Temporal Arithmetic Mixing Months and Days

Feng Pan and Jerry R. Hobbs
Information Sciences Institute (ISI), University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
{pan, hobbs}@isi.edu

Abstract

In this paper, we present our work on creating a complete set of rules for temporal arithmetic mixing months and days based on the "history-dependent intuition". Many examples are presented for demonstrating how the rules are used and how the computations satisfy various desired arithmetic properties, such as the subtraction, commutativity and associativity properties. A notion of "days lost" (DL) is proposed to concisely keep track of the history of the temporal arithmetic computation and explain possible inconsistencies in terms of different desired properties.

1. Introduction

Temporal arithmetic involves the computation of adding durations to dates/times, subtracting durations from dates/times, and computing durations between date/time pairs. Such arithmetic (sometimes is also referred as date arithmetic when only dates are of interest) is very useful in many different domains, such as artificial intelligence (Bettini et al., 2002), databases (Chandra and Segev, 1994), time series management systems (Dreyer et al., 1994), agents (Mallya et al., 2004), and Web services (Pan and Hobbs, 2004).

As long as we stay within the year-month system or the day-hour-minute-second system, temporal arithmetic is just arithmetic and requires only a few simple axioms or rules to encode. When we mix months and days, problems arise. For example, consider that January 31, 2006 plus 2 months equals March 31, 2006. But if we add the months one at a time, we may get a different result: January 31, 2006 plus one month is February 28, 2006, but February 28, 2006 plus one month would seem to be March 28, 2006. If we want to avoid results like this, we need, in some sense, to keep track of the history of the computation.

This *motivating example* is taken from the documentation for Java methods add() and roll() of the class Calendar[1], where they explained the behavior of the methods using the above example and claimed that, "as most users will intuitively expect", the final date after adding the second month should be March 31, 2006, not March 28, 2006, and this is what you will get when you use the temporal arithmetic methods in Java. (In what follows, we will refer to this kind of intuition as the "*history-dependent intuition*".)

However, different people may have different intuitions regarding how the computation should work. Some people may believe adding one month to February 28 should always be March 28, no matter where February 28 was originally computed from. (In what follows, we will refer to this kind of intuition as the "*history-independent intuition*".) This intuition results in much simpler arithmetic rules, where when adding durations to dates, simply add each of their fields (temporal units) respectively, and handle overflow when needed. In fact, this approach is what XML Schema is currently using in their algorithm for adding "durations" to "dateTimes"[2] (Biron and Malhotra, 2004). But they didn't consider at all the alternative (maybe even more popular) intuition (the "history-dependent intuition") we described in the above motivating example.

Biron and Malhotra (2004) also warn that there are cases where the properties of *commutativity* and *associativity* cannot hold using the algorithm in XML Schema, and give one example as follows:

(2000-03-30 + P1D) + P1M
= 2000-03-31 + P1M = 2000-04-30

(2000-03-30 + P1M) + P1D
= 2000-04-30 + P1D = 2000-05-01

[1] http://java.sun.com/j2se/1.4.2/docs/api/java/util/Calendar.html
[2] http://www.w3.org/TR/xmlschema-2/#adding-durations-to-dateTimes

This example shows that adding one month (P1M) and one day (P1D) to March 30, 2000 in different orders using their algorithm would give different destination dates.

Many real-world applications, on the other hand, use a fixed number of days for each month (e.g., car rental companies usually count 28 days as one month), but they also cannot avoid the problem raised when mixing months and days. For example, Stonebraker (1990) pointed out that the yield calculation on financial bonds uses a calendar that has 30 days in every month for date arithmetic, but 365 days in the year for the actual yield calculation, which inevitably causes inconsistency.

To avoid this problem, Malhotra et. al. (2005) proposed, for XQuery and XPath, deriving two new (totally ordered) subtypes, "yearMonthDuration" (in the year-month system) and "dayTimeDuration" (in the day-hour-minute-second system), from the (partially ordered) datatype "duration". But in this case, temporal arithmetic can only be carried out in the two separate systems, and months and days cannot be computed together.

Both the "history-independent intuition" and the approach with a fixed number of days for each month can be encoded straightforwardly with a few simple axioms or rules. However, to our knowledge, there has been no serious effort for doing temporal reasoning based on the "history-dependent intuition", which may be the most popular intuition, but the most difficult to model.

