A Temporal Aggregates Ontology in OWL for the Semantic Web

Feng Pan

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

Abstract

In this paper we describe our approach for representing temporal aggregates in OWL, as an extension to the initial version of the OWL-Time, a temporal ontology for describing the temporal content of Web pages and the temporal properties of Web services. We represent the temporal aggregates ontology in both first-order logic axioms and OWL encodings. We also present several examples in detail to show how our ontology can be used to represent complex multiple-layered and conditional temporal aggregates for the Semantic Web.

1. Introduction

Temporal information is everywhere on the Web, especially in the Web services (Dumas et al. 2001; McIlraith et al. 2001; Medjahed et al. 2003), such as temporal availability of services (e.g., "an advertised service is available from 01/01/2004 to 01/15/2005" (Dumas et al. 2001)), temporal constraints on the user's preferences (e.g., "I would prefer driving over flying if the driving time to my destination is less than three hours." (McIlraith et al. 2001); "I would like to receive this book by next Monday."), temporal information in service description (e.g., rental dates of car rental services; order dates, process time, and delivery dates of book-selling services (Pan and Hobbs 2004)), and so on.

In response to this need, in conjunction with OWL-S (OWL-S Coalition 2004), a temporal ontology, OWL-Time (Hobbs and Pan 2004) (formerly DAML-Time), has been developed for describing the temporal content of Web pages and the temporal properties of Web services, as required for Semantic Web (Berners-Lee et al. 2001) applications. 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.

OWL-Time is currently used in the OWL-S process file for the definitions of the class "Process" and "ControlConstruct", and the property "timeout" and "timeoutAbsolute". As shown in (Pan and Hobbs 2004), it can also be used to define service input parameters (e.g., the departure time for air ticketing services), output parameters (e.g., the process time for book-selling services), and conditional output parameters (e.g., delivery duration for book-selling services, depending on the type of shipping methods the user selects).

The initial version of the OWL-Time includes the topological aspects of time, measures of duration, and the clock and calendar information. We have now extended it to cover temporal aggregates as well, and this paper describes that work.

Temporal aggregates are aggregates/collections of temporal entities, for instance, "every 3rd Monday in 2001", and "4 consecutive Sundays". Such information is very common on the Web, for example, in Web services you may have "the customer service is available from 8am to 5pm EST every working day between 01/01/2004 to 01/15/2005 (Dumas et al. 2001)"; "send me the closing price of IBM, every 7 days after it exceeds $100, as long as it remains above $100 (Motakis and Zaniolo 1997)".

This paper more focuses on the OWL (McGuinness and Harmelen 2003) encodings of the temporal aggregate ontology[1]. The complete first-order logic (FOL) axiomatization of the ontology can be found in (Pan and Hobbs 2005). However, for most of the predicate definitions and the examples, both FOL axioms and the corresponding OWL encodings are shown together. In fact, we can see the two representations are very consistent and it's straightforward to map from the OWL encodings back to the FOL axioms so that it can access the full ontology of time for temporal reasoning.

In Section 2 and 3 we first describe some basics of OWL-Time, including the topological temporal relations, and the calendar and clock information. Here we only present those parts that are essential for our treatment of temporal aggregates. The full definitions of OWL-Time can be found in (Hobbs and Pan 2004). The temporal aggregates ontology, especially its OWL encodings, is described in detail in Section 4, and examples are shown in Section 5 to illustrate how our ontology can be used to represent temporal aggregates information, including complex multiple-layered and conditional temporal aggregates, in FOL and OWL.

[1] For the complete OWL encodings of the temporal aggregates ontology, see http://www.isi.edu/~pan/damltime/TemporalAggregates.owl

2. Topological Temporal Relations

The most basic temporal concepts in the 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, for example, 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. The arcs denote the (super) class has only those subclasses, for example, Instant Thing has only two subclasses: Instant and Instant Event.

