Tripod: A Comprehensive System for the Management of Spatial and Aspatial Historical Objects

Tony Griffiths, Alvaro A.A. Fernandes, Norman W. Paton
Department of Computer Science
University of Manchester
Manchester M13 9PL, UK
{griffitt|alvaro|norm@cs.man.ac.uk}

Keith T. Mason
School of Earth Sciences and Geography
University of Keele
Staffordshire ST5 5BG, UK
k.t.mason@esci.keele.ac.uk

Bo Huang, Mike Worboys, Chris Johnson
Department of Computer Science
University of Keele
Staffordshire ST5 5BG, UK
{b.huang|michael|chrisj@cs.keele.ac.uk}

John Stell
School of Computing
University of Leeds
Leeds LS2 9JT, UK
jgs@comp.leeds.ac.uk

ABSTRACT

Spatio-temporal databases have been the focus of considerable research attention in recent years. To date, much of this work has focused on the relational data model, with object data models receiving far less consideration. Where descriptions of such object models do exist, there is currently a lack of systems that build upon these models to produce database architectures that address the broad spectrum of issues related to the delivery of a fully functional spatio-temporal DBMS. This paper presents an overview of such a system by describing a spatio-historical object DBMS that utilises a specialised mechanism, called a history, for maintaining knowledge about entities that change over time. Key features of the resulting proposal include: (i) consistent representations of primitive spatial and temporal types; (ii) a component-based design in which spatial, temporal and historical extensions are formalised incrementally, for subsequent use together or separately; (iii) compatibility with mainstream query processing frameworks for object databases; and (iv) the integration of the spatio-temporal proposal with the ODMG standard.

1. INTRODUCTION

This paper provides an overview of the Tripod project, which is developing a spatio-temporal object database system that extends the ODMG standard for object databases [3]. Figure 1 illustrates the relationships between the different components in Tripod. At the core is the ODMG object model, which is extended with primitive spatial and temporal types. The spatial types are those of the ROSE algebra [11], and the temporal types are one dimensional versions of the two dimensional ROSE algebra types `Points` and `Lines`. Past states of all ODMG types, including the spatial and temporal types, can be recorded using histories. Figure 1 from *Histories* inwards represents a spatio-historical object model.

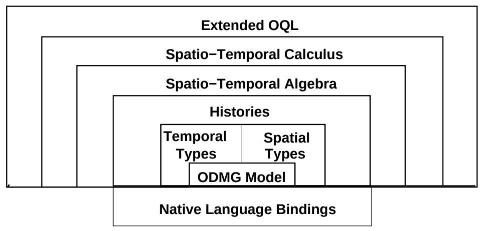

Figure 1: Tripod components.

Outside *Histories* in Figure 1, the upper half of the figure represents the declarative query interface, while the lower half of the figure represents the imperative programming interface. The query interface is based on OQL, and is given a semantics and an optimisation infrastructure through a mapping onto an extension of the monoid comprehension calculus of [5], as described in [6]. The programming interface follows the ODMG approach by mapping object model constructs into programming language objects within an existing object-oriented programming language.

This paper provides an overview of the object model, query language and programming language facilities of Tripod in Sections 4, 5 and 6, respectively. In each of these sections, the facilities of Tripod are illustrated using examples from a land use case study, which is described in Section 3. This case study is representative of many in which the application tracks discrete changes to both spatial and aspatial data over time, as supported by Tripod.

2. ARCHITECTURE

This section describes in more detail the various components of the Tripod architecture shown in Figure 1, and in particular (as shown in Figure 2) how these components interact with each other in the specification of spatio-historical database applications. There are three main components in the Tripod architecture: a Persistent Store that is responsible for loading and saving persistent objects to and from the database and also for maintaining metadata about a particular database schema; a Query Processor that is responsible for optimizing and executing database queries; and a Programming Language Binding that is responsible for providing programming language access to the database.