In this paper, we present our work on creating a complete set of rules that can handle temporal arithmetic mixing months and days for the "history-dependent intuition" (including the motivating example), proposing a notion (called "days lost" DL) to concisely keep track of the history of the computation and explain possible inconsistencies in terms of different desired arithmetic properties. In Section 2 we describe these desired properties for temporal arithmetic computation. The notion of "days lost" is introduced in Section 3, and in Section 4 we present the temporal arithmetic rules with examples. Finally we discuss our future work in Section 5.

Since the problem arise only when months and days are mixed, in this paper we only consider dates with months and days, and all other fields of a date/time (e.g., year, hour) will be omitted in the examples and rules. First, we introduce some notation and assumptions for later examples and rules.

Notation:

- $(m, d)^{DL}$ denotes a *date* with a month "m", a day "d", and days lost "DL" (see Section 3 for the details on "days lost"). For example, $(2, 28)^3$ means February 28th with 3 days lost. DL is omitted when DL = 0.
- [m, d] denotes a *duration* with "m" months and "d" days. For example, [3, 2] means 3 months and 2 days.
- #(m) denotes the number of days in a month "m". For example #(3) = 31, since there are 31 days in March.

Assumptions:

- The year is 2006, so that the number of days in February is 28.
- Dates and durations are in canonical forms, i.e.,
  For dates: $1 \le m \le 12$
  $1 \le d \le$ last day of the current month, #(m)
  For durations: $1 \le m < 12$
  $1 \le d <$ last day of the destination month

2. Desired Properties for Temporal Arithmetic

Before creating the rules for temporal arithmetic computation satisfying the "history-dependent intuition", a list of desired properties which we hope can hold during the computation are presented first, and these properties were also used to guide the creation of the rules. The desired properties are as follows, with the most desired ones at the top.

(1) Motivating example for the "history-dependent intuition":

$(1, 31) + [1, 0] + [1, 0]$
$= (2, 28) + [1, 0] = (3, 31)$

(2) Addition-Simplicity property:

$(m_1, d_1) + [m_2, d_2] = (m_1+m_2, d_1+d_2)$,
if $d_1+d_2 \le \#(m_1+m_2)$, $m_1+m_2 \le 12$

(3) Subtraction-Simplicity property:

$(m_1, d_1) - [m_2, d_2] = (m_1-m_2, d_1-d_2)$,
if $m_1 > m_2$, $d_1 > d_2$, $d_1-d_2 \le \#(m_1-m_2)$

(4) Subtraction property:

$(m_1, d_1) + [m_2, d_2] = (m_3, d_3)$
$\Leftrightarrow (m_3, d_3) - [m_2, d_2] = (m_1, d_1)$
$\Leftrightarrow (m_3, d_3) - (m_1, d_1) = [m_2, d_2]$

(5) Commutativity property:

$(m_1, d_1) + [m_2, d_2] + [m_3, d_3]$
$= (m_1, d_1) + [m_3, d_3] + [m_2, d_2]$

(6) Associativity property:

$\{(m_1, d_1) + [m_2, d_2]\} + [m_3, d_3]$
$= (m_1, d_1) + \{[m_2, d_2] + [m_3, d_3]\}$

Property (1) is actually an instance of associativity, because

$\{(1, 31) + [1, 0]\} + [1, 0]$
$= (2, 28) + [1, 0] = (3, 31)$

(1, 31) + {[1, 0] + [1, 0]}
= (1, 31) + [2, 0] = (3, 31)

3. Meaning of "Days Lost (DL)"

In order to keep track of the history of the temporal arithmetic computation for the "history-dependent intuition", we introduce a notion, called "days lost (DL)". Its meaning is as follows:

(1) If the day of the month is the end of the month (e.g., $(2, 28)^3$), the days lost ("DL") means the number of days lost in the current month. For example, adding 1 month to January 31st results in February 28th, and 3 days are "lost" in the computation. Thus we use $(2, 28)^3$ to "remember" that 3 days were lost in February.

(2) If the day of the month is not the end of the month (e.g., $(3, 2)^3$), the days lost ("DL") is just a record of the number of days lost in the past. For example, adding 1 month and 2 days to January 31st results in March 2nd, where 3 days were "lost" in February.

