAML+OIL's existing features, and our LCW extension merely makes it more convenient to create and maintain documents that need to express LCW information. We also discuss how LCW statements can be added to SHOE more naturally due to its similarity to information integration approaches to using LCW.

We also described an agent that takes advantages of these LCW statements. We discussed two sources for the LCW statements during the agent's OTD process: LCW information is provided explicitly by the information sources using the extensions to the Semantic Web languages discussed and LCW information is inferred as a result of an action. We saw that the Semantic Web Mediator is a key component of the agent's design, whose main function is to evaluate the OTD Plan Generator's preconditions by accessing remote information sites.

We are currently implementing an agent following this design and studying how to deal with changing information during the planning process. In future work, we plan to study how to deal with inconsistencies resulting from LCW information adding implicit negation to SHOE.

References

Berners-Lee, T.; Hendler, J.; and Lasilla, O. 2001. The Semantic Web. *Scientific American*, May 2001.

Dix, J.; Muñoz-Avila, H.; Nau, D. S.; Zhang, L. (2002) IMPACTing SHOP: Putting an AI Planner into a Multi-Agent Environment. To appear in *Annals of Mathematics and Artificial Intelligence*.

Duschka, O. 1997. Query Optimization using Local Completeness. In *Proc. of AAAI-97*, 249-255. Menlo Park, Calif.: AAAI Press.

Erol, K.; Hendler, J. and Nau, D.S. 1994. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proc. of AIPS-94*.

Etzioni, O.; Golden, K; and Weld, D. 1997. Sound and Efficient Closed-World Reasoning for Planning. *Artificial Intelligence*, 89:113-148.

Friedman, M. and Weld, D. 1997. Efficiently Executing Information-Gathering Plans. In *Proc. of IJCAI-97*.

Golden, K.; Etzioni O.; and Weld, D. 1994. Omnipresence Without Omniscience: Efficient Sensor Managment for Planning. *In proc. of AAAI-94*.

Gruber, T. 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199-220.

Guarino, N. 1998. Formal Ontology and Information Systems. In *Proc. of Formal Ontology and Information Systems*, Trento, Italy. IOS Press.

Heflin, J. 2001. *Towards the Semantic Web: Knowledge Representation in a Dynamic, Distributed Environment.* Ph.D. Thesis, University of Maryland, College Park.

Hendler, J.; and McGuinness, D. 2000. The DARPA Agent Markup Language. *IEEE Intelligent Systems*, 15(6):72-73.

Lambrecht, E.; Kambhampati, S.; and Gnanaprakasam, S. 1999. Optimizing Recursive Information Gathering Plans. In *Proc. of IJCAI-1999*. 1204-1210.

Levy, A. 1996. Obtaining Complete Answers from Incomplete Databases. In *Proc. of the 22nd VLDB Conference.*

Luke, S. and Heflin, J. 2000. *SHOE 1.01 Proposed Specification.* At: http://www.cs.umd.edu/projects/plus/SHOE/spec.html

Luke, S.; Spector, L.; Rager, D.; and Hendler, J. 1997. Ontology-Based Web Agents. In *Proc. of First International Conf. on Autonomous Agents*, 59-66. New York, NY: Association of Computing Machinery.

Nau, D.S.; Smith, S.J.J; and Erol, K; 1998. "Control Strategies in HTN Planning: Theory versus Practice." In *AAAI-98/IAAI-98*.

Nau, D.; Cao, Y; Lotem, A.; and Muñoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner. In *Proceedings of IJCAI-99.*

Nau, D.; Muñoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-Order Planning with Partially Ordered Subtasks. In *Proceedings of IJCAI-2001*.

Noy, N. and Hafner, C. 1997. The State of the Art in Ontology Design. *AI Magazine*, 18(3):53-74.

van Harmelen, F., Patel-Schneider, P., and Horrocks, I., eds. 2001. *Reference Description of the DAML+OIL (March 2001) Ontology Markup Language*. At: http://www.daml.org/2001/03/reference.html

Wiederhold, G. 1992. Mediators in the Architecture of Future Information Systems. *IEEE Computer*. 25(3):38-49.