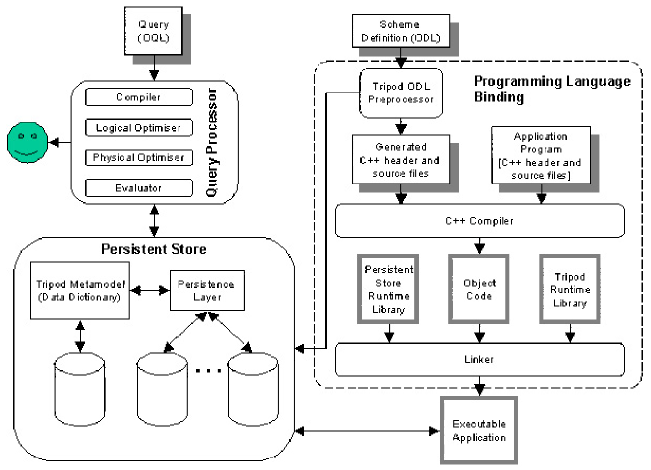

Figure 2: Detailed Tripod Architecture.

The definition of a Tripod database consists of two parts: a schema (defined using a declarative object definition language (ODL)) specifying the structure of the database types and their behaviour, and an implementation of each of these behaviours specified using a programming language binding – in our case this is C++. Since the ODMG model does not define an object manipulation language (OML), developers must use a programming language binding to create, update and delete objects. The Tripod ODL preprocessor lies at the core of the process of producing a database specification. It is responsible for analysing an ODL schema specification to produce: a set of C++ header files whose structure corresponds to that of the types expressed in the ODL schema definition; an instance of the Tripod metamodel (which is a superset of the ODMG metamodel) containing high-level information about the structure of the database schema that is used by (amongst others) the query processor; and methods to load and save persistent objects to and from the Tripod persistent store.

Once the application program and type information is compiled into object code, it is linked with libraries that implement the Tripod runtime system, and the persistent store. The library implements the core ODMG object model types as well as the Tripod spatial, temporal, and historical types. The persistent store runtime library contains the functionality needed to create and manage database connections, transactions and queries. The output of this process is an executable application that interacts with the underlying spatio-historical OODBMS.

3. CASE STUDY: UK NATIONAL LAND USE DATABASE

In the UK, a project is underway to create a National Land Use Database (NLUD) (`http://www.nlud.org.uk/`). The NLUD aims to provide a complete, consistent and detailed geographical record of land use in England. In this, land use parcels (the basic spatial units of the system) will likely be formed from Ordnance Survey (the UK mapping agency) Digital National Framework (DNF) 'atomic polygons', which are themselves defined by topographic features and uniquely referenced by a system of Topographic Identifiers. The NLUD will be delivered by specific projects that respond to particular user requirements. The Tripod investigation, although not officially linked with the NLUD project, builds on one such initiative, the NLUD Previously Developed Land (PDL) project, as a basis for testing the applicability of the Tripod model and languages on a land use change scenario. The PDL project has been set up by the NLUD partnership to monitor the supply and re-use of vacant, derelict, or previously developed sites, that might be available for further development.

Under the PDL proposals, sites are categorised as belonging to one of six possible classes, including land and buildings which are now vacant, derelict land and buildings, and land and buildings going through the various stages of planning permission or construction. A key objective of the NLUD PDL project is to maintain the life histories of PDL sites and to support the update and maintenance of site records as the user records change. Potential changes to individual land use parcels might record changes to one or more of the site attributes, for instance an alteration of PLD classification, or might come from one of seven possible categories of geometric change, including: creation, destruction, alteration, reincarnation, fusion, fission and reallocation.

4. THE TRIPOD OBJECT MODEL

The ODMG Object Model provides a set of object and literal types – including collection types, (e.g., `Set`, `Bag` and `List`) and atomic types (e.g., `long`, `float` and `string`) – with which a designer can specify their own object types, and construct a particular database schema. Each user-defined type has a structure (a collection of attributes and binary relationships with other user-defined types) and a behaviour (a collection of methods whose implementation is specified using the language binding).