In fact, "days lost (DL)" not only is used to keep track of the history of the computation, but also plays a crucial role in explaining the inconsistency of the computation results with respect to different desired properties (e.g., commutativity and associativity). For example, it can be used to explain the inconsistency described in the introduction regarding the temporal arithmetic algorithm in XML Schema (Biron and Malhotra, 2004):

(2000-03-30 + P1D) + P1M
= 2000-03-31 + P1M = 2000-04-30

(2000-03-30 + P1M) + P1D
= 2000-04-30 + P1D = 2000-05-01

The different results are actually due to the "one day lost" in the first computation when adding 1 month (P1M) to 2000-03-31, since there are only 30 days (not 31 days) in April. We will show in the next section how our rules can be used to deal with such inconsistency.

4. Temporal Arithmetic Rules

In this section, we describe a complete set of rules we created for temporal arithmetic satisfying the "history-dependent intuition", including adding durations to dates, subtracting durations from dates, and computing duration between two dates. Each rule is presented with one representative example, and additional examples are shown to demonstrate how to use multiple rules together in the computation.

All the rules can be straightforwardly translated into first-order logic (FOL) axioms, so that they can be incorporated into the OWL-Time (formerly DAML-Time) ontology (Hobbs and Pan, 2004) to give them access to the full ontology of time for temporal reasoning, including representation and reasoning about temporal relations, date-time information, durations, and temporal aggregates (Pan and Hobbs, 2005) such as recurrence events. One FOL translation of a temporal arithmetic rule is shown at the end of the section.

4.1. Adding Durations to Dates

There are three sets of rules for adding durations to dates: *add-decompose*, *add-month*, and *add-day*. In the following table, the rules are shown in the left column with index numbers (e.g., (1.1)), and one representative example is shown for each line of the rules in the right column for the demonstration and justification purpose.

Temporal Arithmetic Rule	**Example**
(1) Add-decompose:	
$(m_1,d_1)^{DL}+[m_2,d_2]$	$(2,28)^2+[1,2]$
(1.1) $=(m_1,d_1)^{DL}+[m_2,0]+[0,d_2]$	$=(2,28)^2+[1,0]+[0,2]$
(2) Add-month: // $N = \#(m_1+m_2)$	
$(m_1, d_1)^{DL} + [m_2, 0]$	
= if $(d_1 = \#(m_1))$	
(2.1) $(m_1+m_2, d_1+DL), d_1+DL \leq N$	$(2,28)^2+[2,0] = (4,30)$
(2.2) $(m_1+m_2,N)^{d1+DL-N}, d_1+DL>N$	$(3,31)+[1,0] = (4,30)^1$
else (i.e., $d_1 < \#(m_1)$)	
(2.3) (m_1+m_2, d_1) , $d_1 \leq N$	$(3, 2) + [1, 0] = (4, 2)$
(2.4) $(m_1+m_2, N)^{d1-N}$, $d_1 > N$	$(1,30) + [1,0] = (2,28)^2$
(3) Add-day: // $N = \#(m_1)$	
$(m_1, d_1)^{DL} + [0, d_2]$	
= if $(d_1 = \#(m_1))$	
(3.1) $(m_1+1, d_2)^{DL}$	$(2,28)^1 + [0,1] = (3,1)^1$
else (i.e., $d_1 < \#(m_1)$)	
(3.2) $(m_1, d_1+d_2)^{DL}$, $d_1+d_2 < N$	$(3,2)^3+[0,20] = (3,22)^3$
(3.3) (m_1, d_1+d_2), $d_1+d_2 = N$	$(5,5) + [0,26] = (5,31)$
(3.4) (m_1+1, d_1+d_2-N), $d_1+d_2 > N$	$(2,20) + [0,10] = (3,2)$

For any given duration, before it adds to a date, *add-decompose* rule is applied first to separate the months and days by decomposing the duration into two sub-durations with only months and days respectively, and then add months first to the date using *add-month* rules, then the days using *add-day* rules.

