A Formal Model for the TIMESPACE Software

Philippos Pouyioutas
Department of Computer Science, Intercollege, Cyprus

This paper proposes a formal model to underpin the development of the TIMESPACE software. This software can be used to produce very useful statistics regarding the use of rooms in houses. The statistics can be utilized for research related to the social aspects of life associated with the houses and the time people spend in the various rooms for various activities. This paper is therefore concerned with the storage and retrieval of architectural spatio-temporal information. The proposed model caters for the representation of time in data structures, which are used to store data regarding architectural spaces. The representation of time is achieved through the extension of these structures with lifespans. The introduction of lifespans into data structures representing spaces and users of spaces will allow us to manipulate the stored data and produce very useful information related to the time and use of spaces. A few examples of possible questions/queries that we will be able to answer are: which space is used the most/least, which users use a space during a particular time, which spaces are used by different users at the same time, etc. In order to produce such information from the data structures, we need to define some operators on lifespans. Such operators can then be used either within a query language allowing ad-hoc or predefined queries, or within a programming language for predefined queries. In this paper we formally define these operators and provide their pseudo-code specifications. We also provide some pseudo-code specifications of possible queries to illustrate the use and usefulness of these operators.

Keywords: spatio-temporal database applications, formal model.

1. Introduction

Architects and sociologists have been studying for some time now the sociological aspects related to architectural design [1, 2, 3, 4, 5, 6]. This paper presents the TIMESPACE software, which can be used as a research tool for architects and sociologists to help them carry out their research. The need for this tool emanates from the need to process data related with spaces and time. Currently, researchers use ready made statistical packages such as SPSS and not specialized tailored-made packages. Experience in using SPSS for processing such data has shown many limitations in producing the required information. We therefore decided to design and develop TIMESPACE, a specialized software tool to process complex spatio-temporal data.

The analysis of the information provided by TIMESPACE will help architects design and build more usable houses. It will also help sociologists to understand better how houses affect the social life of people and vice versa, that is, how the social life of people reflect in the design of houses. A comparison analysis of the analysis of results from different countries/cultures can reveal very interesting information to sociologists and architects.

Herein, we propose a model of time to cater for the representation of spatio-temporal information and to underpin the development of the proposed tool. Work in the area of time-related information in computerized systems is mainly carried out by researchers in temporal databases. The model of time proposed herein is based on this work.

In Section 2 of this paper, we provide a formal mathematical model of time, in order to cater for the storage and retrieval of architectural spatio-temporal information. The model consists of both time extended data structures and operators dealing with time extensions. In Section 3, we give the running example of the paper, which is used to illustrate the storage of data in the time extended data structures and explain the meaning of the represented data. We
also provide examples of sample queries showing the information produced by the software and provide pseudo-code specifications of these queries showing the use and usefulness of the proposed time operators. In the Appendix, we provide pseudo-code specifications for the operators of Section 2. Finally, we conclude by reviewing our work and briefly discussing our future work in this area.

2. Formal Definition of a Time Model

In this section we introduce a formal model for representing time and thus provide the constructs for extending data structures so that data is related with time periods. Our work is based on the work of various researchers in the area of temporal databases and, more particularly, on the work in [7]. Herein, we do not provide a review of relevant work. Such review can be found in [7, 8, 9, 10, 11, 12].

The proposed model differs from any other previously defined object-oriented database model, mainly due to the introduction of object-inheritance, which extends the inheritance concept from the class level to the class instance (object) level. Object inheritance allows sub-objects to inherit values from super-objects. The time-extended model [7] also provided the basis for extending the object-oriented query algebra dealing with object inheritance [7, 13] to deal also with time [7].

An aspect of incorporating time in a system is to consider whether time is continuous or discrete [8]. Continuous time is viewed as being isomorphic to real numbers whereas discrete time is viewed as being isomorphic to natural numbers. With continuous time, each real number corresponds to a time point; with discrete time, each natural number corresponds to a non-decomposable time point having an arbitrary duration. Although time is generally perceived to be infinite and continuous, most computerized systems dealing with time incorporate discrete time for several practical reasons. Finally, it is important to note that whether time is discrete or continuous, it should be ordered linearly. Hence, for two non-equal times, \( t_1 \) and \( t_2 \), either \( t_1 \) is before \( t_2 \) or \( t_2 \) is before \( t_1 \).

