Time in OWL-S

Feng Pan and Jerry R. Hobbs

University of Southern California / Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
{pan, hobbs}@isi.edu

Abstract

To provide support for describing temporal properties for OWL-S (formerly DAML-S), in this paper we first introduce an "entry" sub-ontology of time, which is much simpler than the full ontology of DAML-Time[1] (Hobbs 2002) and provides the basic temporal concepts and relations that most simple applications would need, i.e., a vocabulary for expressing facts about topological relations among instants, intervals, and events, together with information about durations, and about dates and times. Then we demonstrate in detail, using the Congo.com and Bravo Air examples, how this entry sub-ontology of time can be used to support OWL-S, including use cases for defining input parameters and (conditional) output parameters. Segments of OWL encoding of the definitions are shown.

Introduction

OWL-S (formerly DAML-S) is an OWL-based Web service ontology that supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-intepretable form (The DAML Services Coalition 2003). The latest 0.9 draft release is expected to be the last one built on DAML+OIL (Connolly et al. 2001), and the later releases will be based on OWL (Dean et al. 2003).

Temporal information is so common that it's hard to find a real world Web service without it. For example, whenever you place an online order, the order date is always part of your order. When you reserve a car at a car rental site, you have to specify the dates you need it. In response to this need, in conjunction with OWL-S, a temporal ontology, DAML-Time (Hobbs 2002), has been developed for describing the temporal content of Web pages and the temporal properties of Web services. Its development is being informed by temporal ontologies developed at a number of sites and is intended to capture the essential features of all of them and make them and their associated resources easily available to a large group of Web developers and users.

In Section 2 we introduce an entry sub-ontology[2] of time in OWL, which is much simpler than the full DAML-Time and provides most of the basic temporal concepts and relations that most simple applications would need, i.e., a vocabulary for expressing facts about topological relations among instants, intervals, and events, together with information about durations, and about dates and times. The abstract characterization of the concepts and relations are expressed in first-order predicate calculus, and a subset of the ontology has been encoded in OWL. We also describe the time zone data for the entire world that we have developed in OWL. In Section 3 we use the Congo.com and Bravo Air examples to demonstrate in details how our entry sub-ontology of time can be used to support OWL-S. Finally, we discuss future work.

An Entry Sub-Ontology of Time

DAML-Time is an abstract ontology of time intended to be a complete specification of a theory of time as required for Semantic Web (Berners-Lee et al. 2001) applications. Included in it is a rich collection of axioms that tightly constrain the interpretation of the predicates and functions of the theory. But for most simple applications this is far more than is required, and its complexity constitutes a barrier to the use of the ontology. The purpose of this entry sub-ontology of time is to provide quick access to the essential vocabulary in OWL for the basic temporal concepts and relations. The OWL encoding of the entire entry sub-ontology can be found online[3].

This entry sub-ontology of time covers topological relations among instants and intervals and instant-like and interval-like events (called instant events and interval events), such as "before" and "overlaps". It includes measures for durations so that we can say a meeting will last 1 hour and 30 minutes, and it also includes clock and calendar terms so that we can say a meeting starts at 3:00pm PST on Monday, October 20, 2003.

A simple use case[4] in OWL is presented for the sub-ontology: "Suppose someone has a telecon scheduled for 6:00pm EST on November 5, 2003. You would like to make an appointment with him for 2:00pm PST on the

[1] DAML-Time Homepage: http://www.cs.rochester.edu/~ferguson/daml/

[2] http://www.isi.edu/~pan/damltime/time-entry-documentation.txt

[3] http://www.isi.edu/~pan/damltime/time-entry.owl

[4] http://www.isi.edu/~pan/damltime/time-entry-case1.owl

same day, and expect the meeting to last 45 minutes. Will there be an overlap?" In this use case we specify the facts about the telecon and the meeting using our ontology in OWL that will allow a temporal reasoner to determine whether there is a conflict. We have been in contact with several sites about linking our ontology with their temporal reasoners.

Topological Temporal Relations

The most basic temporal concepts in the sub-ontology are Instant, Interval, Instant Event, and Interval Event. Instants are, intuitively, point-like in that they have no interior points, and intervals are, intuitively, things with extent. Instant events are events that are instantaneous, such as the occurrence of a car accident or the arrival of a package, and interval events are events that span some time interval, for example, a meeting from 2pm to 3pm.

Besides these four basic temporal concepts, there are five other more general temporal concepts/classes: Temporal Thing, Temporal Entity, Instant Thing, Interval Thing, and Event. The subclass hierarchy of these temporal concepts/classes is shown in the Figure 1. For example, Instant Thing has two subclasses: Instant and Instant Event.