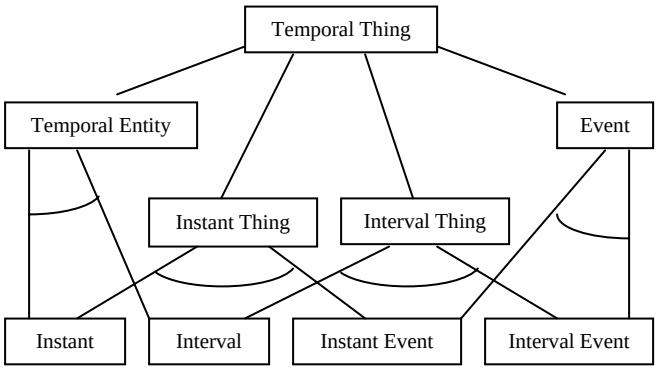

Figure1: Subclass hierarchy of temporal concepts

Their FOL axiom definitions and the corresponding OWL encodings are straightforward. For example, Temporal Entity has only two subclasses: Interval and Instant, and this is defined in FOL and OWL as:

```
FOL:
  TemporalEntity(T) ≡ Interval(T) ∨ Instant(T)

OWL:
 <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>
```

`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. `begins` is defined as:

```
FOL:
  begins(t,T) ⊃ InstantThing(t) ∧ TemporalThing(T)

OWL:
  <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.

The relations between intervals defined in Allen's temporal interval calculus (Allen 1984) and temporal durations are also defined in OWL-Time.

3. Clock and Calendar

Calendar intervals are described with the predicate `calInt`:

`calInt(y,n,u,x)`

This says that *y* is the *n*th calendar interval of type *u* in *x*. For example, the proposition `calInt (12Mar 2002,12,*Day*,Mar2002)` holds. Here *u* is one of the calendar units *Day*, *Week*, *Month*, and *Year*.

Clock intervals, weeks, days of the week, months, and years are also defined (Hobbs and Pan 2004).

Standard notation for date lists the year, month, day, and time zone. It is useful to define a predication for this: (Dates of intervals can be defined similarly)

```
dateOf(t,y,m,d,z)
   ≡ (∃d1 ,m1 ,y1 ,e) [beginsOrIn(t,d1 )
       ∧ calInt(d1 ,d,*Day*,m1 ) ∧ calInt(m1 ,m,*Month*,y1 )
       ∧ calInt(y1 ,y,*Year*,e) ∧ CE(z) = e]
```

3.1 Calendar-Clock Description

To express `calInt(y,n,u,x)` and `clockInt(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 the calendar and clock information for a calendar-clock interval, a subclass of Interval.

A calendar-clock description has the following properties/fields: unit type, year, month, week, day, day of week, day of year, hour, minute, second, and time zone:

```
<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="#timeZone" />
        <owl:maxCardinality
       rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
    </owl:Restriction>
  </owl:subClassOf>
</owl:Class>
```

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:

```
<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 `calendarClock-DescriptionOf` 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 `inCalen-darClockDatatype`. The only difference between these two relations and the above `calendarClock-DescriptionOf` and `inCalendarClock` relations is their ranges: these two simpler relations use the XSD `dateTime` 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>
```

It's much simpler to use the XSD `dateTime`, however, the advantage of using `Calendar-ClockDescription` is that it can express more information than `dateTime`, such as "week", "day of week" and "day of year". Moreover, each field of `CalendarClockDescription` is separate so that it's easier to extract the value of some fields for the later use and easier to reason about.

4. Temporal Aggregates Ontology

The predicate `everynthp` says that a temporal sequence *s* consists of every *n*th element of the temporal sequence s_0 for which property *p* is true (see (Pan and Hobbs 2005) for its definition axiom):

`everynthp(s,`s_0`,p,n)`

For example, `everynthp(s,`s_0`,Monday1,2)` defines a temporal sequence *s* of "every other Monday".

The context temporal sequence s_0 is useful not only to constrain *s* in a particular segment of time, but also to express complex multiple-layered temporal aggregates. For example, "every 3rd Monday in every July in every other year" can be split into three (primitive) temporal sequences: "every 3rd Monday" (s_1), "every July" (s_2), and "every other year" (s_3), and s_1 is in the context of s_2 which is in the context of s_3. Such examples will be illustrated in detail in the next section.

In fact, the property *p* is not limited only to simple temporal properties. In theory it can be any temporal properties. For example, in conditional temporal aggregates "every 3rd rainy day that's not a holiday", *p* is the unary predicate name representing "rainy day that's not a holiday".

The concept of granularity in (Bettini et al. 2002) corresponds to the temporal sequence concept in our terminology. All of the examples they give are uniform temporal sequences. For example, their "hour" granularity within an interval T is the set *s* such that `everynthp(s,T,hr1,1)`, where *hr1* is to *hr* as *Monday1* is

to *Monday* (see (Pan and Hobbs 2005) for the definition of *Monday* and *Monday1*).

In order to map easily between OWL-Time and iCalendar (Dawson and Stenerson 1998), we have introduced the predicate `byTulistRecurs` which is a special predicate for handling temporal aggregates that only involve temporal units (see (Pan and Hobbs 2005) for its definition axiom):