We choose to provide a formal model of time based on the concepts of interval and intervalset. If a value is known to exist (has life) from a time point \( t_1 \) to a time point \( t_2 \) for all the time points between \( t_1 \) and \( t_2 \), then an interval \([t_1, t_2]\) can be used to denote the time period for a value’s lifespan. Finally, if the lifespan of a value is not a set of consecutive time points but rather a set of scattered time points, then an ordered set (list) of intervals can be used to represent such a lifespan (intervalset). The following definitions are taken form [7] and are used to build the formal model of time, which includes both the data structure lifespan (intervalset) and the operators on lifespans.

**Definition 1.** A time domain, \( D \), is a non-empty, finite, totally ordered set of consecutive elements of the same type. The elements of a time domain are referred to as time points.

**Definition 2.** Let \( D = \{d_1, d_2, ..., d_n\} \), \( d_1 < d_2 < ... < d_n \), be a time domain. A time interval, \( T = [d_p, d_q] \), \( d_p \leq d_q \), over the time domain \( D \) is defined as a finite set of consecutive time points, \( [d_p, d_q] = \{d_i | d_i \in D, \ d_p \leq d_i \leq d_q\} \). The time points \( d_p \) and \( d_q \) are called the boundary points of the interval. Point \( d_p \) is called start point and point \( d_q \) is called end point and are given respectively by \( T.start \) and \( T.end \). There is a special time point called now that denotes the current point in time.

**Definition 3.** Let \( t \) be a time point and \( T \) an interval; \( t \) is included in \( T \), written \( t \in T \), iff \( \exists d_i \in T: t = d_i \).

**Definition 4.** Let \( T_1 \) and \( T_2 \) be two intervals over the same time domain; \( T_1 \) meets \( T_2 \), written \( T_1 \cap T_2 \), iff \( T_1.end = T_2.start \).

**Definition 5.** Let \( T_1 \) and \( T_2 \) be two intervals over the same time domain; \( T_1 \) equals \( T_2 \), written \( T_1 = T_2 \), iff \( T_1.start = T_2.start \) \& \( T_1.end = T_2.end \).

**Definition 6.** Let \( T_1 \) and \( T_2 \) be two intervals over the same time domain; \( T_1 \) precedes \( T_2 \), written \( T_1 \preceq T_2 \), iff \( T_1.end \leq T_2.start \).

**Definition 7.** Let \( T_1 \) and \( T_2 \) be two intervals over the same time domain; \( T_1 \) is-contained in \( T_2 \), written \( T_1 \subseteq T_2 \), iff \( T_2.start \leq T_1.start \) \& \( T_1.end \leq T_2.end \).
Definition 8. Let \( T_1 \) and \( T_2 \) be two intervals over the same time domain; \( T_1 \) overlaps with \( T_2 \), written \( T_1 \cap T_2 \), iff \( \exists t : t \in T_1 \) & \( t \in T_2 \).

Definition 9. An interval set, \( S \), is a sequence of distinct intervals of the same time domain,
\[
S = \{ T_i | 1 \leq i \leq n, n \geq 0 \},
\]
such that \( \forall i : 1 \leq i \leq n-1, T_i \cap T_{i+1} \cap \text{start} < (T_{i+1}.\text{start}) - 1^* \), where \( 1^* \) stands for the unit of time. For \( n = 0 \), the interval set \( S \) is said to be the empty interval set (\( \{ \} \)). Given a non-empty interval set \( S \), \( S.\text{first} \) returns the first interval of \( S \), and \( S.\text{last} \) returns the last interval of \( S \).

Definition 10. Let \( T_1 \) be an interval and \( S_1 \) an interval set; \( T_1 \) is interval included in \( S_1 \), written \( T_1 \subseteq S_1 \), iff \( \exists T_i \in S_1 : T_i = T_1 \).

Definition 11. Let \( t \) be a time point and \( S_1 \) an interval set; \( t \) is found in \( S_1 \), written \( t \in S_1 \), iff \( \exists T_i : T_i \subseteq S_1 \) & \( t \in T_i \).