Two more examples of adding durations to dates are also shown below. The first one is from the motivating example, and the second one demonstrates the complete procedure (3 steps) for adding durations to dates, when durations have both months and days (the rule applied for each step of the computation is also shown with a corresponding rule index number):

Ex.1. $(1, 31) + [1, 0] = (2, 28)^3$ // add-month (2.2)
$(2, 28)^3 + [1, 0] = (3, 31)$ // add-month (2.1)

Ex.2. $(1, 29) + [1, 2]$
$= (1, 29) + [1, 0] + [0, 2]$ // add-decompose (1.1)
$= (2, 28)^1 + [0, 2]$ // add-month (2.4)
$= (3, 2)^1$ // add-day (3.1)

4.2. Subtracting Durations from Dates

There are three sets of rules for subtracting durations from dates: *sub-decompose*, *sub-day*, and *sub-month*:

Temporal Arithmetic Rule	Example
(1) Sub-decompose	
$(m_1, d_1)^{DL} - [m_2, d_2]$	$(2,28)^2$-[1,2]
(1.1) = $(m_1,d_1)^{DL} - [0,d_2] - [m_2,0]$	= $(2,28)^2$-[0,2]-[1,0]
(2) Sub-day	
$(m1, d_1)^{DL} - [0, d_2]$	
(2.1) = $(m_1, d_1-d_2)^{DL}$, $d_1 > d_2$	$(4,30)^1$–[0,15]=$(4,15)^1$
(2.2) $(m_1-1,d_1+\#(m_1-1)-d_2)^{DL}$, $d_1 \le d_2$	$(3,2)^3$–[0,10] = $(2,20)^3$
(3) Sub-month // N = #(m_1-m_2)	
$(m_1, d_1)^{DL} - [m_2, 0]$	
= if (d_1 = #(m_1))	
(3.1) (m_1-m_2, d_1+DL), d_1+DL ≤ N	$(2,28)^2$ – [1,0] = (1,30)
(3.2) (m_1-m_2,N)$^{d1+DL-N}$,d_1+DL>N	$(4,30)^1$ – [2,0] = $(2,28)^3$
else (i.e., d_1 < #(m_1))	
(3.3) (m_1-m_2, N)$^{d1-N}$, d_1 > N	$(3,30)^2$ – [1,0] = $(2,28)^2$
(3.4) (m_1-m_2, d1), d_1 ≤ N	(5,16) – [1,0] = (4,16)

For any given duration, before it subtracts from a date, *sub-decompose* rule is applied first to separate the months and days by decomposing the duration into two sub-durations with only days and months respectively, and then subtract days first from the date using *sub-day* rules, then the months using *sub-month* rules.

Two more examples of subtracting durations from dates are also shown below. The first one is from the motivating example, and the second one demonstrates the complete procedure (3 steps) for subtracting durations from dates, when durations have both months and days:

Ex.3. $(3, 31) - [1, 0] = (2, 28)^3$ // sub-month (3.2)
$(2, 28)^3 - [1, 0] = (1, 31)$ // sub-month (3.1)

Ex.1. and Ex.3. show that the temporal arithmetic rules work for the *motivating example* for the "history-dependent intuition" (desired property 1), and the first part of the "*subtraction property*" (desired property 2) also holds for the motivating example, because

$(1,31) + [1,0] = (2,28)^3$ <=> $(2,28)^3 - [1,0] = (1,31)$
$(2,28)^3 + [1,0] = (3,31)$ <=> $(3,31) - [1,0] = (2,28)^3$
$(1, 31) + [1, 0] + [1, 0] = (2, 28)^3 + [1, 0] = (3, 31)$

Ex.4. $(3, 2)^1 - [1, 2]$
$= (3, 2)^1 - [0, 2] - [1, 0]$ // sub-decompose (1.1)
$= (2, 28)^1 - [1, 0]$ // sub-day (2.2)
$= (1, 29)$ // sub-month (3.1)

Ex.2. and Ex.4. also show that the first part of the "*subtraction property*" (desired property 2) holds for this example whose duration has both months and days, because
$(1,29) + [1,2] = (3,2)^1$ <=> $(3,2)^1 - [1,2] = (1,29)$

4.3. Computing Duration between Two Dates

There is only one set of rules for computing duration between two dates:

Temporal Arithmetic Rule	Example
(1) Sub-between-dates	
$(m_2, d_2)^{DL2} - (m_1, d_1)^{DL1}$	
= if (d_1 = #(m_1))	
(1.1) $[m_2-m_1, d_2-(d_1+DL_1)]$, $d_2 >= d_1+DL_1$	(5,31)-$(2,28)^2$=[3,1]
(1.2) $[m_2-m_1-1,(d_2+DL_2)+\#(m_2-1)$ $-(d_1+DL_1)]$, $d_2<d_1+DL_1$, $d_2<\#(m_2)$	$(3,2)^3$-(1,31)=[1,2]