Tripod supports the storage, management and querying of spatial and aspatial entities that change over time through the notion of a *history*. A history models the changes that an entity, or its attributes, or the relationships that it participates in, undergoes as the result of assignments made to it. In the Tripod object model, a request for a history to be maintained can be made for any construct to which a value can be assigned, i.e., a history is a history of changes in value and it records episodes of change by identifying these with a timestamp. For example, the `lu_parcel` type shown in Figure 3 declares historical attributes (`owner` and `land_type`), a spatio-historical attribute (`gext`), and two historical relationships (`has_tpfea` and `in_admin`). In addition, the `lu_parcel` type is itself declared to be historical, indicating that the database should maintain a history (called *lifespan*) recording when instances of this type are active or inactive (i.e., logically deleted) in the database. In contrast, the `admin_region` class is not declared as historical, and therefore its instances will not have their lifespan maintained.

The remainder of this section provides an overview of the Tripod object model [7] by presenting its constructs as instances of abstract data types (ADTs), and commences by overviewing the structure of Tripod spatial values, showing how these provide a foundation for Tripod timestamps.

```
class admin_region
  ( extent admin_regions key name)
{  attribute string name;
   attribute Instant founded;
   historical(timeIntervals, MONTH)
      attribute regions gext;
   historical(timeIntervals, MONTH)
      relationship set<lu_parcel>
      has_parcel inverse lu_parcel::in_admin; };

class council extends admin_region
( extent councils ) { ...  };

class county extends admin_region
( extent counties ) { ...  };

historical(timeIntervals,MONTH) class lu_parcel
  ( extent lu_parcels key site_reference )
{  attribute string site_reference;
   historical(timeIntervals, YEAR)
      attribute list<string> owner;
   historical(timeIntervals, MONTH)
      attribute string land_type;
   historical(timeIntervals, MONTH)
      attribute regions gext;
   historical(timeIntervals, MONTH)
      relationship set<topo_feature>
      has_tpfea inverse topo_feature::lup;
   historical(timeIntervals, MONTH)
      relationship admin_region
      in_admin inverse admin_region::has_parcel; };

historical(timeIntervals,MONTH) class topo_feature
  ( extent topo_features key toid )
{  attribute string toid;
   historical(timeIntervals,MONTH)
      attribute string feature_type;
   historical(timeIntervals,YEAR)
      attribute regions gext;
   historical(timeIntervals,MONTH)
      relationship lu_parcel lup
      inverse lu_parcel::has_tpfea; };
```

Figure 3: Land Use Schema Definition

4.1 Spatial Literals

Tripod's spatial data types (SDTs) are based on the ROSE (RObust Spatial Extensions) approach described in [11]. Underlying the ROSE approach is the notion of a *realm*. A realm is essentially a finite set of points and non-intersecting line segments defined over a discrete grid that forms the ROSE algebra's underlying geometric domain. ROSE spatial values are represented in terms of points and line segments in a realm. A realm guarantees that all spatial operations over realm values are error bound and only take, and return, intersection-free spatial values.

The ROSE approach defines an algebra over three SDTs, namely `Points`, `Lines` and `Regions`, and an extensive collection of spatial predicates and operations (including set operations) over these types [11]. Every spatial value in the ROSE algebra is set-based, thus facilitating set-at-a-time processing. Roughly speaking, each element of a `Points` value is a pair of coordinates in the underlying geometry, each element of a `Lines` value is a set of connected line segments, and each element in a `Regions` value is a polygon containing a (potentially empty) set of holes.

Some examples of spatial objects taken from the NLUD are shown in Figure 4. The polygonal objects **38**, **42**, **44** and **45** are `Regions` values (note that **45** is a set of four polygons that contain holes), representing land use parcels. Other objects of interest (marked with an ×) are represented by `Points` values denoting their centroid.

