Temporal Aggregates in OWL-Time

Feng Pan and Jerry R. Hobbs

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

Abstract

In this paper, we describe our approach for representing temporal aggregates in OWL-Time, and a systematic way of mapping iCalendar recurrence sets to OWL-Time temporal sequences. We present several examples of natural language expressions for temporal aggregates and how they would be represented in OWL-Time; some examples can be described in both iCalendar and OWL-Time, while others can only be represented in OWL-Time. We also illustrate how we can translate from a natural language sentence to our representation.

Introduction

Natural language texts involve many expressions for temporal aggregates, such as "every Tuesday", "every 3rd Monday in 2001", "4 consecutive Sundays", "3 weekdays after today", "the 4th of 6 days of voting", and so on. Thus it is crucial to have a good ontology of temporal aggregates to represent these expressions.

OWL-Time (formerly DAML-Time, Hobbs and Pan 2004) is an ontology of temporal concepts for describing the temporal content of Web pages and the temporal properties of Web services. The initial version of the ontology axiomatized the topological aspects of time, measures of duration, and the clock and calendar. We have now extended it to cover temporal aggregates as well, and this paper describes that work.

The representation proposed in this paper is useful for many natural language processing tasks, such as information retrieval and question answering. For example, a question answering system may have in its knowledge base that "Bob and Mary had meetings on every 3rd Mondays in 2004 except holidays." and we needs a temporal aggregates ontology to reason and answer questions like "how many meetings did Bob and Mary have in 2004?"

In Section 2 we introduce the basic temporal concepts and parts of the full OWL-Time ontology that are essential for our treatment of temporal aggregates. In Section 3 we describe our general approach to temporal aggregates that is intended to handle any collection of temporal entities, regardless of how they are described. An important special case is temporal aggregates consisting of clock and calendar temporal entities, like calendar months. iCalendar (Dawson and Stenerson 1998) is a popular framework for describing these, and in Section 4 we show how iCalendar can be embedded in OWL-Time. In Section 5 we present several examples of natural language expressions for temporal aggregates and how they would be represented in OWL-Time. Some can be described in both iCalendar and OWL-Time, while others can only be represented in OWL-Time. We also demonstrate how we can translate from a natural language sentence to our representation.

OWL-Time Basics

The full definitions of OWL-Time can be found in (Hobbs and Pan, 2004). Here we present those parts that are essential for our treatment of temporal aggregates. In an extension of the time ontology, we also allow temporal predicates to apply directly to events, should the user wish, but here we restrict our treatment to temporal entities.

Topological Temporal Relations

There are two subclasses of TemporalEntity: Instant and Interval.

$$(\forall\ T)[\text{TemporalEntity}(T) \equiv \text{Interval}(T) \vee \text{Instant}(T)]$$

Intervals are, intuitively, things with extent and instants are, intuitively, point-like in that they have no interior points.

The predicates "begins" and "ends" are relations between instants and temporal entities.

$$\text{begins}(t,T) \supset \text{Instant}(t) \wedge \text{TemporalEntity}(T)$$
$$\text{ends}(t,T) \supset \text{Instant}(t) \wedge \text{TemporalEntity}(T)$$

The predicate "inside" is a relation between an instant and an interval.

$$\text{inside}(t,T) \supset \text{Instant}(t) \wedge \text{Interval}(T)$$

This concept of "inside" is not intended to include beginnings and ends of intervals.

It will be useful in characterizing clock and calendar terms to have a relation between instants and intervals that says that the instant is inside or the beginning of the interval.

$$(\forall\ t,T)[\text{beginsOrIn}(t,T) \equiv [\text{begins}(t,T) \vee \text{inside}(t,T)]]$$

We can define a proper interval as one whose start and end are not identical.

$$(\forall\ T)\text{ProperInterval}(T) \equiv \text{Interval}(T) \wedge (\forall\ t_1,t_2)[\text{begins}(t_1,T) \wedge \text{ends}(t_2,T) \supset t_1 \neq t_2\ ]]$$