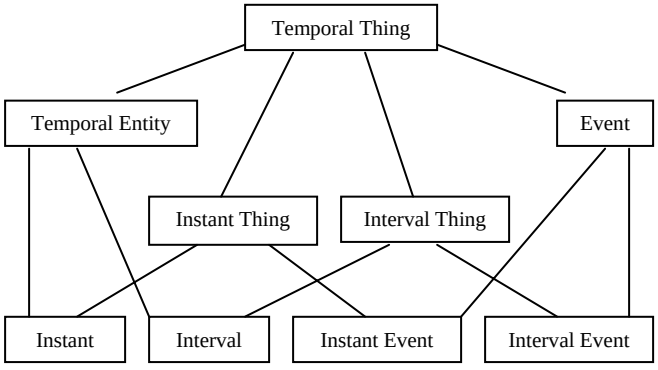

Figure1: Subclass hierarchy of temporal concepts

Their axiom definitions are straightforward. For example, the axiom to define that Temporal Entity has only two subclasses, Instant and Interval, are:

```
TemporalEntity(T) <--> Instant(T) v Interval(T)[5]
```

The above axioms can be expressed in OWL as:

```
<owl:Class rdf:ID="Instant">
  <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
</owl:Class>

<owl:Class rdf:ID="Interval">
  <rdfs:subClassOf rdf:resource="#TemporalEntity"/>
</owl:Class>

<owl:Class rdf:ID="TemporalEntity">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Instant" />
    <owl:Class rdf:about="#Interval" />
  </owl:unionOf>
</owl:Class>
```

Other temporal concepts and their relationships can be defined similarly.

`begins` and `ends` are relations between instant things and temporal things, and the beginnings and ends of temporal entities, if they exist, are unique. In some approach to infinite intervals, a positively infinite interval has no end, and a negatively infinite interval has no beginning. Hence, we use the relations `begins` and `ends` in the ontology, rather than defining functions `beginning-of` and `end-of`, since the functions would not be total. The axioms that characterize `begins`, for example, are:

```
begins(t,T)-->InstantThing(t) & TemporalThing(T)

TemporalEntity(T) & begins(t1,T) & begins(t2,T)
--> t1=t2
```

These properties can be expressed in OWL as:

```
<owl:ObjectProperty rdf:ID="begins">
  <rdf:type rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#TemporalThing" />
  <rdfs:range  rdf:resource="#InstantThing" />
</owl:ObjectProperty>
```

`inside` is a relation between an instant thing and an interval thing, and it is not intended to include beginnings and ends of intervals.

We can define a proper-interval as one whose beginning and end are not identical:

```
(∀ T)[ProperInterval(T)
   <--> Interval(T)
        & (∀ t1,t2)[begins(t1,T) & ends(t2,T)
                    --> t1 =/= t2]]
```

A half-infinite or infinite interval, by this definition, is proper. The ontology is silent about whether there are any intervals that are not proper intervals.

There is a `before` relation on temporal things, which gives directionality to time. If a temporal thing `T1` is before another temporal thing `T2`, then the end of `T1` is before the beginning of `T2`. Thus, `before` can be considered to be basic to instant things and derived for interval things:

```
(∀ T1,T2)[before(T1,T2)
   <--> (∃ t1,t2)[ends(t1,T1) & begins(t2,T2)
                  & before(t1,t2)]]
```

The `before` relation is anti-reflexive, anti-symmetric and transitive.

Interval Relations. The relations between intervals defined in Allen's temporal interval calculus (Allen 1984)

[5] A note on notation: conjunction (&) takes precedence over implication (-->) and equivalence (<-->). Formulas are assumed to be universally quantified on the variables appearing in the antecedent of the highest-level implication.

can be defined in a relatively straightforward fashion in terms of `before` and identity on the beginning and end points. The standard interval calculus assumes all intervals are proper, and we do that here too, but we generalize proper intervals to proper interval things.

Axioms are defined for the interval relations: `intEquals`, `intBefore`, `intMeets`, `intOverlaps`, `intStarts`, `intDuring`, `intFinishes`, and their reverse interval relations: `intAfter`, `intMetBy`, `intOverlappedBy`, `intStartedBy`, `intContains`, `intFinishedBy`. For example, the definition of `intEquals` is:

```
(∀ T1,T2)[intEquals(T1,T2)
    <--> [ProperIntervalThing(T1)
          & ProperIntervalThing(T2)
          & (∀ t1)[begins(t1,T1)
                   <--> begins(t1,T2)]
          & (∀ t2)[ends(t2,T1)
                   <--> ends(t2,T2)]]]
```