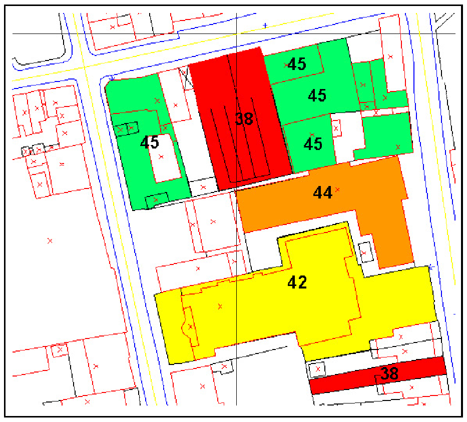

Figure 4: Example of `lu_parcel` objects in a realm

4.2 Timestamp Literals

Tripod extends the set of ODMG primitive types with two temporal types, called `Instants` and `TimeIntervals`. The underlying domain of interpretation is a structure that we refer to as a *temporal realm* because it is defined to be a one-dimensional specialization of the two-dimensional (spatial) realms. In general terms, a temporal realm can be thought of as a finite set of integers (whereas a spatial realm is a finite integer grid). Then, an `Instants` value is a collection of time-points and a `TimeIntervals` value is a collection of pairs of time-points where the first element is the start, and the second the end, of a contiguous time-interval. A *timestamp* is either an `Instants` value or a `TimeIntervals` value. Figure 5 illustrates timestamps in graphical form, where timestamp ***A*** is a `TimeIntervals` value, and timestamps ***B*** and ***C*** are `Instants` values. Notice that ***B*** happens to be a singleton.

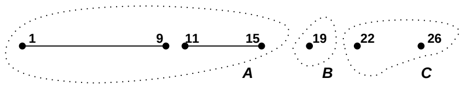

Figure 5: Example Tripod Timestamps

In the ROSE algebra, there is no predefined notion of one `Points` value being spatially ordered with respect to another `Points` value; any such notion of ordering must be defined within application programs that use the algebra. The Tripod temporal algebra, therefore, extends the ROSE algebra with ordering predicates based on the underlying order of the temporal realm's integer domain. These predicates take into consideration the collection-based nature of the timestamp types. Therefore, in addition to what might be considered the 'standard' temporal predicates (e.g., those defined by Allen's algebra [1]), our temporal predicates are extended to take into account quantification over the individual elements of the timestamp. For example, whether every element of a timestamp `A` must be contained by an element from timestamp B, or just some. In addition, the temporal realm utilises a calendar that maps from the underlying integer domain to one more suited to human cognition.

Although Tripod timestamps can be used by application designers to complement the related primitive types in the ODMG standard (e.g., `Interval` or `Time`), their main purpose is to allow histories to be constructed and operated upon, as described below.

4.3 Histories

A *history* is a quadruple $H = \langle V, \theta, \gamma, \Sigma \rangle$, where V denotes the domain of values whose changes H records, θ is either `Instants` or `TimeIntervals`, γ is the granularity of θ, and Σ is a set of pairs, called *states*, of the form $\langle \tau, \sigma \rangle$, where τ is a Tripod timestamp and σ is a snapshot. In the rest of the paper, let $\mathbb{T}$ denote the set of all timestamps; $\mathbb{V}$, the set of all snapshots; $\mathbb{S}$, the set of all states; and $\mathbb{H}$, the set of all histories.

In a history, a set Σ of states is constrained to be an injective function from the set $\mathbb{T}_H$ of all timestamps occurring in H to the set $\mathbb{V}_H$ of all snapshots occurring in H, i.e., for any history H, $states_H : \tau \in \mathbb{T}_H \rightarrow \sigma \in \mathbb{V}_H$. Therefore a particular timestamp is associated with at most one snapshot (i.e., a history does not record different values as valid at the same time), and a particular snapshot is associated with at most one timestamp (i.e., if a value is assigned more than once, in the corresponding history the new occurrence causes the timestamp of the previous occurrence to adjust appropriately).

The remainder of this section provides an overview of the operations available to operate on histories construed as instances of an ADT, which leads to their behaviour being categorised into constructor, query, merge and update operations. These operations require precise definition so that the semantics of operations and structures in the higher level layers of the Tripod OM can be appropriately specified. For example, the language bindings utilise operations to create and manipulate historical data, and the query language and its associated calculus utilise operations that filter and traverse histories to retrieve appropriate results. For reasons of space, the semantics of these operations are provided by exemplars; fuller details are available elsewhere [7].