```
byTulistRecurs (s,ls,s',tu,tu')
```

It says that a temporal sequence *s* consists of a list (*ls*) of elements with temporal unit *tu* of the temporal sequence *s'* whose temporal unit is *tu'*. For example, `byTulistRecurs (s,{1, 5, 20},s',*Week*,*Year*)` defines a temporal sequence s of "every 1st, 5th and 20th weeks of a sequence (*s'*) of years".

4.1. Temporal Sequences and Their Members

In order to encode the temporal aggregates ontology in OWL, we first defined temporal sequence. It has only one optional property `hasMemeber` which maps from a temporal sequence to any temporal thing. A temporal sequence can have no (empty sequence) or many members:

```
<owl:Class rdf:ID="TemporalSeq">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember" />
          <owl:minCardinality
        rdf:datatype="&xsd;nonNegativeInteger">0
          </owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

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

Since we also want to have a backward link pointing from the temporal sequence member to its associated sequence, a `TemporalSeqMember` class is defined. It's a subclass of Temporal Thing, and has a required pair of properties: `isMemberOf` and `hasPosition`, so that it can not only point back to the associated sequence but also locate itself in the sequence:

```
<owl:Class rdf:ID="TemporalSeqMember">
  <rdfs:subClassOf rdf:resource="#TemporalThing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isMemberOf" />
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasPosition" />
        <owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="isMemberOf">
  <rdfs:domain rdf:resource="#TemporalSeqMember" />
  <rdfs:range  rdf:resource="#TemporalSeq" />
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="hasPosition">
  <rdfs:range  rdf:resource="&xsd;integer" />
</owl:DatatypeProperty>
```

Since `hasPosition` is also used for other classes, as will see later, it only has a range of intergers.

For a temporal sequence member that is associated with multiple sequences, multiple instances of `TemporalSeqMember` must be defined. The reason for defining it in this way is that for a given temporal sequence member instance, it will only have one pair of `isMemberOf` and `hasPosition` values, so that it's not confusing which `hasPosition` value should be paired with which `isMemberOf` value. Moreover, different temporal sequences may apply different attributes to their members.

4.2. Temporal Aggregate Description

The most important class in the OWL encodings of the temporal aggregates ontology is the temporal aggregate description class. Analogous to the calendar-clock description, it specifies the temporal aggregate description for temporal sequences, and it's associated with the temporal sequence class by `hasTemporalAggregate-Description` property.

The temporal aggregate description has the following fields/properties: `hasStart`, `hasEnd`, `hasContext-TemporalSeq`, `hasithTemporalUnit`, `hasTemporalUnit`, `hasContextTemporalUnit`, `hasPosition`, `hasGap`, and `hasCount`:

```
<owl:Class rdf:ID="TemporalAggregateDescription">
  <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#hasStart" />
      <owl:maxCardinality
        rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasCount" />
        <owl:maxCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:ObjectProperty
      rdf:ID="hasTemporalAggregateDescription">
  <rdfs:domain rdf:resource="#TemporalSeq" />
  <rdfs:range  rdf:resource="#TemporalAggregateDescription" />
</owl:ObjectProperty>
```

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

The optional properties `hasStart` and `hasEnd` map from the temporal aggregate description to the instant thing, specifying the start and the end instants of a temporal sequence. The calendar and clock properties described in Section 3 can then be used to specify the start and the end times or dates the instants are in.

```
<owl:ObjectProperty rdf:ID="hasContextTemporalSeq">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range  rdf:resource="#TemporalSeq" />
</owl:ObjectProperty>
```

The optional property `hasContextTemporalSeq` maps from the temporal aggregate description to the temporal sequence, specifying the context (super) temporal sequence of a given (sub) temporal sequence.