Duration Description

The duration of an interval (or temporal sequence) can have many different descriptions. An interval can be 1 day 2 hours, or 26 hours, or 1560 minutes, and so on. It is useful to be able to talk about these descriptions in a convenient way as independent objects, and to talk about their equivalences. We do this first in terms of a predicate called "duration-of" that takes eight arguments, one for a temporal thing, and one each for years, months, weeks, days, hours, minutes, and seconds. Then we will define a specific kind of individual called a "duration description", together with a number of functions relating the duration description to the values of each of the eight arguments. Thereby we convert the 8-ary predicate "duration-of" into eight binary relations that are more convenient for description logic-based markup languages, such as OWL. Here is the definition of the duration description:

```
(∀ T,y,m,w,d,h,n,s)[duration-of(T,y,m,w,d,h,n,s)
     <--> (∃ d1)[durationDescriptionOf(d1,T)
                 & DurationDescription(d1)
                 & years-of(d1) = y
                 & months-of(d1) = m
                 & weeks-of(d1) = w
                 & days-of(d1) = d
                 & hours-of(d1) = h
                 & minutes-of(d1) = n
                 & seconds-of(d1) = s]]
```

It can be expressed in OWL as:

```
<owl:Class rdf:ID="DurationDescription">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#years" />
       <owl:maxCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
       </owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
...
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#seconds" />
       <owl:maxCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
       </owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

The axiom saying that instant things have 0 duration is:

```
InstantThing(t) -->
      duration-of(t,0,0,0,0,0,0,0)
```

There are two ways to specify the duration description of a temporal thing. The relation `durationDescriptionOf` uses `DurationDescription` as its range, while `durationDescriptionDataType` uses the XML Schema datatype `duration`[6] as its range:

```
<owl:ObjectProperty
       rdf:ID="durationDescriptionOf">
  <rdfs:domain rdf:resource="#TemporalThing" />
  <rdfs:range  rdf:resource="#DurationDescription" />
</owl:ObjectProperty>

<owl:DatatypeProperty
       rdf:ID="durationDescriptionDataType">
  <rdfs:domain rdf:resource="#TemporalThing" />
  <rdfs:range  rdf:resource="&xsd;duration" />
</owl:DatatypeProperty>
```

Using the XML Schema datatype `duration` is simpler and more standard. But it can't specify weeks, which can be specified by using `DurationDescription`. Using `DurationDescription` also makes it easier to extract values from any field for the later use. The user has the freedom to choose either of these two properties/relations to specify a duration description for a temporal thing.

Time Zones

What hour of the day an instant is in is relative to the time zone. This is also true of minutes, since there are regions in the world, e.g., central Australia, where the hours are not aligned with GMT hours, but are, e.g., offset half an hour. Probably seconds are not relative to the time zone.

Days, weeks, months and years are also relative to the time zone, since, e.g., 2003 began in the Eastern Standard time zone three hours before it began in the Pacific Standard time zone. Thus, predications about all clock and calendar intervals except seconds are relative to a time zone.

We have been referring to time zones, but in fact it is more convenient to work in terms of what we might call the "time standard" that is used in a time zone. That is, it is better to work with the Pacific Standard Time (PST) as a legal entity than with the PST zone as a geographical region. A time standard is a way of computing the time, relative to a world-wide system of computing time. For each time standard, there is a zone, or geographical region, and a time of the year in which it is used for describing local times. Where and when a time standard is used have

[6] http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration

to be axiomatized, and this involves interrelating a time ontology and a geographical ontology. These relations can be quite complex. Only the entities like PST and EDT, the time standards, are part of the time ontology.

If we were to conflate time zones (i.e., geographical regions) and time standards, it would likely result in problems in several situations. For example, the Eastern Standard zone and the Eastern Daylight zone are not identical, since most of Indiana is on Eastern Standard time all year. The state of Arizona and the Navajo Indian Reservation, two overlapping geopolitical regions, have different time standards during the daylight saving times -- one is Pacific and the other is Mountain.

Time standards that seem equivalent, like Eastern Standard and Central Daylight, should be thought of as separate entities. Whereas they function the same in the time ontology, they do not function the same in the ontology that articulates time and geography. For example, it would be false to say those parts of Indiana shift in April from Eastern Standard to Central Daylight time.

Time Zone Data in OWL.

We have developed a time zone resource[7] in OWL for not only the US but also the entire world, including three parts: the time ontology file[8], the US time zone instance file[9], and the world time zone instance file[10].

The time zone ontology links a preliminary geographic ontology with a time ontology. It defines the vocabulary about regions, political regions (countries, states, counties, reservations, and cities), time zones, daylight saving policies, and the relationships between these concepts. Its instances also link to other existing data on the Web, such as Terry Payne's US states instances[11], FIPS 55 county instances[12], and ISO country instances[13].

It can handle all the usual time zone and daylight savings cases. For example, Los Angles uses PST, the time offset from Greenwich Mean Time (GMT) is -8 hours, and it observed daylight savings from April 6 to October 26 in 2003. But it handles unusual cases as well. For example, in Idaho the northern part is in the Pacific zone, the southern part in the Mountain. The city of West Wendover, Nevada is in the Mountain time zone, while the rest of Nevada is in the Pacific.

For the details, see the documentation[14], which includes an outline of the ontology and the anticipated use.

[7] http://www.isi.edu/~pan/timezonehomepage.html
[8] http://www.isi.edu/~pan/damltime/timezone-ont.owl
[9] http://www.isi.edu/~pan/damltime/timezone-us.owl
[10] http://www.isi.edu/~pan/damltime/timezone-world.owl
[11] http://www.daml.ri.cmu.edu/ont/USRegionState.daml
[12] http://www.daml.org/2003/02/fips55/
[13] http://www.daml.org/2001/09/countries/iso
[14] http://www.isi.edu/~pan/damltime/time-zone documentation.txt

Calendar and Clock Units

The aim of this section is to explicate the various standard calendar and clock intervals. A day as a calendar interval begins at and includes midnight and goes until but does not include the next midnight. By contrast, a day as a duration is any interval that is 24 hours in length. This section deals with the day as a calendar interval.

Including the beginning but not the end of a calendar interval in the interval may strike some as arbitrary. But we get a cleaner treatment if, for example, all times of the form 12:xx a.m., including 12:00 a.m. are part of the same hour and day, and all times of the form 10:15:xx, including 10:15:00, are part of the same minute.

For stating general properties about clock intervals, it is useful to have the following predication:

```
clock-int(y,n,u,x)
```

This expression says that `y` is the `n`th clock interval of type `u` in `x`. For example, the proposition `clock-int(10:03,3,*Minute*,[10:00,11:00))` holds. Here `u` can be a member of the set of clock units, that is, one of `*Second*`, `*Minute*`, or `*Hour*`.

In addition, there is a calendar unit function with similar structure:

```
cal-int(y,n,u,x)
```

This says that `y` is the `n`th calendar interval of type `u` in `x`. For example, the proposition `cal-int(12Mar2002,12,*Day*,Mar2002)` holds. Here `u` can be one of the calendar units `*Day*`, `*Week*`, `*Month*`, and `*Year*`.

A distinction is made above between clocks and calendars because they differ in how they number their unit intervals. The first minute of an hour is labeled with 0; for example, the first minute of the hour [10:00,11:00) is 10:00. The first day of a month is labeled with 1; the first day of March is March 1. We number minutes for the number just completed; we number days for the day we are working on. Thus, if the larger unit has N smaller units, the argument `n` in `clock-int` runs from 0 to N-1, whereas in `cal-int` `n` runs from 1 to N. To state properties true of both clock and calendar intervals, we can use the predicate `cal-int` and relate the two notions with the axiom:

```
cal-int(y,n,u,x) <--> clock-int(y,n-1,u,x)
```

In `cal-int(y,n,u,x)` and `clock-int(y,n,u,x)`, `y` is not an arbitrary interval; it has to be a calendar-clock interval which is a subclass of a proper interval:

```
CalendarClockInterval(T) --> ProperInterval(T)
```

Calendar-Clock Description

To express `cal-int(y,n,u,x)` and `clock-int(y,n,u,x)` directly in OWL is inconvenient since `x` is itself a clock or calendar interval that requires description. So we defined a

calendar-clock description in OWL for specifying both calendar and clock information for a calendar-clock interval.

A calendar-clock description has the following properties/fields: unitType, year, month, week, day, dayOfWeek, dayOfYear, hour, minute, second, and time zone. The property unitType specifies the temporal unit type of the calendar-clock description, and its domain is `TemporalUnit`:

```
<owl:Class rdf:ID="TemporalUnit">
  <owl:oneOf rdf:parseType="Collection">
    <TemporalUnit rdf:about="#unitSecond" />
    <TemporalUnit rdf:about="#unitMinute" />
    <TemporalUnit rdf:about="#unitHour" />
    <TemporalUnit rdf:about="#unitDay" />
    <TemporalUnit rdf:about="#unitWeek" />
    <TemporalUnit rdf:about="#unitMonth" />
    <TemporalUnit rdf:about="#unitYear" />
  </owl:oneOf>
</owl:Class>
```

For example, the temporal unit type of 10:30 is minute (unitMinute), and the temporal unit type of March 20, 2003 is day (unitDay). The unit type is required. With a given temporal unit type, all the fields/properties for smaller units will be ignored. For instance, if the temporal unit type is day (unitDay), the values of the field/property hour, minute, and second, if present, will be ignored.

Since calendar-clock description is for describing calendar-clock intervals, we defined a property, called `calendarClockDescriptionOf` with `CalendarClock-Description` as the range, for calendar-clock intervals.

To express `cal-int(12Mar2002,12,*Day*,Mar2002)`, for example, using calendar-clock description, we need an instance of `CalendarClock-Description` that has values only for unitType (unitDay), year (2002), month (3), and day (12). `clock-int(10:03,3,*Minute*,[10:00, 11:00))` can be expressed similarly.

`CalendarClockDescription` and `calendarClock-DescriptionOf` are defined in OWL as:

```
<owl:Class rdf:ID="CalendarClockDescription">
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#unitType" />
       <owl:cardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
       </owl:cardinality>
    </owl:Restriction>
  </owl:subClassOf>
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#year" />
       <owl:maxCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
       </owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
   ...
  <owl:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#timeZone" />
       <owl:maxCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
       </owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>

<owl:ObjectProperty
       rdf:ID="calendarClockDescriptionOf">
  <rdfs:domain
       rdf:resource="#CalendarClockInterval" />
  <rdfs:range
       rdf:resource="#CalendarClockDescription" />
</owl:ObjectProperty>
```

In order to specify that an instant thing is in a calendar-clock interval, an `inCalendarClock` property/relation is defined similarly to `calendarClockDescriptionOf` as follows:

```
<owl:ObjectProperty rdf:ID="inCalendarClock">
  <rdfs:domain rdf:resource="#InstantThing" />
  <rdfs:range
       rdf:resource="#CalendarClockDescription" />
</owl:ObjectProperty>
```

With this `inCalendarClock` relation, we can say that an instant thing is at a specific calendar-clock time. For example, the beginning of a meeting, which is an instant, is at 6:00pm which is actually in a calendar-clock interval of [6:00:00, 6:01:00).

We also defined in OWL two simpler relations, `calendarClockDescriptionDatatype` and `inCalendar-ClockDatatype`. Similar to durations, the only difference between these two relations and the above `calendarClockDescriptionOf` and `inCalendarClock` relations is their ranges: these two simpler relations use the XML Schema datatype `dateTime`[15] as their ranges, while the above uses `CalendarClockDescription`:

```
<owl:DatatypeProperty
       rdf:ID="calendarClockDescriptionDataType">
  <rdfs:domain rdf:resource="#IntervalThing" />
  <rdfs:range  rdf:resource="&xsd;dateTime" />
</owl:DatatypeProperty>

<owl:DatatypeProperty
       rdf:ID="inCalendarClockDataType">
  <rdfs:domain rdf:resource="#InstantThing" />
  <rdfs:range  rdf:resource="&xsd;dateTime" />
</owl:DatatypeProperty>
```

To illustrate more clearly the difference between using `CalendarClockDescription` and using the XML datatype `dateTime`, let's look at a concrete example: an instant, called "instantExample", at 10:30am EST on 01/01/2003 can be expressed using both `inCalendarClockDataType` and `inCalendarClock` in OWL as:

```
<time:Instant  rdf:ID="instantExample">
   <time:inCalendarClock
      rdf:resource="instantExampleDescription" />
   <time:inCalendarClockDataType
      rdf:datatype="&xsd;dateTime">
      2003-01-01T10:30:00-5:00
   </time:inCalendarClockDataType>
</time:Instant>

<time:CalendarClockDescription
      rdf:ID="instantExampleDescription">
```

[15] http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#dateTime

```
    <time:unitType
         rdf:resource="&time;unitMinute" />
    <time:year rdf:datatype="&xsd;gYear">2003
    </time:year>
    <time:month rdf:datatype="&xsd;gMonth">1
    </time:month>
    <time:week
       rdf:datatype="&xsd;nonNegativeInteger">1
    </time:day>
    <time:day rdf:datatype="&xsd;gDay">1
    </time:day>
    <time:dayOfWeekField
       rdf:datatype="&xsd;nonNegativeInteger">3
    </time:day>
    <time:dayOfYearField
       rdf:datatype="&xsd;nonNegativeInteger">1
    </time:day>
    <time:hour
       rdf:datatype="&xsd;nonNegativeInteger">10
    </time:hour>
    <time:minute
       rdf:datatype="&xsd;nonNegativeInteger">30
    </time:minute>
    <time:timeZone rdf:resource="&tz-us;EST" />
  </time:CalendarClockDescription>
```

We can see from this example that it's much simpler to use the XML Schema datatype, `dateTime`. However, the advantage of using `CalendarClockDescription` is that it can express more information than `dateTime`, such as "week", "day of week" and "day of year", so in the above example, we can also know that 01/01/2003 is Wednesday, on the first day of the year, and in the first week of the year. The namespace "tz-us" points to our US time zone data[16]. Moreover, each field of `CalendarClock-Description` is separate so that it's easier to extract the value of some fields for the later use and easier to reason about.

Time in OWL-S

Congo.com and Bravo Air are the two examples being used in the latest OWL-S 0.9 draft release (The DAML Services Coalition 2003). Congo.com is a fictitious book-selling service site, and Bravo Air is a fictitious airline-ticketing service site. We use these two examples to demonstrate in detail how our entry sub-ontology of time can be used to support OWL-S, including use cases for defining input parameters and (conditional) output parameters. Segments of OWL encoding of the definitions are shown.

Use Cases for Input Parameters

In the profile of the Congo.com example (i.e. CongoProfile.owl), for example, our time ontology is currently used for describing the input parameter CreditCardExpirationDate:

```
<profile:input>
  <profile:ParameterDescription
         rdf:ID="CreditCardExpirationDate">
    <profile:parameterName>
         creditCardExpirationDate
    </profile:parameterName>
    <profile:restrictedTo
         rdf:resource="&time;#TemporalEntity"/>
      <profile:refersTo rdf:resource=
        "&congoProcess;#creditCardExpirationDate"/>
  </profile:ParameterDescription>
</profile:input>
```

The namespace "time" points to the location of the current OWL encoding of our entry sub-ontology of time[17]. In this example `Instant`, a subclass of `TemporalEntity`, would be a better class to use than `TemporalEntity` to describe CreditCardExpirationDate, because the expiration date is actually an instant -- the midnight, of the day the credit card expires.

In the Bravo Air example, our time ontology can be used to describe the existing input parameters, DepartureDate and ArrivalDate. We will change this to the more appropriate DepartureTime and ArrivalTime. We can define DepartureTime in the profile of the Bravo Air example (i.e. BravoAirProfile.owl) as:

```
<profile:input>
  <profile:ParameterDescription  rdf:ID="DepartureTime">
    <profile:parameterName>
         DepartureTime</profile:parameterName>
    <profile:restrictedTo
         rdf:resource="&time;#Instant"/>
    <profile:refersTo
         rdf:resource="&ba_process;#outboudDate_In"/>
  </profile:ParameterDescription>
</profile:input>
```

DepartureTime is defined as an Instant. With this definition, as we discussed in the previous calendar-clock description section, an instance of DepartureTime can has either an `inCalendarClockDataType` property/relation pointing to a specific value of XML Schema datatype `dateTime`, say 2003-01-01T10:30:00-5:00, or an `inCalendarClock` object-property/relation pointing to an instance of `CalendarClockDescription` class specifying a specific time, say 10:30am EST on 01/01/2003, Wednesday. It would be the user's decision to define the time in either way based on the trade-offs discussed in last section.

Use Cases for (Conditional) Output Parameters

In fact, there is much more that our time ontology can do to support OWL-S. In the current Congo.com and Bravo Air examples, the time ontology is not used for any output parameters. However, in the real world many service outputs are time-related. For example, in the Congo.com example we can add two outputs that are very common in real world book-selling sites: process time and delivery duration.

Adding a ProcessTime output parameter. ProcessTime is a conditional output parameter that specifies how long before the book will be ready for delivery, say, 24 hours, which depends on whether the book is in stock. In this use case, the process time is returned only if the book is in stock. It can be defined in the process model of the Congo.com example (i.e. CongoProcess.owl) as:

[16] http://www.isi.edu/~pan/damltime/timezone-us.owl

[17] http://www.isi.edu/~pan/damltime/time-entry.owl

```
<owl:Class rdf:ID="ProcessTime">
  <rdfs:subClassOf rdf:resource="&time;#Interval"/>
</owl:Class>

<rdf:Property rdf:ID="fullCongoBuyProcessTime">
  <rdfs:subPropertyOf rdf:resource="&process;#output"/>
  <rdfs:domain rdf:resource="#FullCongoBuy"/>
  <rdfs:range>
    <owl:Class>
       <rdfs:subClassOf rdf:resource=
              "&process;#ConditionalOutput"/>
       <rdfs:subClassOf>
          <owl:Restriction>
            <owl:onProperty rdf:resource=
              "&process;#coCondition"/>
            <owl:allValuesFrom rdf:resource=
              "#BookInStock"/>
          </owl:Restriction>
       </rdfs:subClassOf>
       <rdfs:subClassOf>
         <owl:Restriction>
           <owl:onProperty rdf:resource=
              "&process;#coOutput"/>
           <owl:allValuesFrom rdf:resource=
              "#ProcessTime"/>
         </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
 </rdfs:range>
</rdf:Property>
```

ProcessTime is defined as an `Interval`, rather than a duration. As discussed previously, in our time ontology durations are properties of intervals. Thus to talk about a duration, i.e. a quantity of time, an interval must be defined first. This approach may look roundabout at first glance. However, the process time is not purely a quantity of time; it has a location on the time line. The beginning of the process time is the time the user places the order, and the end of the process time is the time the order is shipped out. An advantage of defining ProcessTime as an interval is that if the relationship among the order time, the shipping time, and the process time is known, any one of them (e.g. the shipping time) can be computed from the other two (e.g. the order time and the process time) by temporal arithmetic.

Adding a DeliveryDuration output parameter. Delivery- Duration is a conditional output parameter that specifies how long it will take for the customer to receive the book safter it is shipped out, which depends on the delivery type the customer selects. As defined in the process model of the Congo.com example (i.e. CongoProcess.owl), the current delivery types are FedExOneDay, FedEx2-3day, UPS, and OrdinaryMail.

To add this output parameter may seem similar to the above ProcessTime example. However, since an instance of Condition is a logical formula that evaluates to true or false (see the comment with the definition of Condition[18]), DeliveryType cannot be directly used as a condition to determine the delivery duration. Thus one property and one condition are defined for each delivery type.

DeliveryDuration is defined with two boundaries: one minDeliveryDuration and one maxDeliveryDuration. For example, an order with the FedEx2-3day delivery type takes 2 to 3 days, so its min delivery duration is 2 days, and its max delivery duration is 3 days. For the delivery duration of the order with FedExOneDay delivery type, the min and max delivery duration will both be 1 day. We can define DeliveryDuration in the process model of the Congo.com example (i.e. CongoProcess.owl) as:

```
<owl:Class rdf:ID="DeliveryDuration">
  <rdfs:subClassOf>
    <owl:Restriction owl:cardinality="1">
      <owl:onProperty
           rdf:resource="#minDeliveryDuration"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction owl:cardinality="1">
      <owl:onProperty
           rdf:resource="#maxDeliveryDuration"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<rdf:Property rdf:ID="minDeliveryDuration">
  <rdfs:domain rdf:resource="#DeliveryDuration"/>
  <rdfs:range rdf:resource="&time;#Interval"/>
</rdf:Property>

<rdf:Property rdf:ID="maxDeliveryDuration">
  <rdfs:domain rdf:resource="#DeliveryDuration"/>
  <rdfs:range rdf:resource="&time;#Interval"/>
</rdf:Property>
```

Both minDeliveryDuration and maxDeliveryDuration are defined as properties of DeliveryDuration. For the same reason discussed for the process time example, both properties use Interval as their ranges. The cardinality of 1 for both properties in the definition of DeliveryDuration indicates that an instance of DeliveryDuration must have one and only one property value for minDeliveryDuration and maxDeliveryDuration respectively.

For example, in order to define delivery duration for FedEx2-3day, we have to first define a condition of FedEx2-3day being selected:

```
<owl:Class rdf:ID="FedEx2-3dayCondition">
  <rdfs:subClassOf rdf:resource="&process;#Condition"/>
</owl:Class>
```

Then we define an output property, called deliverySelect-FedEx2-3day that is conditional on FedEx2-3dayCondition defined above:

```
<rdf:Property rdf:ID="deliverySelectFedEx2-3day">
  <rdfs:subPropertyOf rdf:resource="&process;#output"/>
  <rdfs:domain rdf:resource="#SpecifyDeliveryDetails"/>
  <rdfs:range>
    <owl:Class>
       <rdfs:subClassOf rdf:resource=
                    "&process;#ConditionalOutput"/>
       <rdfs:subClassOf>
          <owl:Restriction>
            <owl:onProperty
                 rdf:resource="&process;#coCondition"/>
            <owl:allValuesFrom rdf:resource=
                    "#FedEx2-3dayCondition"/>
          </owl:Restriction>
       </rdfs:subClassOf>
       <rdfs:subClassOf>
         <owl:Restriction>
           <owl:onProperty
                rdf:resource="&process;#coOutput"/>
           <owl:allValuesFrom rdf:resource=
                    "#FedEx2-3dayDuration"/>
         </owl:Restriction>
      </rdfs:subClassOf>
    </owl:Class>
 </rdfs:range>
</rdf:Property>
```

[18] http://www.daml.org/services/owl-s/0.9/Process.owl

This definition says that deliverySelectFedEx2-3day is a conditional output, and if FedEx2-3dayCondition is true, an instance of FedEx2-3dayDuration class will be the output. However, FedEx2-3dayDuration is not defined yet.

In order to define it, we have to define its min delivery duration, i.e. 2 days, and max delivery duration, i.e. 3 days. Since the range of minDeliveryDuration and maxDeliveryDuration is Interval, intervals with specific durations need to be created first. For FedEx2-3dayDuration, we need to define Interval2Days and Interval3Days first as follows:

```
<owl:Class rdf:ID="Interval2Days">
    <!-- intervals with a duration of 2 days -->
    <rdfs:subClassOf rdf:resource="&time;#Interval"/>
    <owl:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource=
              "&time;#durationDescriptionDataType" />
       <owl:hasValue rdf:datatype="&xsd;duration">
              P2D</owl:hasValue>
      </owl:Restriction>
    </owl:subClassOf>
</owl:Class>