Representative retrieval operations on histories are shown in Figure 6. Note that the first expression in Figure 6 is in fact a template for a set of signatures parameterised on any element of the set of predicates on timestamps. For example, given that $\underline{\texttt{before}}$ is a member of that set, letting $\omega = \underline{\texttt{before}}$ in the template yields the following signature $\texttt{ContainsTimestamp\_}\underline{\texttt{before}} : \mathbb{H} \times \mathbb{T} \rightarrow \texttt{boolean}$. Other such parameterised templates include $\texttt{FilterByTimestamp\_}\omega$.

$$\begin{array}{rl} \texttt{ContainsTimestamp\_}\omega : & \mathbb{H} \times \mathbb{T} \;\rightarrow\; \texttt{boolean} \\ \texttt{FilterBySnapshot} : & \mathbb{H} \times \mathbb{V} \;\rightarrow\; \mathbb{H} \end{array}$$

Figure 6: Example Retrieval Operations

For example, if the state set of two histories representing the history of change to instances H_1 and H_2 of the `lu_parcel` type's `gext` attribute (both with $V =$ `Regions`, $\theta =$ `TimeIntervals` and identical γ) are $\Sigma_1 = \{\langle [1-6], \mathtt{r}_1 \rangle, \langle [9-11], \mathtt{r}_2 \rangle\}$ and $\Sigma_2 = \{\langle [5-10], \mathtt{r}_3 \rangle, \langle [13-20], \mathtt{r}_1 \rangle\}$ (where $\mathtt{r}_1$, $\mathtt{r}_2$ and $\mathtt{r}_3$ are `Regions` values) then $\texttt{ContainsTimestamp\_}\underline{\texttt{before}}(H_1, [9-10]) = \texttt{true}$ and $\texttt{ContainsTimestamp\_}\underline{\texttt{after}}(H_2, [21-22]) = \texttt{false}$.

Representative update operations on histories are shown in Figure 7.

$$\begin{array}{rl} \Cup : & \mathbb{H} \times \mathbb{H} \;\rightarrow\; \mathbb{H} \\ \texttt{DeleteTimestamp} : & \mathbb{H} \times \mathbb{T} \;\rightarrow\; \mathbb{H} \\ \texttt{InsertState} : & \mathbb{H} \times \mathbb{S} \;\rightarrow\; \mathbb{H} \end{array}$$

Figure 7: Example Update Operations

The union of two histories (obtained through the $\Cup$ operator) is equivalent to taking the union of their state sets but choosing the state in the second argument whenever there is a state in the first argument with the same timestamp but different snapshot. This is to satisfy the constraint that a history does not record different values as valid at the same time. For example, using infix notation, if the state sets of two histories H_1 and H_2 are as exemplified above, then the state set of $H = H_1 \Cup H_2$ is $\Sigma = \{\langle [1-5, 13-20], \mathtt{r}_1 \rangle, \langle [5-10], \mathtt{r}_3 \rangle, \langle [10-11], \mathtt{r}_2 \rangle\}$. The definitions of $\Cap$ and $\backslash\backslash$, for intersecting and subtracting histories, are analogous.

`DeleteTimestamp` takes a history $H = \langle V, \theta, \gamma, \Sigma \rangle$ and a timestamp τ of type θ and yields a new history $H' = \langle V, \theta, \gamma, \Sigma' \rangle$ in which all states in Σ whose timestamp τ' is such that $\underline{\texttt{common\_points}}(\tau, \tau')$ is true, have been recomputed so that τ does not occur in Σ'. For example, if $\tau = [3-4]$ and $\Sigma = \{\langle [1-6], \mathtt{r}_4 \rangle\}$, then $\Sigma' = \{\langle [1-3, 4-6], \mathtt{r}_4 \rangle\}$.

