On Reasoning on Time and Location on the Web

François Bry, Bernhard Lorenz, Hans Jürgen Ohlbach, Stephanie Spranger

University of Munich, Germany,
http://www.pms.informatik.uni-muenchen.de

Abstract Reasoning on time and location is receiving increasing attention on the Web due to emerging fields like Web adaptation, mobile computing, and the Semantic Web. Web applications in these fields often refer to rather complex temporal, calendric, and location information. Unfortunately, today's Web languages and formalisms have merely primitive temporal and location data types and temporal and location reasoning capabilities – if any. This article reports on work in progress aiming at integrating temporal and locational reasoning into XML query and transformation operations. We analyze the problem and propose a concrete architecture. A prototype of the temporal reasoner, the WebCal system has already been realized.

1 Introduction

The next generation of the World-Wide-Web must have *machine readable* and *machine understandable* Web content, and no longer just html pages – this is the now widely agreed vision of the 'Semantic Web' initiative [5]. The basic tool for the first step, to make Web content machine readable, is undisputedly XML. The second step, however, to make Web content machine understandable, seems to become a never ending story. Almost anything, human beings can think of, can become Web content. Making all this machine understandable, means formalizing the whole knowledge of mankind in a way that computers can work with; and this is the old vision of Artificial Intelligence. One can attack this problem in at least two ways. The first approach is to develop XML compatible knowledge representation and reasoning tools, and to leave the concrete formalization of knowledge with these tools to the application developers. RDF and OWL are systems of this kind. The second approach is to develop formal theories and mechanisms of particular concepts occurring in Web content, and to integrate these theories into the other XML mechanisms, in particular into XML query and transformation languages. Both approaches are complementary to each other in the same way as general purpose inference systems in logic are complementary to special theory reasoning.
In this paper we propose the development of special theories and XML mechanisms for the concepts time and location, and the integration into XML query and transformation languages. Our proposal goes far beyond the temporal data types of the W3C standard XML Schema [1,2,3], which has in fact no reasoning mechanisms and no location data types and operations.

The time theory for XML we propose works with the WebCal system, a server for advanced calendrical calculations. The WebCal system is based on ideas first published in [13,14,15]. WebCal has an international dimension: it provides most of the calendar systems world-wide in use, with timezones, daylight savings time regulations, leap years, leap seconds etc. It also has an historical dimension: it models historical sequences of calendrical regulations, for example sequences of calendar systems, sequences of daylight savings time regulations, timezones changing over time etc. It has an application oriented dimension, i.e. one can define new application specific temporal notions, for example school holidays, financial years, ecclesiastical calendars, Mary's birthday, my own working hours, and a lot more. WebCal can deal with fuzzy notions like 'late night' or 'around noon' because it represents time intervals as fuzzy sets.
The location theory for XML is not yet as far developed as the time theory, but the solutions for the location theory will be in the same style as the solutions for the time theory.
We shall integrate these mechanisms into the rule based query and transformation language Xcerpt [7,8,6]. Xcerpt is a declarative (logic-based) query and transformation language for the Web currently developed and tested on web-based systems at the University of Munich. Because of its 'logic bias', Xcerpt appears to be a very convenient 'host' for temporal data types and temporal reasoning. Most of the ideas and methods proposed in this paper, however, are independent of Xcerpt and should work with any query language.
In order to illustrate the problems and approaches, we start with a small introductory example.

2 An Introductory Example

The first example illustrates the problems with temporal reasoning in the Web context. Quite similar problems come up with locational reasoning.
Suppose we have a cinema program as an XML document. The XML document could look like:

```
<cinema_program>
  <cinema name="Atlantis" location="Munich">
    <year year="2003">
       <month month="January">
         <day day="1"> <film>
                         <title>Lampedusa</title>
                         <start>20:00</start>
                         <duration>120min </duration>
                        </film>
                        <film>
                         <title>City of God</title>
                         <start>22:00</start>
                         <duration>121min </duration>
                       </film>
         </day> ... </cinema_program>
```

A reasonable query could now be "give me all films ending before midnight 1/1/2003". None of the currently available query languages can deal with such a query. What are the problems?