Definition 12. Let \( T_1 \) be an interval and \( S_1 \) an interval set; \( T_1 \) is time-included in \( S_1 \), written \( T_1 \subseteq S_1 \) iff \( \forall t : t \in T_1 \), \( t \in S_1 \).

Definition 13. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) meets \( S_2 \), written \( S_1 \cup S_2 \), iff \( S_1.\text{last} = S_2.\text{first} \).

Definition 14. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) co-starts with \( S_2 \), written \( S_1 \cap S_2 \), iff \( S_1.\text{first} = S_2.\text{first} \).

Definition 15. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) co-ends with \( S_2 \), written \( S_1 \cup S_2 \), iff \( S_1.\text{last} = S_2.\text{last} \).

Definition 16. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) is interval-contained in \( S_2 \), written \( S_1 \subseteq T S_2 \), iff \( \forall T : T \in S_1 \), \( T \in S_2 \).

Definition 17. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) is time-contained in \( S_2 \), written \( S_1 \subseteq T S_2 \), iff \( \forall T : T \in S_1 \), \( T \subseteq S_2 \).

Definition 18. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) is time-contained in \( S_2 \), written \( S_1 \subseteq T S_2 \), iff \( \forall T : T \in S_1 \), \( T \subseteq S_2 \).

Definition 19. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) is time-equal to \( S_2 \), written \( S_1 = T S_2 \), iff \( \forall S_1 \cap S_2 \subseteq S_2 \).

Definition 20. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) is time-equal to \( S_2 \), written \( S_1 = T S_2 \), iff \( \forall S_1 \cap S_2 \subseteq S_2 \).

Definition 21. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) time-overlaps with \( S_2 \), written \( S_1 \cap T S_2 \), iff \( \exists T_1 : T_1 \subseteq S_1 \) & \( \exists T_2 : T_2 : T_1 \cap T_2 \).

Definition 22. Let \( S_1 \) and \( S_2 \) be two non-empty interval sets; \( S_1 \) interval-overlaps with \( S_2 \), written \( S_1 \cap T T S_2 \), iff \( \exists T_1 : T_1 \in S_1 \), \( \exists T_2 : T_2 : T_1 \cap T_2 \).

Definition 23. Let \( S_1 \) be an interval set. The unfold operator, denoted by \( \text{unfold}(S_1) \), returns the set of all time points that are found in \( S_1 \), and is defined by \( \text{unfold}(S_1) = \{ t | t \in S_1 \} \).

Definition 24. Let \( X \) be a set of time points. The fold operator, denoted by \( \text{fold}(X) \), returns an interval set, and is defined by (\( \exists ! \) stands for there exists a unique) \( \text{fold}(X) = \{ T_i | 1 \leq i \leq n, n \geq 0 \} \& \forall t : t \in X \) \( \exists t' : t' \in T_i \) & \( \forall j : 1 \leq j \leq n \), \( \forall t' : t' \in T_j \).

Definition 25. Let \( S_1 \) and \( S_2 \) be two interval sets. Let also \( * \) denote one of the three set operators, union, intersection or difference (\( \cup, \cap, - \)). Finally let \( * \_T \) denote the corresponding interval set operator (\( \cup_T, \cap_T, -_T \)). Then \( * \_T \) is defined by \( S_1 * \_T S_2 = \text{fold}(\text{unfold}(S_1) * \text{unfold}(S_2)) \).