`InsertState` takes a history $H = \langle V, \theta, \gamma, \Sigma \rangle$ and a state $\langle \tau', \sigma' \rangle$, where τ' is of type θ and $\sigma' \in V$, and yields a new history $H' = \langle V, \theta, \gamma, \Sigma' \rangle$. If σ' is equal to some σ occurring in Σ then the timestamp τ associated with it is recomputed into a timestamp τ_+ that includes τ', and $\Sigma' = \Sigma \backslash \{\langle \tau, \sigma \rangle\} \cup \{\langle \tau_+, \sigma \rangle\}$. If, on the other hand, σ' does not occur in Σ, then Σ is recomputed into a state set Σ_+ that is everywhere equal to Σ except that every state in Σ whose timestamp has common points with τ' has been recomputed so as to make that no longer the case in Σ_+, and $\Sigma' = \Sigma_+ \cup \{\langle \tau', \sigma' \rangle\}$. For example, if $\langle \tau', \sigma' \rangle = \langle [5-8], \mathtt{r}_1 \rangle$ and $\Sigma = \{\langle [1-6], \mathtt{r}_2 \rangle\}$, then $\Sigma' = \{\langle [1-5], \mathtt{r}_2 \rangle, \langle [5-8], \mathtt{r}_1 \rangle\}$ and if $\langle \tau', \sigma' \rangle = \langle [5-8], \mathtt{r}_4 \rangle$ and $\Sigma = \{\langle [1-6], \mathtt{r}_4 \rangle\}$, then $\Sigma' = \{\langle [1-8], \mathtt{r}_4 \rangle\}$.

5. THE QUERY LANGUAGE

Tripod-OQL employs and extends the facilities of OQL to retrieve spatial, temporal and historical information. The states in histories are extracted through iteration in the OQL *from-clause*. Constraints in the *where-clause* can then be applied to the snapshot value, timestamp or index number of a state, and the granularity of the timestamp, through operations that have been defined in the Tripod OM. Finally, the result is obtained through the projection operation in the *select-clause*.

Given the above, Tripod-OQL provides syntactical extensions to OQL to manipulate historical information. In Table 1, **e** is an expression denoting a history, and **es** is an expression denoting a state within a history.

The nth-state operation views **e** as a chronologically ordered history when selecting the appropriate state. Similarly **e.index(es)** returns the index number of **es** within the chronologically ordered history **e**.

Tripod-OQL can express spatial, temporal and historical queries. Examples are given in Figure 8 using the land use application.

Expressions	Explanation
es.value	snapshot value of state **es**
es.validTime	timestamp of state **es**
es.granularity	granularity of state **es**
e.Nth(n)	*nth*-state of **e**
e.index(es)	index number of state **es** in **e**

Table 1: Accessing histories in Tripod-OQL

(Q1) *Which counties were founded before Avon?* (Temporal query)

```
select  county1.name
from    county1 in counties, county2 in counties
where   county2.name = 'Avon' and
        county1.founded.before(county2.founded)
```

(Q2) *What are the parcels that at some point in time bordered land parcel 2601?* (Spatio-Historical query)

```
select  lup2.site_reference
from    lup1 in lu_parcels, lup2 in lu_parcels,
        lupgext1 in lup1.gext, lupgext2 in lup2.gext
where   lup1.site_reference = '2601' and
        lupgext1.value.border_in_common
          (lupgext2.value) and
        lupgext1.validTime.common_points
          (lupgext2.validTime) and
        lup1 != lup2
```

(Q3) *Display the previous version of parcel 2604's geometry?* (Historical version selection)

```
select  gext1.value
from    lup in lu_parcels,
        gext1 in lup.gext, gext2 in lup.gext
where   lup.site_reference = '2604' and
        gext2.validTime.common_points(|now|) and
        lup.gext.index(gext1) = lup.gext.index(gext2) - 1
```

(Q4) *What were the neighbouring parcels of parcel 2586 when topological feature 1897 was associated with this parcel?* (Spatio-Historical join)

```
select  distinct lup2.site_reference
from    lup1 in lu_parcels, lup1gext in lup1.gext,
        lup2 in lu_parcels, lup2gext in lup2.gext,
        tpfeas in lup1.has_tpfea,
        topo_feature in tpfeas.value
where   lup1 != lup2 and
        lup1.site_reference = '2586' and
        topo_feature.toid = '1897' and
        lup2gext.value.border_in_common
          (lup1gext.value) and
        lup2gext.validTime.common_points
          (tpfeas.validTime)
```

Figure 8: Spatial, temporal and historical queries

6. THE LANGUAGE BINDINGS

The Tripod language bindings provide developers with a programming language (C++) interface that allows them to create, update and delete objects (i.e., an OML). The language bindings are also used to specify a user-defined type's operations. When the state of a database needs to be queried, developers can either issue declarative OQL queries or write native language application programs. The language bindings extend those of the ODMG standard by mapping the Tripod OM types into C++ classes that can persist in the database. The language bindings provide implementations of the Tripod spatial, temporal and historical types.