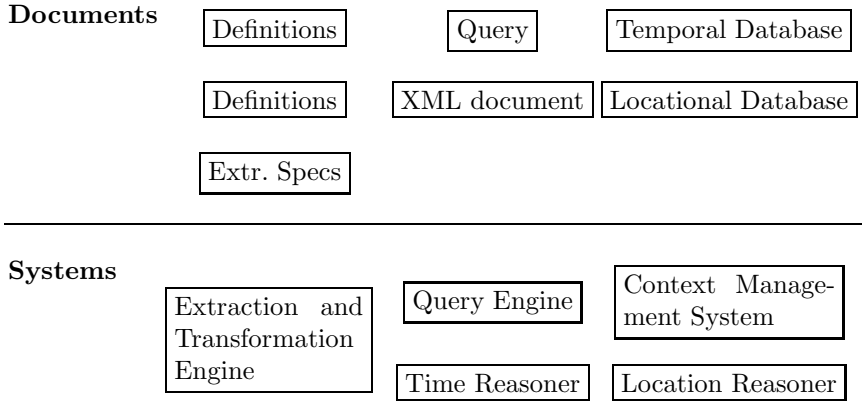
1. The end times are not explicitly represented in the XML document.
2. The end times could be computed, but the information needed for this is distributed over parts of the XML document.
3. The used calendar system is not mentioned. It could, however, be deduced from the location "Munich".
4. The notion 'midnight' in the query is not clear. Is midnight in Munich meant, or midnight at the place of the querying person, which could be in another continent.
5. Would we really want to exclude 'City of God' although it ends only one minute after midnight? What kind of relation is to be used to evaluate '24:01 before midnight'? A fuzzy 'before' relation could evaluate this expression to a non-zero fuzzy value, such that this film could also be included in the answer set, maybe at the end of the answer list.

In order to evaluate this query in the user's sense, we need a query language plus quite a number of extra mechanisms:

1. first of all we need a *context detection and managing mechanism*. In the example it must figure out the calendar system from the location 'Munich' (alternatively one could require to specify the calendar system in the XML document). It must also figure out the calendar system used in the query. In addition it must determine the timezone and the daylight savings time regulations – again from the location 'Munich' or from some explicit data in the XML document.
2. in order to compute from start time and duration the end time, we need a *time reasoner* which can compute from "year = 2003", "month = January", "day = 1", "start = 22:00" and "duration = 121min" the end time;
3. since the information to be fed into the time reasoner is distributed over the XML document, we need an *extraction language* with which we can specify where to get the necessary data from and how to transform it to become a valid input for the time reasoner;
4. since 'midnight' is just a string, we need a definition of 'midnight' in terms of concrete time points. To this end, a *specification language* for specifying application or user specific temporal notions is necessary;
5. the relation '24:01 before midnight' must be evaluated by the time reasoner, possibly with a fuzzy version of the 'before' relation.
6. the query language must use the fuzzy values to order the answers.

3 The Top Level Architecture

We propose the following top level architecture:



Let us explain each component in more detail.

The Definition Files. Both, the query and the XML document may use application or user specific temporal and locational notions. These notions must be defined in special *definition files*, and loaded into the time and location reasoners before they get invoked. Examples of defined temporal notions which might occur in XML documents are: '3 weeks after Easter', 'the first half or the semester', 'the third school hour', 'weekend', 'the time of the Olympic Games', 'late night', 'around noon' etc. Queries can contain similar expressions, but in addition very user specific expressions. For example, the user may define 'weekend' for himself different to the common notion of 'weekend'. Therefore the query and the XML document may need different definitions.
A few examples of definable *locational* notions which might occur in XML documents and queries are: 'in Munich', 'in the south of Munich', 'close to the station', 'along the A1' 'between Munich and Frankfurt'.
Since locations are 2 or 3-dimensional, there is a much greater variety of locational notions, than of temporal notions. One of the first jobs in the proposed project will therefore be to select classes of locational notions which can be processed in a reasonable way.
The definition files should not be part of the DTD or the XML schema because the notions defined there are not specific to one particular XML document type. With the definition files I can build up libraries of temporal notions which can be used for many different XML-documents. Therefore the definition files are more like style sheet files or JavaScript program files to be loaded together with the XML-document.