3. The Running Example and Pseudo Code Specifications of Sample Queries

In this section, we present the running example of the paper and provide pseudo code specifications of sample queries. The supporting data structures of the application are the Interval and Lifespan (see Appendix). The database of the application consists of the Person and House data structures (see Figure 1). The Person data structure is used to store personal information such as gender, age, marital status, educational background, nationality, job and job work hours. The personal information stored will allow (if needed) more detailed analysis in terms of the various characteristics of the users of the spaces. The House data structure stores the information related to the various spaces in the house (Labels) and the use of these spaces (Use). The use of the spaces is described by the Activities taking place in these Labels, the people (By) participating in these activities (Husband, Wife, Children) and the time (When) these activities take place. The example given in the next page (see Figure 2) provides the following information for House1. There are four people living in the house (Husband (p1), Wife (p2) and two Children (p3, p4)). The Labels Kitchen and Lounge are used for various activities. In
the Kitchen, *Eating* takes place during 08.00 to 09.00 and 13.00 to 14.00 by the wife and the two children and during 08.00 to 09.00 by the husband. *Cooking* also takes place during 11.00 to 13.00 by the wife and during 18.00 to 19.00 by the husband. In the Lounge, *Eating* takes place during 20.00 to 21.00 by all the family. Finally, *Watching TV* takes place during 21.00 – 24.00 by the husband and wife and during 21.00 to 22.00 by the children.

Herein, we also illustrate the use and usefulness of the proposed model (data structures and operators) by providing sample queries describing the user requirements. The queries are specified
in pseudo-code. In order to understand the way the query results will be presented, we give for each query a sample result. In order to simplify our example, we present a simplified version of the Activity/Label table, including some of the Labels and some of the Activities of our interest. In the following queries, \textit{Houses} will be the set of all houses under investigation.

**Query 1.** List the Labels and the number of houses that these Labels are occupied during the day at 14.00 – \textit{use of the found operator (definition 11)} – see Figure 3 for results.

Let ActivityLabel1 be a 1D Array [Label] of Integer, Let flag be a boolean

for H in Houses
for L in H.HouseLabels
  flag = false
  for U in L.Use
    for W in U.Used
      if 14.00 is found in W.\textit{When} then flag = true end if
    end for
  end for
if flag \neq true then
  ActivityLabel1[L.Label] = ActivityLabel1[L.Label] + 1
end if
end for
end for

time-included

**Query 2.** List the number of houses that an Activity happens in a Label during the day from 13.00 to 14.00 – \textit{use of the time-included operator (definition 12)} – see Figure 4 for results.

for H in Houses
for L in H.HouseLabels
for U in L.Use
  flag = false
  for W in U.Used
    if [13.00,14.00] is time-included in W.\textit{When} then flag = true end if
  end for
if flag = true then
end if
end for
end for

Similar to Query 1 and Query 2 we can write the pseudo-code of various other queries. Below, we just give some more queries (without the pseudo-code and expected type of result) and state the operator, which should be used in answering each query.

**Query 3.** For each house, list the Labels, which start to be occupied when the \textit{Eating} activity finishes – \textit{use of the meets operator (definition 13)}.

**Query 4.** For each house, list the activities, which start to happen when the \textit{Watching TV} activity starts, together with the labels where these activities happen – \textit{use of the co-starts operator (definition 14)}.

**Query 5.** For each house, list the activities, which finish at the same time – \textit{use of the co-ends operator (definition 15)}.

<table>
<thead>
<tr>
<th>Label</th>
<th>Kitchen</th>
<th>Yard</th>
<th>Lounge</th>
<th>Bedrooms</th>
<th>Formal Lounge</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>85</td>
<td>10</td>
<td>30</td>
<td>10</td>
<td>5</td>
</tr>
</tbody>
</table>

*Fig. 3. Results of Query 1.*

<table>
<thead>
<tr>
<th>Activity/Label</th>
<th>Kitchen</th>
<th>Yard</th>
<th>Lounge</th>
<th>Bedrooms</th>
<th>Formal Lounge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eating</td>
<td>80</td>
<td>5</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Relaxing</td>
<td></td>
<td>10</td>
<td>10</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Cooking</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Playing</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleeping</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Reading</td>
<td>5</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meetings</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watching TV</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

*Fig. 4. Results of Query 2.*
**Query 6.** For each house, list the labels, the activities and the people involved during $[[12.00-14.00], [15.00-18.00]]$ – use of the interval-contained operator (definition 16).

**Query 7.** For each house, list the labels, the activities and the people involved during only at $[[12.00-14.00], [15.00-18.00]]$ and at no other time – use of the interval-equals operator (definition 17).

**Query 8.** For each house, list the activities and labels where these activities happen together with all other activities/labels that happen at the same time – use of the time-overlaps operator (definition 21).