It corresponds to s_0 in `everynthp(s,s₀,p,n)` and *s'* in `byTulistRecurs(s,ls,s',tu,tu')`. When it's not present, context-free temporal aggregates (e.g. "every Monday") can be represented.

```
<owl:DatatypeProperty rdf:ID="hasithTemporalUnit">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

The required property `hasithTemporalUnit` maps from the temporal aggregate description to positive integers, specifying the *i*th temporal unit elements in the temporal sequence.

It corresponds to *ls* in `byTulistRecurs(s,ls,s', tu,tu')`. Thus it's very possible to have many such property values for a given temporal sequence. For example, "every 3rd Monday, Tuesday, and Friday" (such examples will be illustrated in detail in the next section).

```
<owl:ObjectProperty rdf:ID="hasTemporalUnit">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range  rdf:resource="#TemporalUnit" />
</owl:ObjectProperty>
</owl:ObjectProperty>
```

The required properties `hasTemporalUnit` and the optional `hasContextTemporalUnit` map from the temporal aggregate description to the temporal unit, as defined in Section 3.1. They specify the temporal unit of the given temporal sequence and the context temporal sequence respectively. They correspond to *tu* and *tu'* in `byTulistRecurs(s,ls,s',tu,tu')`.

The context temporal unit is associated with the context temporal sequence property. Thus if the context temporal sequence is not present, so is the context temporal unit, but not vice versa, since it's possible that the temporal unit of the context temporal sequence is unknown or not relevant.

```
<owl:DatatypeProperty rdf:ID="hasPosition">
  <rdfs:range  rdf:resource="&xsd;integer" />
</owl:DatatypeProperty>
```

The optional property `hasPosition` is a shared property with the `TemporalSeqMember` class. It specifies the position of the element in the temporal sequence. For example, "the first two Tuesdays in every May" would have `hasPosition` value of 2. It's also possible to have negative positions. For example, "the last Thursday in every November" would have `hasPosition` value of -1.

If this property value is not present, all the positions will be included in the temporal sequence. For example, "the Thursday in every November" includes all the Thursdays in every November, while "the last Thursday in every November" only includes the last Thursday in every November.

```
<owl:DatatypeProperty rdf:ID="hasGap">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

The optional property `hasGap` maps from the temporal aggregate description to positive integers, specifying the gap between the elements in the temporal sequence. If it's not present, the default value of 1 will be used, for example, as in "every Monday".

It corresponds to *n* in `everynthp(s,s₀,p,n)`. For example, "every 3rd Monday" would have `hasGap` value of 3.

```
<owl:DatatypeProperty rdf:ID="hasCount">
  <rdfs:domain rdf:resource="#TemporalAggregateDescription" />
  <rdfs:range  rdf:resource="&xsd;positiveInteger" />
</owl:DatatypeProperty>
```

The optional property `hasCount` maps from the temporal aggregate description to positive integers, specifying the cardinality or the size of the temporal sequence. For example, "four consecutive Sundays" would have `hasCount` value of 4.

5. Temporal Aggregates Examples

In this section, we will demonstrate how our temporal aggregates ontology can be used to represent different kinds of temporal aggregates examples in both FOL axioms and OWL encodings, including complex multiple-layered temporal aggregates and conditional temporal aggregates.

- *Every other Monday in every 3rd month.*

FOL:

$(\exists\, s,s_1,s_2)$ [everynthp(s_2,s_1,Month1,3)
$\wedge$ everynthp(s,s_2,Monday1,2)]

where $(\forall\, m)$ [Month1(m) $\equiv$ $(\exists\, n,x)$ [calInt(m,n,*Month*,x)]]

(∀ d) [Monday1(d) ≡ (∃ w) [Monday(d,w)[2]]]