There is a "before" relation on temporal entities, which gives directionality to time. If temporal entity T_1 is before temporal entity T_2, then the end of T_1 is before the start of T_2. Thus, "before" can be considered to be basic to instants and derived for intervals.

$$(\forall T_1,T_2)[\text{before}(T_1,T_2) \equiv (\exists t_1,t_2)[\text{ends}(t_1,T_1) \wedge \text{begins}(t_2,T_2) \wedge \text{before}(t_1,t_2)]]$$

The relations between intervals defined in Allen's temporal interval calculus (Allen 1984) can be defined in a straightforward fashion in terms of "before" and identity on the beginning and end points. It is a bit more complicated than the reader might at first suspect, since allowance has to be made for the possibility of infinite intervals. Since one of the intervals could be infinite and lack an end point, the relation between the end points has to be dependent on their existence.

OWL-Time includes axioms defining 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:

$$(\forall T_1,T_2)[\text{intEquals}(T_1,T_2) \equiv [\text{ProperInterval}(T_1) \wedge \text{ProperInterval}(T_2) \wedge (\forall t_1)[\text{begins}(t_1,T_1) \equiv \text{begins}(t_1,T_2)] \wedge (\forall t_2)[\text{ends}(t_2,T_1) \equiv \text{ends}(t_2,T_2)]]]$$

Clock and Calendar

A day as a calendar interval begins at and includes midnight, and goes until, but does not include, the next midnight. This contrasts with a day as a duration which is any interval that is 24 hours in length. The day as a duration is dealt with in the full OWL-Time ontology; for this paper we need only 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 am, including 12:00 am, 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. Clock intervals are described with the predicate "clockInt":

clockInt(y,n,u,x)

This expression says that y is the nth clock interval of type u in x. For example, the proposition "clockInt(10:03,3, *Minute*,[10:00,11:00])" holds. Here u is a member of the set of clock units, that is, one of *Second*, *Minute*, or *Hour*. The larger interval x may not line up exactly with clock intervals. In this case we take y to be the nth complete clock interval of type u in x. In addition, there is a calendar unit function with similar structure:

calInt(y,n,u,x)

This says that y is the nth calendar interval of type u in x. For example, the proposition "calInt(12Mar2002,12, *Day*,Mar2002)" holds. Here u is 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 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 "clockInt" runs from 0 to N-1; whereas, in "calInt", n runs from 1 to N. To state properties true of both clock and calendar intervals, we can use the predicate "calInt" and relate the two notions with the axiom

calInt(y,n,u,x) ≡ clockInt(y,n-1,u,x)

The names of months can be defined in terms of the predicate "calInt". For example, July is the seventh month of a year.

July(m,y) ≡
calInt(m,7,*Month*,y) ∧ (∃ n,t) [calInt(y,n,*Year*,t)]

The top-level time interval (for modern applications) is CE(z), which is the Common Era in time zone z. Thus, the year 2005 in the Eastern Standard Time Zone is the y such that "calInt(y, 2005, *Year*, CE(*EST*))".

A week is any seven consecutive days. A calendar week, by contrast, according to a commonly adopted convention, starts at midnight, Saturday night, and goes to the next midnight, Saturday night. That is, weeks start with Sunday. (By contrast, the ISO 8061 standard week starts with Monday.) There are 52 weeks in a year, but there are not usually 52 calendar weeks in a year. Weeks are independent of months and years. However, we can still talk about the *n*th week in some larger period of time, e.g., the third week of the month or the fifth week of the semester. To say a time interval is a calendar-week, we say

calInt(y,n,*Week*,x)

As it happens, the *n* and *x* arguments will often be irrelevant when we only want to say that some period is a calendar week, and not say which.

The day of the week is a calendar interval of type *Day*. The *n*th day-of-the-week in a week is the *n*th day in that interval.

dayofweek(y,n,x) ≡
calInt(y, n,*Day*,x) ∧ (∃ n_1,x_1) calInt(x,n,*Week*, x_1)

The days of the week have special names in English, such as Sunday, Monday, and so on. For example, Monday is defined as follows:

dayofweek(y,2,x) ≡ Monday(y,x)

This says that *y* is the Monday of week *x*.