The Extraction Specifications. These specify how to extract and transform the data from the XML document to be submitted to the time and location reasoner. In the example of Section 2 we would specify something like: 'in order

to calculate the end time, extract the 'year' attribute, the 'month' attribute, the 'day' attribute, the 'start time', turn the 'month' attribute into a number and combine this to a string of the form year/month/day/hour/minute. Then extract the 'duration' element and split it into the duration number and the time unit. Now call the time reasoner to add the duration to the start time. (start hour and minute + duration does not always yield the proper end time, for example when summertime is changed to wintertime. Therefore the complete date and time is needed here.) The preparation of the input to the time reasoner may even require access to external reources. If for example in the cinema example the year is not explicitly stated in the document, one may get the current year as the default from the system clock. This very informal description must of course be put into a formal language. A rule based query language like Xcerpt might be appropriate here.

The Temporal Database. This database contains information which is not specific to any application or user. It need not be one single database, but it can be distributed and hierarchically organized. It must contain in particular mappings from countries and other areas to calendar systems, timezones, daylight savings time regulations. If we want to process XML documents with historical content, we may even need mappings from countries to historical sequences of calendar systems and other time measurement parameters.
It also can contain the dates of public and ecclesiastical holidays, school holidays, the dates of public events, and a lot more.

The Locational Database. This database could be a Geographic Information System with geographic information about the landscape, places, streets, parks, houses etc. It could also contain routes and timetables of public transport systems: buses, underground, trains, taxi stands etc.
Information like 'persons in a wheelchair cannot use bus line 263' could also be there, or heuristics of the kind 'single female persons should avoid Central Park at night'.

The Extraction and Transformation Engine. The Extraction and Transformation Engine processes the extraction specifications. In the example of Section 2 it would collect the relevant data for the start time and duration and turn it into suitable input for the time reasoner.

The Query Engine processes XML queries almost in the usual way. It moves down the document tree and does string comparisons and maybe evaluate regular expressions over strings. In addition it must also call the time and location reasoners for evaluating expressions like '24:01 before midnight'. If these expressions yield fuzzy values as a result, the query engine must use them to order the answers. This is quite easy if only one expression is involved. But consider the following query 'give me all large cinemas in the south of Munich'. 'large

cinemas' is a fuzzy expression and 'in the south of Munich' is a fuzzy expression. The evaluation of this query may yield for each cinema a pair of fuzzy values, and this pair has to be turned into a single total ordering. There may be extra parameters necessary in the query itself, which tell the evaluation engine whether and how to prefer larger cinemas over ones more in the south of Munich, or the other way round. Another alternative is to leave the answers unordered, but return the fuzzy values together with the answers. Then it is up to the application what to do with them. The special theories – time and location and others – should of course not be hardwired into the query language. Instead one should be able to load them as a kind of library into the language evaluator. This is much more flexible and extendible.

The Context Management System. This system is needed to configure the time and location reasoner. In particular, the time reasoner needs to know the calendar system, the timezone, the daylight savings time regulations, in addition to various application specific parameters. For example, on the northern hemisphere the seasons spring, summer, autumn and winter begin and end at different dates than on the southern hemisphere. It is the job of the context management system to turn the information it can get about the document, the application, and the user into corresponding configuration parameters for the time and location reasoner. In the age of mobile computing where the user of a Web Service can move his location (he drives in a car), or even change his device (he switches for the notebook to his PDA or his mobile phone), while accessing the Web service, the context becomes dynamic. It may therefore be necessary for the Context Management System to permanently monitor the query context. Dynamic context need not only come from users changing their location or device. In an ordinary session of a user with some system, for example a tutoring system, the context may change during the course of the session, just by the interactions of the user with the system. Therefore context management has a much wider scope than just tracking times and locations.

The Time Reasoner. The proposed time reasoner is the WebCal system, which is explained in more detail in Section 4.

The Location Reasoner. The main purpose of locational notions in XML documents is to relate locational aspects of objects and events to locational aspects of previously introduced objects or events, or to concrete coordinates. Compared to the one-dimensional time axis, there is a much greater variety of relations between 2- or 3-dimensional locational data. The basic relations between 2-dimensional regions have been investigated in the area of qualitative spatial reasoning [10,4,16]. In this area, sophisticated automated reasoning about the spatial relations between physical objects or regions of space has been investigated; and in many cases, this must be done without precise, quantitative information about these relations. Typically, some knowledge of the topological