OWL:

```
<time-entry:TemporalSeq rdf:ID="tseq">
  <time-entry:hasTemporalAggregateDescription
        rdf:resource="#everyOtherMondayEvery3rdMonth" />
</time-entry:TemporalSeq>

<time-entry:TemporalSeq rdf:ID="tseq-every3rdMonth">
  <time-entry:hasTemporalAggregateDescription
        rdf:resource="#every3rdMonth" />
</time-entry:TemporalSeq>

<time-entry:TemporalAggregateDescription
                        rdf:ID="every3rdMonth">
  <time-entry:hasTemporalUnit
        rdf:resource="&time-entry;unitMonth" />
  <time-entry:hasGap rdf:datatype="&xsd;positiveInteger">3
</time-entry:TemporalAggregateDescription>

<time-entry:TemporalAggregateDescription
        rdf:ID="everyOtherMondayEvery3rdMonth">
  <time-entry:hasContextTemporalSeq
        rdf:resource="#tseq-every3rdMonth" />
  <time-entry:hasithTemporalUnit
        rdf:datatype="&xsd;positiveInteger">1
  </time-entry:hasithTemporalUnit>
  <time-entry:hasTemporalUnit
        rdf:resource="&time-entry;unitDay" />
  <time-entry:hasContextTemporalUnit
        rdf:resource="&time-entry;unitMonth" />
  <time-entry:hasGap rdf:datatype="&xsd;positiveInteger">2
  </time-entry:hasGap>
</time-entry:TemporalAggregateDescription>
```

The FOL axiom defines *s* as the set corresponding to the given temporal aggregate. The first part of the axiom defines s_2 as the set corresponding to "every 3rd month", and it serves as the context temporal sequence for the desired temporal sequence *s*.

The OWL encodings show how `hasContext-TemproalSeq` is used to represent a two-layered temporal sequence ("every other Monday" in "every 3rd month"), and how `hasGap` is used to represent "every other" (with a gap of 2) and "every 3rd" (with a gap of 3).

In order to define a temporal sequence (i.e., `tseq`) for the given temporal aggregate, we need to first define a temporal description for that (i.e., `everyOtherMondayEvery3rdMonth`). To represent this two-layered temporal sequence, we first define the outer layer temporal sequence (i.e., "every 3rd month"), and it serves as the context temporal sequence for the inner layer temporal sequence (i.e., "every other Monday").

The same such embedding structure for multiple-layered temporal sequences is shown consistently in the above representations in both FOL axioms and OWL encodings.

- *Every other week on Monday, Wednesday and Friday until December 24, 1997, but starting on Tuesday, September 2, 1997.*[3]

FOL:

(∃ s,s',T,t_1,t_2) [everynthp(s',{T},Week1,2)
∧ byTulistRecurs(s,{1, 3, 5},s',*Day*,*Week*)
∧ begins(t_1,T) ∧ ends(t_2,T) ∧ dateOf(t_1,1997,9,2)
∧ dateOf(t_2,1997,12,24)]

where (∀ w) [Week1(w) ≡ (∃ n,x) [calInt(w,n,*Week*,x)]]