We also give below some queries, their pseudo-code and the expected result to illustrate the use of the unfold and union operators, as well as some more complicated calculations.

**Query 9.** Given a House H, give for each Label the total time occupied by the family – see Figure 5 for results.

Let H be the House, Let ActivityLabel3 be a 1D Array [Label] of Integer
for L in H.HouseLabels
  for U in L.Use
    for W in U.Used
      Hours = Hours $\cup T$ unfold(W.When)
    end for
  end for
ActivityLabel3[L.Label] = count(Hours)
end for

**Query 10.** For all houses give the time spent by the family on each Activity in a Label – see Figure 6 for results.

Let ActivityLabel4 be a 2D Array [Activity, Label] of Integer
for H in Houses
  for L in H.HouseLabels
    for U in L.Use
      for W in U.Used
        Hours = Hours $\cup T$ unfold(W.When)
      end for
    end for
  end for
end for

**Query 11.** What activities happen, in which Labels, in how many houses, by whom and in how much time? – see Figure 7 for results.

Let ActivityLabel5 be a 2D Array [Activity, Label] of [NumberOfHouses, People, TotalTime]
for H in Houses
  for L in H.HouseLabels
    for U in L.Use
      ActivityLabel5[U.Activity, L.Label].NumberOfHouses = ActivityLabel5[U.Activity, L.Label].NumberOfHouses + 1
      for W in U.Used
        Hours = Hours $\cup T$ unfold(W.When)
        ActivityLabel5[U.Activity, L.Label].People = ActivityLabel5[U.Activity, L.Label].People $\cup T$.By
      end for
      ActivityLabel5[U.Activity, L.Label].TotalTime = ActivityLabel5[U.Activity, L.Label].TotalTime + count(Hours)
    end for
  end for
end for

<table>
<thead>
<tr>
<th></th>
<th>Kitchen</th>
<th>Yard</th>
<th>Lounge</th>
<th>Bedrooms</th>
<th>Formal Lounge</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>2</td>
<td>10</td>
<td>21</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

*Fig. 5. Results of Query 9.*

<table>
<thead>
<tr>
<th>Activity/Label</th>
<th>Kitchen</th>
<th>Yard</th>
<th>Lounge</th>
<th>Bedrooms</th>
<th>Formal Lounge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eating</td>
<td>250</td>
<td>60</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Relaxing</td>
<td></td>
<td>30</td>
<td>120</td>
<td>100</td>
<td>40</td>
</tr>
<tr>
<td>Cooking</td>
<td>200</td>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Playing</td>
<td>60</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sleeping</td>
<td></td>
<td></td>
<td></td>
<td>700</td>
<td></td>
</tr>
<tr>
<td>Reading</td>
<td>30</td>
<td>100</td>
<td>100</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meetings</td>
<td>30</td>
<td>200</td>
<td></td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>Watching TV</td>
<td></td>
<td>500</td>
<td>200</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Fig. 6. Results of Query 10.*
4. Conclusion

In this paper we have presented a formal model for the representation of time-related information. The model comprises the data structures needed for the representation of time (lifespans) and the operators on these data structures. We have illustrated the use and usefulness of the model through an example. By showing sample queries and providing the pseudo-code of these queries we have shown how we can produce rich information. Such information would be impossible to be produced using either a statistical package or a non-specialized software package.

Our initial work has been presented in [15]. Our future work involves the implementation of the proposed formal model in a database system such as Oracle or Microsoft Access. This model will provide the basis for writing the implementation of the data structure lifespan and the operators on lifespans. Once the data structures and the operators are implemented and the database is built, the pseudo-code of the given queries will be used to write the programs, which will provide the required information. The implementation of the data structures and the operators will result in the software package TIMESPACE. Further requirements analysis will also be carried out so that the users of the expected software provide us with the expected full functionality of the system.