relationships between the entities of interest may be available, along with incomplete and imprecise information about distances, directions and relative sizes; and from this partial information, useful conclusions must be drawn. Examples of the kind of question for which qualitative spatial reasoning is required are: identify the islands in the lake and the largest one. Which parts of the network of tunnels can the robot traverse without getting stuck? Could the collection of objects in the scene fit together to make a spherical or cylindrical shell? When cog A is turned clockwise, will cog B turn and if so, in which direction?
The kind of locational reasoning in the XML-context we want to investigate has a different focus. It is about relations between geographical objects, countries, cities, streets, footpaths, houses etc. which involve a *metric*. The metric on 2- or 3-dimensional locational data comes from the way objects can move from point A to point B. Suppose the query is 'give me the nearest cinema playing Terminator 3'. 'nearest' is here the crucial notion. It is very unlikely that it just refers to the geographical distance to my current location. Instead it very likely refers to the time I will need to get there, and maybe the money it will cost me. But this depends on a lot of different factors: am I walking or driving by bicycle, motorbike or car?; am I sitting in a wheelchair?; can I use public transport?; am I alone or with children or with a pram?; how old am I?; am I male or female?; can I park in a difficult parking lot?; is it advisable for a woman to cross a dark park at night? etc. Evaluating 'nearest' actually amounts to planning a route from my current location to the destination. Route planning for cars is not a difficult issue anymore. The difficulties come up when all the above mentioned possibilities should be taken into account. In particular for people without cars one has to combine quite a number of different networks into a search graph: streets, footpaths, bus lines, underground lines, tram lines etc.
One of the tasks of the location reasoner will be to evaluate 'nearest' in this sense. The location reasoner has to get all the relevant parameters from the context management system, combine the relevant networks into a search graph, and find the shortest path to the potential destinations (the cinemas playing Terminator 3 in the above example).

The XML Document and the Query. We want to put as few restrictions to the structure of an XML document with temporal and locational data as possible, but certain parts need to be stated in a formal language if the data is to be processed automatically. In the cinema program document above, for example, we used the XML element '<start>', but we could also have used '<start_time>' or something else. This freedom is possible because we have the extraction language, where we can specify where to look for the start time in the XML document. The specifications in the extraction language must follow the DTD or XML Schema specification of the XML document. If we want to process two different XML documents, where one uses '<start>' and the other uses '<start_time>', we need two different extraction specifications, which is not a good idea. One could think of an additional abstraction layer, where the difference between '<start>' and '<start_time>' becomes irrelevant, and the

matching to the concrete DTD is an extra step, but this has still to be investigated.

The extraction language allows us to leave the structure of the XML document free. The content of the XML elements and attributes with temporal and locational data, however, need to be written in a formal language. In particular we need to agree on formats for date and time strings, but this is only the most simple case. In the general case we may express temporal and locational notions by relating them to other temporal and locational notions. The spectrum of variations for this is almost infinite. Examples for temporal notions are 'three weeks after Easter', 'during my holidays', 'after the Iraq war', 'from around noon until early evening', 'at Mary's birthday' etc. Examples for locational notions are 'in Munich', 'along the A1', 'in the center of London', 'between Picadilly and Trafalgar Square', 'three miles from the river' etc. It would be extremely user friendly if we could put this as strings in natural language into the XML documents, but making the reasoners understanding its meaning is then almost impossible. Therefore we need a formal language to express these facts. The specification language of WebCal provides some basics for the temporal part of such a language, but it needs to be considerably extended for this purpose.

Modern designs of formal languages are *typed*. Types and type checking for a language not only help avoiding mistakes. They also can provide useful information for guiding the processing of the expressions. Therefore the formal language for the XML documents and for the queries will be typed. A first proposal for a type system for this purpose has been presented in [9]. Typical types are 'time point', 'time interval', 'duration' etc.

Another problem to be solved is deliberate ambiguity. Consider for example the string 'three weeks after Easter'. First of all, it does not contain the information about the year in which the term 'Easter' is to be evaluated. The year may be listed in another part of the XML document, such that the extraction language can make it precise by turning 'Easter' into for example 'Easter(2003)'. But this is still ambigue. It is not clear whether the western Easter date or the orthodox Easter date is meant. They may differ by a week. We may want to leave this ambigue in the XML document itself, and resolve the ambiguity by the query context. One querying person may want to interpret Easter as western Easter, and another person may want to interpret it as orthodox Easter.

A similar example with ambigue locational notions could be an XML element <location> capital </location>, where it is left open which capital of which country is meant. Only the query context may make this clear.