OWL:

```
<time-entry:TemporalSeq rdf:ID="tseq">
  <time-entry:hasTemporalAggregateDescription
        rdf:resource="#MWFeveryOtherWeek" />
</time-entry:TemporalSeq>

<time-entry:TemporalSeq rdf:ID="tseq-everyOtherWeek">
  <time-entry:hasTemporalAggregateDescription
        rdf:resource="#everyOtherWeek" />
</time-entry:TemporalSeq>

<time-entry:TemporalAggregateDescription
                        rdf:ID="everyOtherWeek">
  <time-entry:hasTemporalUnit
        rdf:resource="&time-entry;unitWeek" />
  <time-entry:hasGap rdf:datatype="&xsd;positiveInteger">2
  </time-entry:hasGap>
</time-entry:TemporalAggregateDescription>

<time-entry:TemporalAggregateDescription
                        rdf:ID="MWFeveryOtherWeek">
  <time-entry:hasStart rdf:resource="#tseqStart" />
  <time-entry:hasEnd rdf:resource="#tseqUntil" />
  <time-entry:hasContextTemporalSeq
        rdf:resource="#tseq-everyOtherWeek" />
  <time-entry:hasithTemporalUnit
        rdf:datatype="&xsd;positiveInteger">1
  </time-entry:hasithTemporalUnit>
  <time-entry:hasithTemporalUnit
        rdf:datatype="&xsd;positiveInteger">3
  </time-entry:hasithTemporalUnit>
  <time-entry:hasithTemporalUnit
        rdf:datatype="&xsd;positiveInteger">5
  </time-entry:hasithTemporalUnit>
  <time-entry:hasTemporalUnit
        rdf:resource="&time-entry;unitDay" />
  <time-entry:hasContextTemporalUnit
        rdf:resource="&time-entry;unitWeek" />
</time-entry:TemporalAggregateDescription>

<time-entry:Instant rdf:ID="tseqStart">
  <time-entry:inCalendarClock
        rdf:resource="#tseqStartDescription" />
</time-entry:Instant>

<time-entry:Instant rdf:ID="tseqUntil">
  <time-entry:inCalendarClockDataType
        rdf:datatype="&xsd;dateTime">1997-12-24
  </time-entry:inCalendarClockDataType>
</time-entry:Instant>

<time-entry:CalendarClockDescription
                        rdf:ID="tseqStartDescription">
  <time-entry:unitType rdf:resource="&time-entry;unitDay" />
  <time-entry:year rdf:datatype="&xsd;gYear">1997
  </time-entry:year>
  <time-entry:month rdf:datatype="&xsd;gMonth">9
  </time-entry:month>
  <time-entry:day rdf:datatype="&xsd;gDay">2</time-entry:day>
  <time-entry:dayOfWeekField
      rdf:datatype="&xsd;nonNegativeInteger">2
  </time-entry:dayOfWeekField>
</time-entry:CalendarClockDescription>
```

[2] It says *d* is the Monday of the week *w*, see (Hobbs and Pan 2004) for its definition.

[3] This example is taken from iCalendar RFC 2445 page 120.

The FOL axiom defines *s* as the set corresponding to the given temporal aggregate. The first part of the axiom defines *s'* as the set corresponding to "every other week", and it serves as the context temporal sequence for the desired temporal sequence *s*. Predicates `begins` and `ends` are used to represent the start and the end times of the given temporal aggregate.

Besides what is shown in the first example, the OWL encodings for this one show how `hasithTemporalUnit` is used to represent a list of temporal elements in the temporal sequence (i.e., "on Monday, Wednesday, and Friday"), and how `hasStart` and `hasEnd` are used and combined with the calendar and clock representations to represent the start and end dates of the given temporal aggregate.

This example also shows the tradeoffs of using XSD `dateTime` and the `CalendarClockDescription` class defined in OWL-Time, as mentioned in Section 3. In this example, the end date is represented using XSD `dateTime`, while the start date is represented using the `CalendarClockDescription` class. As we can see, XSD dateTime is simpler, but there's some information it cannot represent, for example, the start date is Tuesday, and this is the reason why `CalendarClock-Description` class (with `dayOfWeekField` property) is used for the start date. In fact, `CalendarClock-Description` class can also represent other information that XSD `dateTime` cannot, such as "week" and "day of year". Moreover, each field of the class is separate so that it's easier to extract the values of some fields for the later use and easier to reason about.

- *Every Monday that's a holiday.*

FOL:

$(\exists\ s,s_0)$ [everyp(s,s_0,HolidayMonday)]

where $(\forall\ d)$ [HolidayMonday(d) $\equiv (\exists\ w)$ [Monday(d,w)] $\wedge$ holiday(d)[4]]

OWL:

```
<EveryHolidayMonday rdf:ID="tseq" />

<owl:Class rdf:ID="EveryHolidayMonday">
  <rdfs:subClassOf rdf:resource="&time-entry;TemporalSeq"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMember" />
      <owl:allValuesFrom
        rdf:resource="#EveryHolidayMondayMember" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="EveryHolidayMondayMember">
  <rdfs:subClassOf
      rdf:resource="&time-entry;TemporalSeqMember"/>
  <rdfs:subClassOf rdf:resource="&time-entry;Holiday"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#isMemberOf" />
      <owl:allValuesFrom rdf:resource="#EveryMonday" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:ID="EveryMonday">
  <rdfs:subClassOf rdf:resource="&time-entry;TemporalSeq"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty
        rdf:resource="#hasTemporalAggregateDescription" />
      <owl:hasValue rdf:resource="#everyMonday" />
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>

<time-entry:TemporalAggregateDescription
                                  rdf:ID="everyMonday">
  <time-entry:hasithTemporalUnit
      rdf:datatype="&xsd;positiveInteger">1
  </time-entry:hasithTemporalUnit>
  <time-entry:hasTemporalUnit
      rdf:resource="&time-entry;unitDay" />
</time-entry:TemporalAggregateDescription>
```