The ISO 8061 standard week is related to the traditional week as follows:

0 < n < 7 ⊃ [isodayofweek (y,n,x) ≡ dayofweek(y,n+1,x)]
isodayofweek (y,7,x) ≡ Sunday(y,x)

Holidays can also be specified in this ontology. To say that July 4 is a holiday one could write

(∀ d,m,y)[calInt(d,4,*Day*,m) ∧ July(m,y) ⊃ holiday(d)]

Holidays like Easter can be defined in terms of this ontology coupled with an ontology of the phases of the moon.

Standard notation for date lists the year, month, day, and time zone. It is useful to define a predication for this.

dateOf(t,y,m,d,z)
≡ ($\exists d_1, m_1, y_1$,e) [beginsOrIn(t,d_1)
∧ calInt(d_1,d,*Day*,m_1) ∧ calInt(m_1,m,*Month*,y_1)
∧ calInt(y_1,y,*Year*,e) ∧ CE(z) = e]

Dates of intervals can be defined similarly.

Temporal Aggregates

In this section, we assume the notation of set theory. Sets and elements of sets will be ordinary individuals, and relations such as "member" will be relations between such individuals. In particular, we will use the relation "member" between an element of a set and the set. We will use the notation "{x}" for the singleton set containing the element x. We will use the function "union" to refer to the union operation between two sets. The function "card" will map a set into its cardinality.

In addition, for convenience, we will make moderate use of second-order formulations, and quantify over predicate symbols. This could be eliminated with the use of an "apply" predicate and axiom schemas systematically relating predicate symbols to corresponding individuals, e.g., the axiom schema for unary predicates p,

(∀ x)[apply(*p*,x) ≡ p(x)]

It will be convenient to have a relation "ibefore" that generalizes over several interval and instant relations, covering both "intBefore" and "intMeets" for proper intervals.

(∀ T_1,T_2)[ibefore(T_1,T_2)
≡ [before(T_1,T_2) ∨ [ProperInterval(T_1)
∧ ProperInterval(T_2) ∧ intMeets(T_1,T_2)]]]

It will also be useful to have a relation "iinside" that generalizes over all temporal entities and aggregates. We first define a predicate "iinside-1" that generalizes over instants and intervals and covers "intStarts", "intFinishes" and "intEquals" as well as "intDuring" for intervals.

A temporal aggregate is first of all a set of temporal entities, but it has further structure. The relation "ibefore" imposes a natural order on some sets of temporal entities, and we will use the predicate "tseq" to describe those sets.

(∀ s)[tseq(s) ≡ (∀ t)[member(t,s) ⊃ TemporalEntity(t)]
∧ (∀ t_1,t_2)[member(t_1,s) ∧ member(t_2,s)
⊃ [t_1 = t_2 ∨ ibefore(t_1,t_2) ∨ ibefore(t_2,t_1)]]]

That is, a temporal sequence is a set of temporal entities totally ordered by the "ibefore" relation. A temporal sequence has no overlapping temporal entities.

It will be useful to have the notion of a temporal sequence whose elements all have a property p.

(∀ s,p)[tseqp(s,p) ≡ tseq(s) ∧ (∀ t)[member(t,s) ⊃ p(t)]]

The same temporal aggregate can be broken up into a set of intervals in many different ways.

A minimal temporal sequence is one whose intervals are maximal, so that the number of intervals is minimal. We can view a week as a week or as 7 individual successive days; the first would be minimal. We can go from a non-minimal to a minimal temporal sequence by concatenating intervals that meet.

(∀ s)[min-tseq(s)
≡ (∀ t_1,t_2)[member(t_1,s) ∧ member(t_2,s)
⊃ [t_1 = t_2 ∨ (∃ t)[ibefore(t_1,t) ∧ ibefore(t,t_2)
∧ ~member(t,s)]]]]

That is, s is a minimal temporal sequence when any two distinct intervals in s have a temporal entity not in s between them.