For each type in a Tripod schema, the Tripod ODL preprocessor generates a corresponding C++ class. For example, Figure 9 is the class definition automatically generated for the `lu_parcel` type of Figure 3 (note that all operations have been omitted). Line 1 illustrates that each persistence capable class inherits from the built-in `d_PersistentObject` type. Each historical property in an ODL type is mapped to a history template type which requires the type of the property, its temporal type, and its granularity. For example, the `owner` attribute is mapped to a history type (line 7) whose snapshots are each a list of strings. The `has_tpfea` relationship on the other hand is mapped to a history type (line 10) whose snapshots are each a set of topological features.

```
1  class lu_parcel:public d_PersistentObject
2  {
3  private:
4    history<d_TimeIntervals,Status,MONTH> lifespan;
5  public:
6    d_String site_reference;
7    history<d_TimeIntervals,d_List<d_String>,YEAR> owner;
8    history<d_TimeIntervals,d_String,MONTH> land_type;
9    history<d_TimeIntervals,d_Regions,MONTH> gext;
10   history<d_TimeIntervals,
11      d_Rel_Set<topo_feature,lup>,MONTH> has_tpfea;
12 };
```

Figure 9: `lu_parcel` class definition

```
1  d_TimeIntervals t1("[1/1990 - until_changed]");
2  d_TimeIntervals t2("[1/1990 - 4/1995]");
3  d_TimeIntervals t3("[4/1995 - 5/1999]");
4  d_State<d_Regions,d_TimeIntervals> s1(regions1,t1);
5  d_State<d_Regions,d_TimeIntervals> s2(regions2,t3);
6
7  d_Ref<lu_parcel> lup8601 =
8    new(ludb,"lu_parcel", t1) lu_parcel;
9
10 lup8601->gext.InsertState(s1);
11 lup8601->gext.InsertState(s2);
12 lup8601->gext.DeleteTimestamp(t2);
13
14 d_BiDirectionalIterator iter =
15   lup8601->gext.createStateIterator();
16 d_TimeInstants now("|now|");
17 d_State<d_Regions,d_TimeIntervals> tmp;
18 while(!iter.at_end()) {
19   tmp = (d_State)iter.getElement();
20   if(tmp.getValidTime().before(now)
21     cout << "value" << tmp.getSnapshot() << endl;
22   iter.nextPosition();
23 }
```

Figure 10: Language binding example

Figure 10 shows how instances of user-defined types are created using an extended version of the C++ `new` operator. Lines 7 and 8 create a new `lu_parcel` object that is stored in the `ludb` database, with an appropriate lifespan. Lines 10 and 11 populate this object's `gext` spatio-historical attribute with two states whose snapshots are previously created regions values (not shown). Line 12 deletes a portion of this history. Lines 14 to 23 illustrate how a history can be iterated over using one of the iterators provided for this purpose. The `d_BiDirectionalIterator` allows chronological as well as reverse-chronological of a history. Lines 14 to 23 are equivalent to the query: "*What is the boundary history of parcel 8601 prior to today?*".

7. RELATED WORK

Despite progress in certain aspects of spatio-temporal modelling and implementation (e.g., indexing, join algorithms, etc.), there are few examples of spatio-temporal database systems, and most lack support for changes to aspatial data. Langran [8] developed a spatial vector model in which line segments are used as primitives to produce polygons. Each of these polygons is then timestamped with its own attribute history using discrete semantics. The TRIAD model [10] takes a different approach by using events as the basic notion in their raster-based model. More recently, the MADS model [9] reflects many of the concerns addressed in the Tripod object model, including the orthogonal treatment of spatial and aspatial data. However, MADS does not address manipulation and querying issues.