(1.3) $[m_2\text{-}m_1, 0], d_2 < d_1 + DL_1, d_2 = \#(m_2)$	$(4,30)^1\text{-}(2,28)^3$
else (i.e., $d_1 < \#(m_1)$)	= [2,0]
(1.4) $[m_2\text{-}m_1, d2\text{-}d_1], d_2 >= d_1$	(3,20)-(2,10)
	=[1, 10]
(1.5) $[m_2\text{-}m_1\text{-}1, d2]$,	(3,2) - (1,28) = [1,2]
$d_2 < d_1, d_2 < \#(m_2), \#(m_2\text{-}1) \le d_1$	
(1.6) $[m_2\text{-}m_1\text{-}1, d_2+\#(m_2\text{-}1)\text{-}d_1]$,	(3,2)-(1,20) = [1,10]
$d_2 < d_1, d_2 < \#(m_2), \#(m_2\text{-}1) > d_1$	
(1.7) $[m_2\text{-}m_1, 0], d_2 < d_1, d_2 = \#(m_2)$	$(2,28)^1\text{-}(1,29)$=[1,0]

Two more examples of computing duration between two dates are also shown below. The first one is from the motivating example, and the second one demonstrates the procedure for computing durations between two dates, when durations have both months and days:

Ex.5. $(3,31) - (2,28)^3 = [1,0]$ // sub-between-dates (1.1)
$(2,28)^3 - (1,31) = [1,0]$ // sub-between-dates (1.3)
$(3,31) - (1,31) = [2,0]$ // sub-between-dates (1.1)

Ex.1., Ex.3., and Ex.5. show the complete "*subtraction property*" (desired property 2) holds for the motivating example, because

$(1, 31) + [1, 0] = (2, 28)^3$
<=> $(2, 28)^3 - [1, 0] = (1, 31)$
<=> $(2, 28)^3 - (1, 31) = [1, 0]$

$(2, 28)^3 + [1, 0] = (3, 31)$
<=> $(3, 31) - [1, 0] = (2, 28)^3$
<=> $(3, 31) - (2, 28)^3 = [1, 0]$

They also show the "*associativity property*" holds for the motivating example, because

$\{(1, 31) + [1, 0]\} + [1, 0] = (3, 31)$
<=> $(1, 31) + \{[1, 0] + [1, 0]\}$
$= (1, 31) + [2, 0] = (3, 31)$

Ex.6. $(3,2)^1 - (1,29) = [1,2]$ // sub-between-dates (1.5)

Ex.2., Ex.4., and Ex.6. also show that the complete "*subtraction property*" (desired property 2) holds for this example, since

$(1, 29) + [1, 2] = (3, 2)^1$
<=> $(3, 2)^1 - [1, 2] = (1, 29)$
<=> $(3, 2)^1 - (1, 29) = [1, 2]$

Ex.7. $(1, 31) + [1, 2] + [2, 0]$
$= (1, 31) + [1, 0] + [0, 2] + [2, 0]$
// add-decompose (1.1)
$= (2, 28)^3 + [0, 2] + [2, 0]$ // add-month (2.2)
$= (3, 2)^3 + [2, 0]$ // add-day (3.1)
$= (5, 2)$ // add-month (3.4)

$(1, 31) + [2, 0] + [1, 2]$
$= (3, 31) + [1, 2]$ // add-month (2.1)
$= (3, 31) + [1, 0] + [0, 2]$ // add-decompose (1.1)
$= (4, 30)^1 + [0, 2]$ // add-month (2.2)
$= (5, 2)^1$ // add-day (3.1)

Ex.7. shows that the "*commutativity property*" holds for this more complicated example, if the supplemental information (days lost "DL") is ignored.

Ex.8. In the introduction, we showed the following example from XML Schema (Biron and Malhotra, 2004) that demonstrates the case where the properties of commutativity and associativity cannot hold using their temporal arithmetic algorithm:

(2000-03-30 + P1D) + P1M
= 2000-03-31 + P1M = 2000-04-30

(2000-03-30 + P1M) + P1D
= 2000-04-30 + P1D = 2000-05-01

Can our rules handle this problem? The computation is as follows.