A temporal sequence s_1 is a minimal equivalent temporal sequence to temporal sequence s_2 if s_1 is minimal and equivalent to s_2.

(∀ s_1,s_2)[min-equiv-tseq(s_1,s_2)
≡ min-tseq(s_1) ∧ tseq(s_2)
∧ (∀ t,t_2)[TemporalEntity(t) ∧ iinside-1(t,t_2)
∧ member(t_2,s_2) ⊃ (∃ t_1)[member(t_1,s_1) ∧ iinside(t,t_1)]]]

We can now generalize "iinside-1" to the predicate "iinside", which covers both temporal entities and temporal sequences. A temporal entity is "iinside" a temporal sequence if it is "iinside-1" one of the elements of its minimal equivalent temporal sequence.

(∀ t,s)[iinside(t,s)
≡ [TemporalEntity(t) ∧ TemporalEntity(s) ∧ iinside-1(t,s)]
∨ [TemporalEntity(t) ∧ tseq(s)
∧ (∃ s_1,t_1)[min-equiv-tseq(s_1,s) ∧ member(t_1,s_1)
∧ iinside-1(t,t_1)]]]

We can define a notion of "isubset" on the basis of "iinside".

(∀ s,s_0)[isubset(s,s_0)
≡ [tseq(s) ∧ tseq(s_0) ∧ (∀ t)[member(t,s) ⊃ iinside(t,s_0)]]]

That is, every element of temporal sequence s is inside some element of the minimal equivalent temporal sequence of s_0.

We can also define a relation of "idisjoint" between two temporal sequences.

(∀ s1,s2)[idisjoint(s_1,s_2)
≡ [tseq(s_1) ∧ tseq(s_2)
∧ ~(∃ t,t_1,t_2)[member(t_1,s_1) ∧ member(t_2,s_2)
∧ iinside(t,t_1) ∧ iinside(t,t_2)]]]

That is, temporal sequences s1 and s2 are disjoint if there is no overlap between the elements of one and the elements of the other.

The last temporal entity in a temporal sequence is the one with any of the others "ibefore" it.

(∀ t,s)[last(t,s)
≡ [tseq(s) ∧ member(t,s)
∧ (∀ t_1)[member(t_1,s) ⊃ [t_1 = t ∨ ibefore(t_1,t)]]]]

More generally, we can talk about the nth element of temporal sequence.

(∀ t,s)[nth(t,n,s)
≡ [tseq(s) ∧ member(t,s) ∧ natnum(n)

$\wedge (\exists\ s_1)[(\forall\ t_1)[member(t_1,s_1)$
$\equiv [member(t_1,s) \wedge ibefore(t_1,t)]] \wedge card(s_1) = n\text{-}1]]]$

That is, the nth element of a temporal sequence has n-1 elements before it.

The predicate "ngap" will enable us to define "everynthp" below. Essentially, we are after the idea of a temporal sequence s containing every nth element of s_0 for which p is true. The predicate "ngap" holds between two elements of s and says that there are n-1 elements between them that are in s_0 and not in s for which p is true.

$(\forall\ t_1,t_2,s,s_0,p,n)\ [ngap(t_1,t_2,s,s_0,p,n)$
$\equiv [member(t_1,s) \wedge member(t_2,s) \wedge tseqp(s,p)$
$\wedge tseq(s_0) \wedge isubset(s,s_0) \wedge natnum(n)$
$\wedge (\exists\ s_1)[card(s_1) = n\text{-}1 \wedge idisjoint(s,s_1)$
$\wedge (\forall\ t)[member(t,s_1)$
$\equiv [iinside(t,s_0) \wedge p(t) \wedge ibefore(t_1,t) \wedge ibefore(t,t_2)]]]]]$

The predicate "everynthp" says that a temporal sequence s consists of every nth element of the temporal sequence s_0 for which property p is true. It will be useful in describing temporal aggregates like "every third Monday in 2001".