Deliberate ambiguity means that the formal language for the time and location dates cannot always refer to concrete events whose data can be taken from the temporal and locational databases or from the definition files. We must allow for an extra level of indirectness. This also means that this indirectness has to be eliminated in an extra step at query time, for example by transforming 'Easter(2003)' into 'Easter(2003,orthodox)'.

The time and location part of the query language is a little bit simpler than the time and location part in the XML documents. Ambigue queries where it is not

clear whether for example the western Easter dates or the orthodox Easter dates are meant, cannot be reasonably evaluated. Therefore we do not need the extra level of indirectness as in the XML document.

4 Temporal Reasoning with WebCal

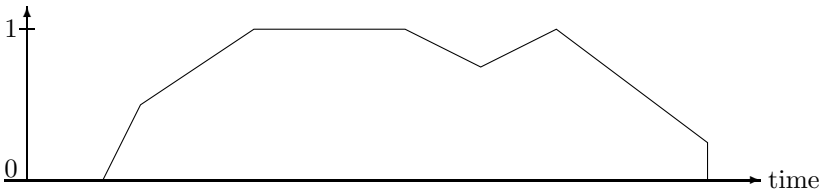
WebCal is a computer program which provides advanced calendrical calculations for Web services. WebCal has an international dimension: it provides most of the calendar systems world-wide in use, with timezones, daylight savings time regulations, leap years, leap seconds etc. It also has an historical dimension: it models historical sequences of calendrical regulations, for example sequences of calendar systems, sequences of daylight savings time regulations, timezones changing over time etc. It has an application oriented dimension, i.e. one can define new application specific temporal notions, for example school holidays, financial years, ecclesiastical calendars, Mary's birthday, my own working hours, and a lot more. WebCal can deal with fuzzy notions like 'late night' or 'around noon' because it represents time intervals as fuzzy sets. The program is based on an algebraic model of basic temporal notions, which gives all the operations a very precise semantics. The main idea behind WebCal is to provide a few powerful datastructures, and to offer applications as many operations as possible on these datastructures.

4.1 Fuzzy Time Intervals

All time information which is submitted to WebCal is immediately turned into intervals of reference time seconds. For example, the command 'parse 2000' (year in UTC) turns the year 2000 into the interval [946684831 978307231[which corresponds to the time interval from the beginning of the first second in the year 2000 till the end of the last second in the year 2000.

In order to be able to deal with fuzzy notions like 'around noon', WebCal represents all intervals as *fuzzy sets*. Even ordinary intervals are in fact fuzzy sets. For example, the above mentioned year 2000 is represented as a rectangular polygon: [946684831,0 946684831,1000 978307231,1000 0,0[. Here the number 1000 stands for the fuzzy value 1.0. We choose an integer representation for fuzzy values instead of a floating point representation because algorithms for polygons with integer coordinates are more efficient and less error prone than algorithms for polygons with floating point coordinates.

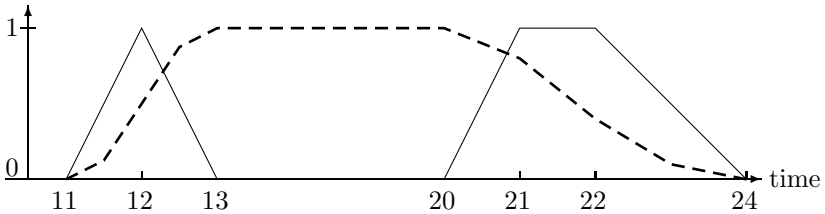
Admissible fuzzy intervals in WebCal are any polygons of the form $[x_1, y_1\ x_2, y_2 \ldots x_{n-1}, y_{n-1}\ x_n, y_n[$ where the x_i are integers in increasing order, and the y_i are integers between 0 and 1000.