$(3, 30) + [0, 1] + [1, 0]$
$= (3, 31) + [1, 0]$ // add-day (3.3)
$= (4, 30)^1$ // add-month (2.2)

$(3, 30) + [1, 0] + [0, 1]$
$= (4, 30) + [0, 1]$ // add-month (2.3)
$= (5, 1)$ // add-day (3.1)

The rules not only compute the results exactly based on the "history-dependent intuition", but also include supplemental information (days lost "DL") for explaining the reason why the commutativity property doesn't hold: there was "one day lost" in the first computation when one month is added to March 31.

In fact, the commutativity property would hold, if we specify that $(4, 30)^1$ "equals" to (5, 1). More generally, the commutativity and associativity properties hold with our temporal arithmetic rules, if we define the month-day equality "more softly" and take into account the effect of "days lost" (assume overflow is properly handled):

$(m_1, d_1)^{DL_1} = (m_2, d_2)^{DL_2}$, if $m_1 = m_2$
AND ($d_1 = d_2$ *OR* $d_1 + DL_1 = d_2 + DL_2$)

We have only given examples in this paper, but we believe and are in the process of demonstrating that the rules satisfy the desired properties in general, given this definition of month-day equality.

4.4. Translating Rules to FOL Axioms in OWL-Time

All the above temporal arithmetic rules can be straightforwardly translated into FOL axioms in OWL-Time to give them access to the full ontology of time for temporal reasoning. For example, the rule "add-month (2.1)"

$(m_1, d_1)^{DL} + [m_2, 0] = (m_1+m_2, d_1+DL)$,
if $d_1 = \#(m_1)$, $d_1+DL \leq \#(m1+m2)$

can be translated as (see (Hobbs and Pan, 2004) for the definitions of the predicates):

$(\forall\ T, t_1, t_3, m_1, m_2, m_3, d_1, d_3, DL, DL_3, n_1, n_3)$
$dateOf(t_1, m_1, d_1, DL) \wedge durationOf(T, m_2, 0)$
$\wedge\ begins(t_1, T) \wedge ends(t_3, T)$
$\wedge\ dateOf(t_3, m_3, d_3, DL_3) \wedge Hath(n_1, *Day*, m_1)$
$\wedge\ Hath(n_3, *Day*, m_3) \wedge d_1 = n_1 \wedge d_1+DL \leq n_3$
$\supset m_3 = m_1+m_2 \wedge d_3 = d_1+DL \wedge DL_3 = 0$

5. Conclusions

In this paper, we have presented our work on creating a complete set of rules for temporal arithmetic mixing months and days, based on the "history-dependent intuition". A notion of "days lost" (DL) was proposed for both keeping track of the history of the computation and explaining possible inconsistencies in terms of different desired arithmetic properties, such as the subtraction, commutativity and associativity properties.

Acknowledgement

This work was supported by the Advanced Research and Development Activity (ARDA), now the Disruptive Technology Office (DTO), 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. The authors have profited from discussions with José Luis Ambite, Hans Chalupsky, Kevin Knight, Rutu Mulkar, Daniel O'Leary, Paul Rosenbloom, and Tom Russ.

Reference

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

Biron, P. V. and Malhotra, A. 2004. XML Schema Part 2: Datatypes Second Edition. *W3C Recommendation*. http://www.w3.org/TR/xmlschema-2/

Chandra, R., Segev, A., and Stonebraker, M. 1994. Implementing Calendars and Temporal Rules in Next Generation Databases". In *Proceedings of the Tenth International Conference on Data Engineering*, pages 264–273, Houston, Texas.

Dreyer, W, Dittrich, A. Koa, and Schmidt, D. 1994. Research Perspectives for Time Series Management Systems. *SIGMOD RECORD*, Vol. 23, No. 1, pp. 10-15.

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

Malhotra, A, Melton, J., and Walsh, N. 2005. XQuery 1.0 and XPath 2.0 Functions and Operators. *W3C Candidate Recommendation*. http://www.w3.org/TR/xpath-functions/

Mallya, A. U., Yolum, P., and Singh, M. P. 2004. Resolving commitments among autonomous agents. In F. Dignum, editor, Advances in Agent Communication, In *Proceedings of the International Workshop on Agent Communication Languages*, Germany.

Pan, F. and Hobbs, J. R. 2004. Time in OWL-S. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, Stanford University, California, pp. 29-36, AAAI Press.

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.

Stonebraker, M. R. Extensibility. 1990. In M.R. Stonebraker, editor, *Readings in Database Systems*. Morgan Kaufman.