$(\forall\ s,s_0,p,n)[everynthp(s,s_0,p,n)$
$\equiv [tseqp(s,p) \wedge tseq(s_0) \wedge natnum(n)$
$\wedge (\exists\ t_1)[nth(t_1,1,s) \wedge \sim(\exists\ t)[iinside(t,s_0) \wedge ngap(t,t_1,s,s_0,p,n)]]$
$\wedge (\exists\ t_2)[last(t_2,s) \wedge \sim(\exists\ t)[iinside(t,s_0) \wedge ngap(t_2,t,s,s_0,p,n)]]$
$\wedge (\forall\ t_1)[last(t_1,s) \vee (\exists\ t_2)\ ngap(t_1,t_2,s,s_0,p,n)]]]$

That is, the first element in s has no p element n elements before it in s_0, the last element in s has no p element n elements after it, and every element but the last has a p element n elements after it.

The variable for the temporal sequence s_0 is, in a sense, a context parameter. When we say "every other Monday", we are unlikely to mean every other Monday in the history of the Universe. The parameter s_0 constrains us to some particular segment of time. (Of course, that segment could in principle be the entire time line.)

The definition of "everyp" is simpler.

$(\forall\ s,s_0,p)[everyp(s,s_0,p)$
$\equiv (\forall\ t)[member(t,s) \equiv [iinside(t,s_0) \wedge p(t)]]]$

It is a theorem that every p is equivalant to every first p.

$(\forall\ s,s_0,p)[everyp(s,s_0,p) \equiv everynthp(s,s_0,p,1)]$

We could similarly define "every-other-p", but the resulting simplification from "everynthp(s,s_0,p,2)" would not be sufficient to justify it.

Embedding iCalendar Recurrence Sets in OWL-Time

Internet Calendaring and Scheduling Core Object Specification (iCalendar) is a widely supported standard for personal data interchange. It provides the definition of a common format for openly exchanging calendaring and scheduling information across the Internet. The recurrence set in iCalendar is the complete set of recurrence instances for a calendar component. iCalendar recurrence sets are a subset of OWL-Time temporal sequences.

We have developed a systematic way of mapping a recurrence set in iCalendar to a temporal sequence in OWL-Time. Here we demonstrate this on an illustrative subset of recurrence sets.

As shown in the next section with natural language examples, there are cases that can be expressed in OWL-Time more accurately than in iCalendar, and there are also cases that iCalendar can't express, but OWL-Time can. Moreover, embedding recurrence sets in OWL-Time gives us access to the full ontology of time for temporal reasoning.

A recurrence set *s* is defined by a recurrence rule *r*. We express this as RecurrenceSetRule(s,r). First, we list the properties of a recurrence rule r:

$(\forall\ r)\ [rule(r)$
$\supset (\exists\ tu,g)\ [freq(r)=tu \wedge TemporalUnit(tu) \wedge gap(g,r)]]$

freq(r) corresponds to the FREQ property of recurrence rules. Since it is a required property, it is defined as a function mapping from a recurrence rule to a temporal unit. For example, FREQ = WEEKLY will have a return value of temporal unit *Week*.

gap(g,r) corresponds to the INTERVAL property of recurrence rules; if it is missing, we assume it is given a default value of 1 during the translation process from iCalendar to OWL-Time.

$(\forall\ r)\ [rule(r)$
$\supset (\exists\ n)\ [count(n,r) \wedge natnum(n)]$
$\vee (\exists\ t,y,mo,d,h,mi,s,z)\ [until(t,r)$
$\wedge timeOf(t,y,mo,d,h,mi,s,z)]]$

count(n,r) corresponds to the COUNT property of recurrence rules. until(t,r) corresponds to the UNTIL property of recurrence rules. It is required in iCalendar that either UNTIL or COUNT may appear in a recurrence rule, but they must not occur in the same rule.

Then, we specify the optional properties of a recurrence rule as in:

$(\forall\ r,\ ls)\ [rule(r) \wedge bysecond(ls,\ r)$
$\supset (\forall\ s)\ [member(s,\ ls) \supset integer(s) \wedge 0<=s<=59]]$

corresponding to the BYSECOND property of recurrence rules. BYMINUTE, BYHOUR, and so on are defined similarly.