Once TIMESPACE is implemented, we will process the questionnaires of a survey conducted last year researching in this topic area. This survey was analyzed through the statistical package SPSS. The limitations of SPSS to produce accurate and rich information proved the need for the TIMESPACE software. Through the analysis of the survey with this specialized software, we will be able to produce accurate and rich information, which will help us understand the various issues under investigation and hence arrive to concrete conclusions. In our future work, we will present the developed software and the results of the processing of the survey questionnaires through this software.
Appendix – Pseudo-Code Specifications for the Time Operators

Herein, we give the definitions of the data structures of the formal model of time by giving the data structures Interval and Lifespan and the pseudo-code specifications of the functions, which will implement the various operators, defined on these data structures. The Lifespan data structure is the main structure which will be used for the representation of time when building data structures in applications using TIMESPACE.

Definition 2.

type Interval : tuple (start: integer, end: integer);

Definition 9.

type Lifespan: list(interval)

Function for Operator of Definition 10.

Function interval-included (I: Interval, L: Lifespan):

Boolean flag = false
index = 1
while not(flag) and index ≤ length(L)
  if (I.start = L[index].start and I.end = L[index].end)
    then flag = true
  end if
  index = index + 1
end while
return flag

Function for Operator of Definition 11.

Function found (t: integer, L: Lifespan): Boolean
flag = false
index = 1
while not(flag) and index ≤ length(L)
  if (t ≥ L[index].start and t ≤ [index].end)
    then flag = true
  end if
  index = index + 1
end while
return flag

Function for Operator of Definition 12.

Function time-included (I: Interval, L: Lifespan):

Boolean flag = false
index = 1
while not(flag) and index ≤ length(L)
  if (I.start ≥ L[index].start and I.end ≤ L[index].end)
    then flag = true
  end if
  index = index + 1
end while
return flag

Function for Operator of Definition 13.

Function meets (L1: Lifespan, L2: Lifespan): Boolean
flag = false
if length(L1) <> 0 and length(L2) <> 0 then
  if (L1[length(L1)].end = L2[1].start) then
    flag = true
  end if
end if
return flag

Function for Operator of Definition 14.

Function co-starts (L1: Lifespan, L2: Lifespan): Boolean
flag = false
if length(L1) <> 0 and length(L2) <> 0 then
  if (L1[1].start = L2[1].start) then flag = true end if
end if
return flag

Function for Operator of Definition 15.

Function co-ends (L1: Lifespan, L2: Lifespan): Boolean
flag = false
if length(L1) <> 0 and length(L2) <> 0 then
  if (L1[length(L1)].end = L2[length(L2)].end) then flag = true
  end if
end if
return flag

Function for Operator of Definition 16.

Function interval-contained (L1: Lifespan, L2: Lifespan):

Boolean flag = true
index = 1
if length(L1) = 0 or length(L2) = 0 then
  flag = false
else
  while flag and index ≤ length(L1)
    if not(interval-included(L1[index], L2)) then
      flag = false
    end if
    index = index + 1
  end while
end if
return flag

Function for Operator of Definition 17.

Function interval-equal (L1: Lifespan, L2: Lifespan):

Boolean flag = false
if length(L1) <> 0 and length(L2) <> 0 then
  if interval-contained(L1, L2) and interval-contained(L2, L1)
    then flag = true end if
  else
    return flag
end if

Function for Operator of Definition 18.

Function time-contained (L1: Lifespan, L2: Lifespan):

Boolean flag = true
index = 1
if length(L1) = 0 or length(L2) = 0 then
  flag = false
else...
while flag and index ≤ length(L1) if not(time-contained(L1[index], L2)) then  
   flag = false end if  
index = index + 1  
end while  
return flag

**Function for Operator of Definition 19.**

Function time-equal (L1: Lifespan, L2: Lifespan):  
Boolean  
flag = false  
If length(L1) <> 0 and length(L2) <> 0 then  
   flag = time-contained(L1, L2) and  
time-contained(L2, L1)  
end if  
return flag

**Function for Operator of Definition 20.**

Function time-overlap (L1: Lifespan, L2: Lifespan):  
Boolean  
flag1 = false  
flag2 = true  
if length(L1) = 0 or length(L2) = 0 then  
   flag = false  
else  
while not(flag1) and index ≤ length(L1)  
t = L1[index].start  
while flag2 and t ≤ L1[index].end  
if found(t, L2) then  
   flag1 = true; flag2 = false  
end if  
t = t + 1  
end while  
index = index + 1  
end while  
end if  
return flag1