Recently, much work on spatio-temporal databases has centered on objects whose properties (spatial and aspatial) are continuously changing (the so-called *moving-object* models). Such models (e.g., [12]) allow the state of each spatial and aspatial property to be expressed as a continuous function of time. Queries about the position of spatial data can then be inferred by the interpolation of spatial values between known bounds. However, such models do not provide comprehensive support for temporally changing aspatial data and object model constructs such as relationships, which are supported in a uniform way in Tripod. In contrast, the Tripod data model and calculus do not model continuous change, as we explicitly target the large body of applications in which objects change in discrete steps, as exemplified by the NLUD case study.

One of the aims of the Tripod design has been to provide effective support for spatial (but not historical) and historical (but not spatial) applications, as well as those that must manage spatio-historical data. As a "pure" spatial database, Tripod provides spatial types as primitive types in the object model, and thus the extension to the ODMG model in Tripod can be seen as analogous to the extension of object-relational products with spatial Data Blades or Cartridges. As a "pure" temporal database, Tripod provides a concise collection of valid-time modelling facilities. Probably the most closely related work is that described in [2], which also presents an extension to the ODMG model. The principal contributions of Tripod relative to [2] have been to address programming and querying of the temporal ODMG extension. Another closely related proposal is that of TOQL [4]. Tripod has taken a similar approach to TOQL in the addition of histories into the ODMG object model, although the Tripod temporal types are richer than those in TOQL, and a wider range of query and manipulation operations are supported for histories.

8. CONCLUSIONS

This paper has provided an overview of the Tripod project, which is developing a spatio-temporal object database system. Key features of Tripod are: (i) Modeling, querying and programming facilities are extensions of those provided by the ODMG standard. The extensions to the standard in Tripod are orthogonal, in that the spatial and historical facilities can be used together or separately. (ii) The spatial and temporal types have shared origins, and both provide comprehensive, consistent, collection-based structures and operations to higher levels in the architecture. (iii) The historical modeling facilities can be applied consistently to spatial and to aspatial aspects of an application; in most applications in which historical spatial data is important, historical aspatial data is important as well. (iv) The proposal is targeted at discrete changes to spatial and aspatial data; although there has been considerable attention directed in recent years at moving object databases, we note that few prototypes have been developed that support histories of spatial and aspatial data. The paper has been illustrated using a dataset that is characterised by such data.

The implementation of the Tripod system is underway; it is hoped that all the functionality described in this paper will be implemented by the Spring of 2002, and that the system will be made publically available during 2002.

Acknowledgments. This research is funded by the UK Engineering and Physical Sciences Research Council (EPSRC), whose support is gratefully acknowledged.

9. REFERENCES

[1] J. Allen. Maintaining knowledge about temporal intervals. *CACM*, 26(11):832–843, 1983.

[2] E. Bertino, E. Ferrari, G. Guerrini, and I. Merlo. Extending the ODMG Object Model with Time. In *Proceedings ECOOP'98*, pages 41–66, 1998.

[3] R. G. G. Cattell, editor. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann, 2000.

[4] L. Fegaras and R. Elmasri. A Temporal Object Query Language. In *Proc. TIME*, pages 51–59. IEEE Press, 1998.

[5] L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *ACM TODS*, 25(4), 2000.

[6] T. Griffiths, A.A.A. Fernandes, N. Djafri, and N.W. Paton. A Query Calculus for Spatio-Temporal Object Databases. In *Proc. TIME*, pages 101–110. IEEE Press, 2001.

[7] T. Griffiths, A. Fernandes, N. Paton, K. Mason, B. Huang, and M. Worboys. Tripod: A Comprehensive Model for Spatial and Aspatial Historical Objects. In *Proc. ER*. Springer-Verlag, 2001.

[8] G. Langran. *Time in Geographical Information Systems*. Taylor and Francis, 1992.

[9] C. Parent, S. Spaccapietra, and E. Zimanyi. Spatio-Temporal Conceptual Models: Data Structures + Space + Time. In *Proc. ACM GIS*, pages 26–33, 1999.

[10] D. Peuquet and L. Qian. An Integrated Database Design for Temporal GIS. In *Proc. 7th SDH*, pages 21– 31. Taylor and Francis, 1997.

[11] R. H. Güting and Markus Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):243–286, 1995.

[12] R.H. Güting et al. A Foundation for Representing and Querying Moving Objects. *ACM Transactions on Database Systems*, 25(1):1–42, 1000.