Now we can map from the recurrence set generated by a recurrence rule *r* with either COUNT or UNTIL to a temporal sequence *s*:

$(\forall\ r,n,s,g)\ [RecurrenceSetRule(s,r) \wedge count(n,r) \wedge gap(g,r)$
$\supset (\exists\ s_0)\ [card(s)=n \wedge everynthp(s,s_0,map2p(freq(r)),g)]]$

$(\forall\ r,t,s,g)\ [RecurrenceSetRule(s,r) \wedge until(t,r) \wedge gap(g,r)$
$\supset (\exists\ t',s_0)\ [last(t',s_0) \wedge ends(t,t')$
$\wedge everynthp(s,s_0,map2p(freq(r)),g)]]$

"map2p" is a function that maps from temporal units to their corresponding unary predicate names. For example, map2p(*year*) = year1, where $(\forall\ y)\ [year1(y) \equiv (\exists\ n,x)\ [calInt(y,n,*Year*,x)]]$

iCalendar can express very complicated recurrence sets. For example, "The 1st and 2nd hours of the 4th and 5th months of every year" will have a recurrence rule in which

BYHOUR = 1, 2, BYMONTH = 4, 5, and FREQ = YEARLY.

To formalize this we need recursion through the sequence of temporal units (with the predicate "everyithtempunit") and recursion through the list of integers associated with each temporal unit (with the predicate "byTulistRecurs"). We do the latter first.

$$(\forall\ s,ls,r,tu,g)\ [\text{RecurrenceSetRule}(s,r) \wedge \text{bytempunit}\ (ls,r,tu) \wedge \text{gap}(g,r) \supset (\exists\ s',s_0)\ [\text{everynthp}(s',s_0,\text{map2p}(\text{freq}(r)),g) \wedge \text{byTulistRecurs}(s,ls,s',tu,\text{freq}(r))]]$$

$$(\forall\ s,ls,s',tu,tu')\ [\text{byTulistRecurs}\ (s,ls,s',tu,tu') \wedge \text{card}(ls) > 1 \supset (\exists\ s_1,ls_1,i,si,i)\ [ls = \text{union}(\{i\},ls_1) \wedge \sim\text{member}(i,ls_1) \wedge \text{everyithtempunit}(si,s',i,tu,tu') \wedge s = \text{union}(s_1,si) \wedge \text{byTulistRecurs}(s_1,ls_1,s',tu,tu')]]$$

Base case of the recursion:

$$(\forall\ s,s',i,tu,tu')\ [\text{byTulistRecurs}(s,\{i\},s',tu,tu') \supset \text{everyithtempunit}(s,s',i,tu,tu')]$$

"bytempunit" is a generalized relation from bysecond, byminute, and so on. It is a relation among a recurrence rule *r*, a temporal unit *tu*, and a list of values associated with BYxxx with that temporal unit. For example,

$$(\forall\ ls,r)\ [\text{bytempunit}(ls,r,\text{*Second*}) \equiv \text{bysecond}(ls,r)]$$

"byTulistRecurs" is a relation among two temporal sequences (*s*, *s'*), their temporal units (*tu*, *tu'*), and a list of values associated with *tu*.

"everyithtempunit" is an important relation among two temporal sequences (*s*, *s'*), their temporal units (*tu*, *tu'*), and an integer value *i*. For example, everyithtempunit(s,s', 2,*Month*,*Year*) specifies the members of temporal sequence s are the every 2nd month, i.e. February, of the members (with temporal unit of *Year*) of temporal sequence s'.