**Function for Operator of Definition 21.**

Function interval-overlap (L1: Lifespan, L2: Lifespan):  
Boolean  
L3: Lifespan  
flag = true  
index = 1  
if length(L1) = 0 or length(L2) = 0 then  
   flag = false  
else  
while flag and index ≤ length(L1)  
   L3 = list(L1[index]))  
if not(time-overlap(L3, L2)) then  
   flag = false  
end if  
end while  
end if  
return flag

**Function for Operator of Definition 22.**

Function intersect(L1:Lifespan, L2:Lifespan): Lifespan  
return fold(sort-list(set-to-list(list-to-set(unfold(L1) + unfold(L2)))))

**Function for Operator of Definition 23.**

Function unfold (L: Lifespan): List  
I: Interval  
t: integer  
X: list(integer)  
X = list()  
for i in L  
   t = t + 1  
else while t ≤ L.end  
   X = X + list(t)  
end while  
return X

**Function for Operator of Definition 24.**

Function fold(S: list(integer)): Lifespan  
I: Interval  
L: Lifespan  
if (length(S) = 0) then return list()  
else L = ([S[0], S[0]]  
   num_of_int = 1  
i = 1  
while i ≤ length(S) - 1  
   if S[i] = S[i - 1] + 1 then  
      L[num_of_int - 1].end = S[i]  
   else  
      num_of_int = num_of_int + 1  
      i = [S[i], S[i]]  
      L = L + list (1)  
   end if  
i = i + 1  
end while  
end if  
return L

**Supporting Functions**

Function list-to-set(L: list(integer)): set(integer)  
S: set(integer)  
for i in L  
   if (not(i in S)) then S = S + i end if  
end for  
return S

Function set-to-list(S: set(integer)): list(integer)  
L: list(integer) L;  
for s in S L = L + list(s) end for  
return L

**Functions for Operators of Definition 25.**

Function union(L1:Lifespan, L2:Lifespan): Lifespan  
return fold(sort-list(set-to-list(list-to-set(unfold(L1) + unfold(L2)))))

Function intersect(L1:Lifespan, L2:Lifespan): Lifespan  
return fold(sort-list(set-to-list(list-to-set(unfold(L1) * unfold(L2)))))

Function intersect(L1:Lifespan, L2:Lifespan): Lifespan  
return fold(sort-list(set-to-list(list-to-set(unfold(L1) - unfold(L2)))))

where the operators +, *, − are the standard polymorphic (set, list) operators for union, intersection and difference and sortlist is a standard sortlist function.
References


Received: October, 2005
Accepted: April, 2006

Contact address:
Professor Philippos Pouyioutas
Department of Computer Science
Intercollege
46 Makedonitissas Avenue
Nicosia 1700, Cyprus
e-mail: pouyioutas.p@intercollege.ac.cy
www.intercollege.ac.cy

Dr. Philippos Pouyioutas, a Professor in Computer Science, holds a BSc degree in Computer Science from Queen Mary College (University of London), an MSc. degree in Databases and Information Systems from Birkbeck College (University of London) and a PhD. degree in Computer Science (Object-Oriented Databases) from the same University. He also holds a CertEd Diploma in Teaching and Learning in Higher Education from the University of North London. He is currently an Associate Dean and the Director of Academic Affairs at Intercollege, Cyprus. He has also served as the Director of the Computer Science and Multimedia programs and as a chairperson of the Intercollege Research Committee. In the past, Dr. Pouyioutas was a lecturer at Birkbeck College (1989–1996) and at the University of North London (1992–1996) teaching in undergraduate and postgraduate programs. He was also the Director of the BSc program in Business Information Systems at the University of North London (1993–1996). He has served as a chair, editor of proceedings and member of scientific committees of International Conferences and as supervisor (to completion) of a PhD student in the area of Database Systems. His research interests include Relational and Object-Oriented Databases, Graphical and Visual Interfaces and Information Technology in Education and Cultural Heritage. His research work has been supported by various research grants and published in International Conference Proceedings and Journals (1 book written, 1 book edited, 6 journal papers and 34 conference proceedings papers).