Fuzzy Interval

If y_1 or y_n is not 0 then this represents an infinite fuzzy interval where y_1 or y_n respectively stretch the interval to the infinity. Fuzzy intervals can also be non-convex and therefore represent the union of separate intervals.
WebCal provides the following classes of operations on crisp and fuzzy time intervals:

- turning crisp intervals into fuzzy intervals by applying 'fuzzification functions';
- measuring various features of an interval, in particular its size as the integral over the polygon;
- different hull operations (monotone hull, convex hull, crisp hull);
- the usual set theoretic operations on fuzzy intervals, in particular union, intersection, complement, set difference. (For the expert: these operations can be parameterized with t-norms, t-conorms and negation functions);
- some other functions which turn fuzzy sets into fuzzy sets. As an example, consider an XML document about, say, the institute's birthday parties. It may contain the entry that the birthday party for the director took place 'from around noon until early evening' of 20/7/2003. 'Around noon' is a fuzzy notion and 'early evening' is a fuzzy notion. What is now the duration of the birthday party? It must obviously also be a fuzzy set. The fuzzy value of the birthday party duration at a time point x is 1 if the probability that the party started before x is 1 and the probability that the party ended after x is also 1. Therefore the fuzzy value at point x is computed by integrating over the probabilities of the start points and the end points. This is one of the operations the WebCal system provides. The resulting fuzzy set is:



Birthday Party Time form around noon until early evening

The dashed curve may for example represent the percentage of people at the party at a give time.
We can continue this example to illustrate another phenomenon when we reason about time in the XML context. Suppose my diary is also an XML

document and it contains the information 'I met this nice girl at the director's birthday party'. If I query the XML document 'did I meet a nice girl between 20 and 21 hours?' the system needs to determine the probability that I met the girl in this time. Since the girl was at the party, the 'meeting the girl' event is correlated with the party time. Therefore the probability that this happened is given by the integral over the party time's fuzzy value during this time period divided by the whole integral. If my diary contained in addition the information 'my mobile phone rang during the director's birthday party', this is an event which is not correlated with the party itself. Therefore the probability that it rang between 20 and 21 hours is just given by 1 hour /(maximal length of the party time).
All these operations are supported by WebCal;

- various relations between points and intervals. These relations yield again fuzzy values as results. Therefore a relation '12:01 before midnight', where midnight may for example be represented by a one hour interval, in fact yields a non-zero fuzzy value. In addition to the standard relations 'before', 'starting', 'during' etc. WebCal also provides relations like 'during the first third of the interval' or 'in the middle of the second half' etc.
- Fuzzy versions of Allen's interval relations 'before', 'starts', 'during', 'finishes', 'after', 'equal'. The relations are parameterized. The parameter controls how fuzzy the relation behaves. If the parameter is very large then the relations behave like the ordinary non-fuzzy relations.

4.2 Partitionings

The key notion for modeling calendar systems as well as many repeating events, for example the seasons, is the notion of *a finite partitioning of the real numbers*. A finite partitioning of $\mathbb{R}$ splits the time axis into an infinite sequence of finite intervals, and these intervals can be numbered by integers. Basic time units like seconds, minutes, hours, days, weeks, months, years etc. can all be represented by finite partitions of $\mathbb{R}$. The partitions can be of equal length, but usually they have different sizes. In fact, the only relevant partitioning with equal sized partitions in WebCal are the seconds. Already the minutes have different sizes: minutes with added leap seconds are longer than 60 seconds.
Besides the basic time units in calendar systems, there are a lot of other temporal notions which can be modeled as partitions: the seasons, the sequence of Easter dates, financial years, semesters in universities, the sequence of sunrises and sunsets, the sequence of the tides, the sequence of school holidays etc. The WebCal interface provides different specification mechanisms for defining temporal notions as partitionings at run time. There are, however, other partitionings which need to be defined with special algorithms. Examples are the sequence of Easter dates, sunsets and sunrises etc. They are built-in.
WebCal treats built-in partitions which model time units of calendar systems in the same way as defined partitions representing for example the seasons on the northern or southern hemisphere. All operations working for example with weeks and months work in the same way for the defined partitions. The algorithm for

computing the n'th week in a month can therefore also compute for example the n'th week in the summer.
In WebCal there are four different methods for defining partitionings. The first method requires a starting point, an average length of the partitions and a correction function. All standard time units, seconds, minutes, hours, weeks, months, years etc. can be defined this way. The correction function for years, for example, has to deal with leap years.
The second method defines a partitioning by giving an anchor date and the length or the partitions. For example, the following command specifies semesters as sequences of 7 months followed by five months, starting in October.

timeUnit semester regular 2000/10 winter 7 summer 5 month

In the third method we provide instead of the lengths of the partitions concrete dates. For example, the following command

timeUnit season calendrical 2000/3/21 spring 6/21 summer 9/23 autumn 12/21 winter +1/3/21 month

specifies the seasons as a partitioning.
Finally, with the fourth method we provide concrete dates. It can for example be used to define holidays.