It's possible to have a gap between FREQ and BYxxx, for example, "the 1st two hours in every month", which can be expressed in iCalendar with FREQ = MONTHLY, BYHOUR = 1, 2. To paraphrase this sentence, we get "the 1st two hours of the 1st day of every month". In order to handle this case, we define the predicate "everyithtempunit" recursively:

$$(\forall\ s_1,s_2,i,tu_1,tu_2)\ [\text{everyithtempunit}\ (s_1,s_2,i,tu_1,tu_2) \wedge \text{tulevel}(tu_2) > \text{tulevel}(tu_1)+1 \wedge tu_1 \neq \text{*Monthday*} \wedge tu_1 \neq \text{*Yearday*} \wedge tu_1 \neq \text{*Week*} \supset (\exists\ s')\ [\text{everyithtempunit}\ (s_1,s',i,tu_1,\text{level2tu}(\text{tulevel}(tu_2)-1)) \wedge \text{everyithtempunit}\ (s',s_2,1,\text{level2tu}(\text{tulevel}(tu_2)-1),tu_2)]]$$

Base case:

$$(\forall\ s_1,s_2,i,tu_1,tu_2)\ [\text{everyithtempunit}\ (s_1,s_2,i,tu_1,tu_2) \wedge \text{tulevel}(tu_2)=\text{tulevel}(tu_1)+1 \wedge tu_1 \neq \text{*Monthday*} \wedge tu_1 \neq \text{*Yearday*} \wedge tu_1 \neq \text{*Week*} \supset (\forall\ t_1)\ [\text{member}(t_1,s1) \equiv (\exists\ t_2)\ [\text{member}(t_2,s_2) \wedge \text{iinside-1}(t_1,t_2) \wedge \text{calclockInt}(t_1,i,tu_1,t_2)]]]$$

where $\text{calclockInt}(y,n,u,x) \equiv \text{calInt}(y,n,u,x) \vee \text{clockInt}(y,n,u,x)$

"tulevel" is a function mapping from a temporal unit to its level value according to a hierarchy. For example, tulevel(*second*) = 1, tulevel(*month*) = 5.

"level2tu" is an inverse function of "tulevel". It maps from the level value to the associated temporal unit. For example, level2tu(1) = *second*, level2tu(5) = *month*.

Month days, year days, and weeks are axiomatized specially.

iCalendar allows one to specify lists of dates/times as well, with the attributes RDATE and EXDATE. These are translated into simple temporal sequences. A recurrence set is generated by generating recurrence sets specified by the RRULE and/or RDATE attributes and by the EXRULE and/or EXDATE attributes, subtracting the latter from the former, and constraining the result by the DTSTART attribute.

Some Natural Language Examples

In the previous section, we've shown how iCalendar statements can be systematically mapped to OWL-Time predicates; in this section we demonstrate how these predicates in OWL-Time can be used to express natural language expressions.

We will first take an example from iCalendar, and show how to express it in OWL-Time. Then we will show an example that OWL-Time can do better than iCalendar. After showing an example that iCalendar can't handle, but OWL-Time can, we will finally show more natural language expressions that can be represented using OWL-Time predicates.

1) "*Every other week on Monday, Wednesday and Friday until December 24, 1997, but starting on Tuesday, September 2, 1997.*" (Taken from RFC 2445 page 120.)

```
DTSTART;TZID=US-Eastern:19970902T090000
RRULE:FREQ=WEEKLY;INTERVAL=2;UNTIL=19971224
T000000Z;WKST=SU;BYDAY=MO,WE,FR
```

iCalendar uses "FREQ=WEEKLY;INTERVAL=2" to represent "every other week", and uses "BYDAY=MO, WE,FR" to get "every Monday, Wednesday, and Friday". WKST specifies the start of the week, which is Sunday in this example.

This example can be expressed in OWL-Time as a temporal sequence *s* for which the following is true:

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

where $(\forall\ w)\ [\text{Week1}(w) \equiv (\exists\ n,x)\ [\text{calInt}(w,n,\text{*Week*},x)]]$

s is the desired temporal sequence representing "every Monday, Wednesday, and Friday" of *s'* which is a temporal sequence of "every other week from 09/02/1997 to 12/24/1997".