<owl:Class rdf:ID="Interval3Days">
    <!-- intervals with a duration of 3 days -->
    <rdfs:subClassOf rdf:resource="&time;#Interval"/>
    <owl:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource=
              "&time;#durationDescriptionDataType" />
        <owl:hasValue rdf:datatype="&xsd;duration">
              P3D</owl:hasValue>
      </owl:Restriction>
    </owl:subClassOf>
</owl:Class>
```

These two definitions use `durationDescriptionDataType`, a relatively simpler duration property of Interval using the XML Schmea datatype `duration` as its range. P2D and P3D are values of the XML Schema datatype `duration`[19], meaning 2 days and 3 days.

Finally, FedEx2-3dayDuration restricts the value of minDeliveryDuration and maxDeliveryDuration to class Interval2Days and Interval3Days respectively as follows:

```
<owl:Class rdf:ID="FedEx2-3dayDuration">
  <rdfs:subClassOf rdf:resource="#DeliveryDuration"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
            rdf:resource="#minDeliveryDuration"/>
      <owl:allValuesFrom rdf:resource="#Interval2Days"/>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
            rdf:resource="#maxDeliveryDuration"/>
      <owl:allValuesFrom rdf:resource="#Interval3Days"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Properties to output delivery durations when the user selects other delivery types (FedExOneDay, UPS, and OrdinaryMail) can be defined similarly.

[19] http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#duration

Conclusion and Future Work

In this paper, we have demonstrated how the entry sub-ontology of time can support OWL-S. We believe it should be able to help describe most of the temporal properties of real world services, since they usually only require basic topological relations, and information about durations, dates and times.

In order to test our time ontology, it needs to be linked to a temporal reasoner that reasons about the query from the user using our ontology, and produces the result to the query.

We are currently axiomatizing temporal arithmetic for the time ontology, for example, adding a duration (e.g. one month and two days) to a date/time (e.g. January 30, 2004). A treatment of temporal aggregates will also be extracted from full DAML-Time to the entry sub-ontology, so that it can express something like "every other Wednesday".

Acknowledgements

This research was funded in part by the Defense Advanced Research Projects Agency (DARPA) as part of the DAML program under Air Force Research Laboratory contract F30602-00-C-0168, and in part with funds from the Advanced Research and Development Agency (ARDA). The authors have profited from discussions with Ken Barker, Mike Dean, George Ferguson, Richard Fikes, Pat Hayes, Jessica Jenkins, David Martin, Drew McDermott, Ian Niles, James Pustejovsky, Tom Russ, James Zaiss.

References

Allen, J.F. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23, pp. 123-154.

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

Connolly D. et al., eds. 2001, DAML+OIL (March 2001) Reference Description, W3C Note 18, *World Wide Web Consortium*, Dec. 2001; http://www.daml.org/2001/03/reference

The DAML Services Coalition 2003. DAML-S (and OWL-S) 0.9 Draft Release. http://www.daml.org/services/daml-s/0.9/

Dean, M.; Schreiber, G.; Bechhofer, S.; Harmelen, F.; Hendler, J.; Horrocks, I.; McGuinness, L. D.; Patel-Schneider, F. P.; and Stein, A. L. 2003. W3C Candidate Recommendation 18 August 2003. OWL Web Ontology Language Reference. http://www.w3.org/TR/2003/CR-owl-ref-20030818/

Hobbs, J. R. 2002. A DAML Ontology of Time. http://www.cs.rochester.edu/~ferguson/daml/daml-time-nov2002.txt

OWL-S API provides support for reading, writing and executing Web Service described in OWL-S. See doc/index.html for instructions about how to use the API. There is a mailing lists for questions and feedback: owl-s@lists.mindswap.org You can subscribe to the list and see the archives here: http://lists.mindswap.org/mailman/listinfo/owl-s. Â© 2019 GitHub, Inc. Terms.Â You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Science Owls are an early version of what would eventually become Express Owls. Their main appearance was in the act Train Rush, where they attempted to prevent Hat Kid from reaching the front of the train. They also are the creators of the Owl's Brew Badge. Science Owls share many visual similarities with Express Owls, with the only notable differences being their clothing. Science Owls are clothed in a long, white lab coat which reaches down to the tops of their feet. Underneath, they wear a plain "Good Time" is a song by American electronica project Owl City and Canadian singer Carly Rae Jepsen. It was released as the lead single from Owl City's album The Midsummer Station and was used as the second single from Jepsen's second studio album, Kiss. "Good Time" was written by Matt Thiessen, Brian Lee, and Adam Young of Owl City. The song received generally positive reviews from music critics, with critics describing it as a "summer anthem".