timeUnit BavarianHolidays finite 2002/10/28 Herbst 2002/11/2 gap 2002/12/23 Weihnachten 2003/1/4 gap 2003/3/3 Winter 2003/3/7 gap 2003/4/14 Ostern 2003/4/26 gap 2003/6/10 Pfingsten 2003/6/21 gap 2003/7/28 Sommer 2003/9/8

In this case the partitioning is preceded by an infinite partition and followed by an infinite partition. In all other cases the sequence of partitions is infinite itself. Partitions can be labeled, and these labels can be used in various algorithms. 'summer', 'winter' etc. are such labels. 'gap' is a built-in label.

4.3 Calendar Systems

A calendar system in WebCal is essentially a collection of time units (seconds, minutes, hours, day, weeks, months, years etc.). Each of these time units is modeled as a partitioning of the reference time line. WebCal provides a class `Partitioning`, which allows one to define partitionings by providing the *average length* of a partition and a correction function for the cases that the length of a particular time unit differs from the average length. For example, the Gregorian year is defined by an average length of 31536000 seconds (365 days) and a correction function which adds 86400 seconds (one day) for each leap year. This way, adding a new calendar system is quite easy, and the code to be implemented needs to deal with the calendar specific concepts (leap years, length of months, etc.) only.
Since calendar systems have been changed during the history of countries and societies, WebCal allows one to define historical sequences of calendar systems. For example, 'Julian 1582/10/4 Gregorian' in the 'calendarSequence' command specifies that all dates before October, 4th, 1582 (in the Julian calendar) should be interpreted in the Julian calendar, and all later dates should be interpreted in the Gregorian calendar.

4.4 Manipulation of Intervals

With the 'shift' command one can shift intervals by a given number of time units. For example, one can shift forwards by 3 month, 1.5 days, 1 semester, and backwards by 2.6 weeks etc.
Very similar to the shift command is the 'extend' command. Whereas shift always shifts the whole interval, the extend command only shifts the upper or lower part of the (fuzzy) interval. This way intervals can be extended for example by 1 year, by 3.5 minutes, by 1.2 seasons, by 2 holiday periods etc.
The most powerful command is the 'within' command. It models expressions like 'the second day within the week' (Tuesday) or 'the third month within the year' (March) or the 'the last day within the year' (new years eve) or 'the third decade within the century' or 'the second but last winter within the decade' or 'the last week within the summer holiday' etc. Within comes with a number of control parameters which give it an enormous flexibility to compute different things with the same basic algorithm. For example, the version 'first month within day' can be used to cast a smaller interval, for example a day, into a larger interval, in this case a month.

4.5 Definition Language

Many temporal notions are very application or user specific. For example, 'the second school hour' is a temporal notion, which may only be relevant for me and a few other people. Such a notion can't be built into a system. Nevertheless, I may want to ask a TV-program database 'give me all documentations about the Iraq war shown during the second school hour'. To account for this, WebCal provides a specification language for defining new temporal notions. With this language, we may for example define the n'th school hour during the time interval x:

$$\begin{aligned} &school\_hour(x, n) = \\ &shift(extend(begin\_minute(hour\_within\_day(x, (n <= 5)?8 : 13)), \\ &\qquad 45, minute), (n-1) * 45, minute) \end{aligned}$$

This defines a block of 45 minute school hours between 8 o'clock and 11:45, and a second block of 45 minute school hours after 13:00.

4.6 Limits of the Current Version of WebCal

There are various things on the agenda to be built-in. Some of them are only a matter of time, but they do not need new concepts: more calendar systems, special time sequences like sunrises, sunset, moonrises, tides, solar and lunar eclipses etc. Although these require special algorithms, they all can be modeled as partitionings, and this fits nicely into the concepts of WebCal. The Easter dates (western and orthodox) are already modeled this way

More serious changes are required to overcome the next restriction: so far, WebCal can only deal with concrete time intervals, for example 'one week after Easter'. It cannot deal with notions like 'one week between Christmas and Easter'. This denotes a 'floating interval' which is constrained by concrete dates. In order to deal with such intervals, we need a constraint handling mechanism. Unfortunately things are not so easy that we can take one off the shelf. Consider the following constraints: 'at most one month' and 'at least 29 days'. This means that it cannot be a February, except in a leap year. Obviously these constraints depend very much on the structure of the calendar system. It is currently not clear how to integrate this into WebCal.