2) "*Every 3rd Monday in 2001.*"

This looks like a simple example, but iCalendar can't express it exactly. Though iCalendar can express "the 3rd Monday" using BYDAY=3MO, it can't express "every 3rd Monday". In order to express this phrase in iCalendar, we have to paraphrase it first to: "Every 3rd week on Monday in 2001".

However, the paraphrased one is not exactly the same as the original, since it depends on what the first week is. In iCalendar, the first week is defined as the week that "contains at least four days in that calendar year". Based on this definition, however, if the first week starts from a Tuesday, the first 3rd Monday will be different from the first Monday of the 3rd week!

In order to express year 2001, in iCalendar it has to specify the start date (01/01/2001) using DTSTART and the end date (12/31/2001) using UNTIL. Here is how to express "Every 3rd week on Monday in 2001" in iCalendar:

DTSTART;TZID=US-Eastern:20010101T000000
RRULE:FREQ=WEEKLY;INTERVAL=2;UNTIL=20011231
T000000Z;WKST=SU;BYDAY=MO

In OWL-Time, this example can be expressed very straightforwardly as a temporal sequence *s* for which the following is true:

$(\exists\ y,z)\ [yr(y,2001,CE(z)) \wedge everynthp(s,\{y\},Monday1,3)]$

where $(\forall\ d)\ [Monday1(d) \equiv (\exists\ w)\ [Monday(d,w)]]$

3) "*Every Monday that's a holiday.*"
There is no way in iCalendar to express "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.

OWL-Time, however, can express any kind of temporal aggregates, using the argument *p* in "everynthp(s,s_0,p,n)" or "everyp(s,s_0,p)". All the conditions and non-temporal-unit concepts can be captured using this *p*.

For example, we can express "Every Monday that's a holiday" in OWL-Time as a temporal sequence *s* for which the following is true:

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

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

4) More natural language expressions represented in OWL-Time are as follows, where *s* is the set corresponding to the noun phrase:

- "*The past three Mondays.*"

$(\exists\ s,s_0,t)\ [everyp(s,s_0,Monday1) \wedge card(s) = 3 \wedge last(t,s_0) \wedge ends(nowfn(D),t)]$

In our treatment of temporal deictics, the function "nowfn" maps a document d into the instant or interval viewed as "now" from the point of view of that document, and D is the document this phrase occurs in.

- "*Every other Monday in every 4th month in every year.*"

$(\exists\ s,s_1,s_2,s_0)\ [everyp(s_1,s_0,Year1) \wedge everynthp(s_2,s_1,Month1,4) \wedge everynthp(s,s_2,Monday1,2)]$

where $(\forall\ y)\ [Year1(y) \equiv (\exists\ n,x)\ [calInt(y,n,*Year*,x)]]$
$(\forall\ m)\ [Month1(m) \equiv (\exists\ n,x)\ [calInt(m,n,*Month*,x)]]$

- "*Four consecutive Mondays.*"

$(\exists\ s,s_0)\ [everyp(s,s_0,Monday1) \wedge card(s) = 4]$

In order to translate from natural language sentences to our representation, we first run a semantic parser to get a "surface" logical form from a given sentence. Then we use some additional rules to map from the "surface" logical form to our domain ontology. For example, given the above input sentence, the "surface" logical form would be:

$(\exists\ s,d)\ [plur(d,s) \wedge Monday1(d) \wedge consecutive(s,Monday1) \wedge card(s) = 4]$

"plur" takes as its arguments both a set and a representative member of the set. "consecutive" takes as its arguments both a set and the property of the set, meaning that the members of the set are not only consecutive but also share a certain property. "card" is a function that returns the cardinality/size of the set.

Then we map the above "surface" logical form to our ontology predicates by applying the following rule:

$(\forall\ s,p)consecutive(s,\ p) \equiv (\exists\ s_0)\ everyp(s,s_0,p)$

Conclusion

In this paper, we have described our work of representing temporal aggregates in OWL-Time, and showed how iCalendar recurrence sets can be embedded in OWL-Time. Examples are also showed to illustrate how it can be used to represent natural language expressions for temporal aggregates.

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.

References

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

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

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.