This example shows an important advantage of using our temporal aggregates ontology. It can represent *conditional temporal aggregates* which is hard or impossible in some other representations. For example, there is no way in iCalendar to express such conditional temporal aggregates or any temporal aggregates that are not in a form of standard temporal units, such as "holidays", "voting dates", "days with classes", "months starting with a Monday", and so on.

As we can see, the FOL axiom is much simpler than the corresponding OWL encodings. It defines *s* as the set corresponding to the given temporal aggregate, and a new predicate (i.e., `HolidayMonday`) for this conditional temporal aggregate.

The OWL encodings show how this kind of conditional temporal aggregates can be defined in our representation in OWL.

The most important class in this example is the `EveryHolidayMondayMember` class which defines a class for the members of the desired temporal sequence class (i.e., `EveryHolidayMonday`). This class is both a temporal sequence member and a holiday, and its associated temporal sequence class must be "every Monday" (i.e., `EveryMonday` class).

The desired temporal sequence is an instance of the `EveryHolidayMonday` class whose members are only from the `EveryHolidayMondayMember` class.

6. Conclusion

In this paper, we have described our work of representing temporal aggregates in OWL-Time. The ontology was represented in both first-order logic axioms and OWL encodings. Examples have also been shown to illustrate how our ontology can be used to represent temporal aggregates information in FOL and OWL for the Semantic Web.

[4] It says *d* is a Holiday, see (Hobbs and Pan 2004) for its definition.

Acknowledgements
This work was supported by the Advanced Research and Development Activity (ARDA) under DOD/DOI/ARDA Contract No. NBCHC040027. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARDA or the Department of Interior.

7. 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, 2001. The Semantic Web. *Scientific American*, 284(5):34–43.

Bettini, C., Wang, S. X. and Jajodia, S., 2002. Solving multi-granularity temporal constraint networks, *Artificial Intelligence*, vol. 140, pp. 107-152.

Dawson, F. and Stenerson, D, 1998. Internet Calendaring and Scheduling Core Object Specification (iCalendar), RFC2445, *Internet Society*.

Dumas, M., J. O'Sullivan, M. Heravizadeh, D. Edmond, and A. Hofstede, 2001. Towards a semantic framework for service description. *In Proceedings of the IFIP Conference on Database Semantics*, Hong Kong.

Hobbs, J. R. and Pan, F, 2004. An Ontology of Time for the Semantic Web. *ACM Transactions on Asian Language Processing (TALIP)* Vol. 3, No. 1, pp. 66-85.

McGuinness, D. L. and Harmelen, F. v, 2003. OWL Web Ontology Language Overview. *World Wide Web Consortium (W3C) Candidate Recommendation*.

McIlraith, S. A., Son, T. C. and Zeng, 2001. H. Semantic Web Services. *IEEE Intelligent Systems* 16(2):46–53.

Medjahed, B., Bouguettaya, A. and Elmagarmid, A, 2003. Composing Web Services on the Semantic Web. *The Very Large Data Base Journal, Special Issue on the Semantic Web,*Springer Verlag, 12(4).

Motakis, I. and C. Zaniolo, 1997. Temporal Aggregation in Active Databases Rules. In *International Conference. on the Management of Data*, May.

OWL-S Coalition, 2004. OWL-S 1.1 Release. (http://www.daml.org/services/owl-s/1.1/)

Pan, F. and Hobbs, J. R., 2004. Time in OWL-S. *In Proceedings of the AAAI Spring Symposium on Semantic Web Services*, Stanford University, CA.

Pan, F. and Hobbs, J. R., 2005. Temporal Aggregates in OWL-Time. *In Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, Clearwater Beach, Florida, pp. 560-565, AAAI Press.