5 Conclusion

In this paper we tried to give an overview on the various aspects of temporal and locational reasoning in the Web context. We focused on the problem to evaluate queries to XML documents and taking into account the proper semantics of temporal and locational notions. We have shown that there are quite a number of different ideas, methods and systems involved to evaluate even such simple queries like 'give me all films ending before midnight' to an XML cinema program database. Some of the problems have already been solved, in particular for temporal reasoning. Before we can show a first prototype of a complete system, however, many other problems still have to be investigated. For most of them it is not really difficult to find some solution, but all the solutions must fit together in a quite complex system, and the solutions must really be practical, and not only solve simplified abstracted cases.

References

1. *W3C http://www.w3.org/TR/xmlschema-0/: XML Schema Part 0: Primer.* 2001.
2. *W3C http://www.w3.org/TR/xmlschema-1/: XML Schema Part 1: Structures.* 2001.
3. *W3C http://www.w3.org/TR/xmlschema-2/: XML Schema Part 2: Datatypes.* 2001.
4. B. Bennett, A.G. Cohn, F. Wolter, and M. Zakharyaschev. Multi-dimensional modal logic as a framework for spatio-temporal reasoning. *Applied Intelligence*, 17(3):239–251, 2002.
5. T. Berners-Lee, M. Fischetti, and M. Dertouzos. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web.* Harper, San Francisco, September 1999. ISBN: 0062515861.
6. François Bry, Sacha Berger, and Sebastian Schaffert. Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and Semistructured Data. In *Proceedings of the* 29^{th} *Intl. Conference on Very Large Databases (VLDB03), Germany*, 2003.
7. François Bry and Sebastian Schaffert. A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In *International Workshop*

on Rule Markup Languages for Business Rules on the Semantic Web (invited article), http://www.soi.city.ac.uk/~msch/conf/ruleml *, Italy, June*, 2002.
8. François Bry and Sebastian Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *Akmal B. Chaudhri, et al. (Eds.), Lecture Notes in Computer Science (LNCS) on Web, Web-Services, and Database Systems*, pages 295–310, 2002.
9. François Bry and Stephanie Spranger. Temporal Constructs for a Web Language. In *Proceedings of the* 4^{th} *Workshop on Interval Temporal Logics and Duration Calculi during ESSLLI'03, Austria, August, to appear*, 2003.
10. A G Cohn, B Bennett, J M Gooday, and N Gotts. RCC: a calculus for region based qualitative spatial reasoning. *GeoInformatica*, 1:275–316, 1997.
11. Anne Doucet, Stéphane Gançarski, Geneviève Jomier, and Sophie Monties. Versions of Integrity Constraints in Multiversion Databases. In *Abdelkader Hameurlain, A. Min Tjoa (Eds.): Database and Expert Systems Applications,* 8^{th} *International Conference, DEXA, Toulouse, Proceedings. Lecture Notes in Computer Science, 1308, Springer*, pages 252–261, 1996.
12. Stéphane Gançarski. Database Versions to represent Bitemporal Databases. In *Trevor J. M. Bench-Capon, Giovanni Soda, A. Min Tjoa (Eds.): Database and Expert Systems Applications,* 10^{th} *International Conference, DEXA, Florence, Proceedings. Lecture Notes in Computer Science, 1677, Springer*, pages 832–841, 1999.
13. Hans Jürgen Ohlbach. About real time, calendar systems and temporal notions. In H. Barringer and D. Gabbay, editors, *Advances in Temporal Logic*, pages 319–338. Kluwer Academic Publishers, 2000.
14. Hans Jürgen Ohlbach. Calendar logic. In I. Hodkinson D.M. Gabbay and M. Reynolds, editors, *Temporal Logic: Mathematical Foundations and Computational Aspec ts*, pages 489–586. Oxford University Press, 2000.
15. Hans Jürgen Ohlbach and Dov Gabbay. Calendar logic. *Journal of Applied Non-Classical Logics*, 8(4), 1998.
16. J. G. Stell. Part and complement: Fundamental concepts in spatial relations. In *Proceedings 7th International Syposium on AI and Mathematics, Florida, January 2002.* to appear.