Tripod: A Comprehensive Model for Spatial and Aspatial Historical Objects

Tony Griffiths[1], Alvaro A.A. Fernandes[1], Norman W. Paton[1],
Keith T. Mason[2], Bo Huang[3], and Michael Worboys[3]

[1] Department of Computer Science, University of Manchester
Manchester M13 9PL, UK – {griffitt|alvaro|norm}@cs.man.ac.uk
[2] School of Earth Sciences and Geography, University of Keele
Staffordshire ST5 5BG, UK – k.t.mason@esci.keele.ac.uk
[3] Department of Computer Science, University of Keele
Staffordshire ST5 5BG, UK – {b.huang|michael}@cs.keele.ac.uk

Abstract Spatio-temporal extensions to data models have been an active area of research for a number of years. To date, much of this work has focused on the relational data model, with object data models receiving far less consideration. This paper presents a spatio-historical object model that uses a specialized mechanism, called a *history*, to maintain 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 formalized incrementally, for subsequent use together or separately; (iii) a formally specified data model. The model can be used directly during the design of spatio-historical applications, but also forms the basis of an implementation activity developing a spatio-historical object database management system.

1 Introduction

Spatio-temporal databases have been an active area of research for a number of years. Central to this effort has been the development of models to facilitate the description of complex applications with spatial and/or temporal features. Few of these models, however, provide facilities to track both spatial and aspatial changes to data over time. The temporal aspects of conceptual modelling have been considered by several researchers (see [24] for a survey), with more recent proposals dealing with both temporal and spatial concerns (e.g. [4,16]). There has also recently been considerable interest in models that characterize objects whose properties (spatial and aspatial) are continuously changing – the so-called *moving object* approaches (e.g. [17,20]). However, there remain considerable challenges concerning the modelling of discretely changing objects (as exemplified by the running example in this paper). Also, any implementation of a data model requires a formally specified model, considering not only the structure of the model and its inherent constraints, but also the pragmatics of how to manipulate and query its entities.

The Tripod project, from which this paper emerges, is seeking to design and prototype a complete spatio-temporal database system. The activity is focusing on the extension of the ODMG standard for object databases [3] with facilities for managing vector spatial data, and for describing past states of both spatial and aspatial data. The key principles underpinning the Tripod project are *orthogonality* and *synergy*. By *orthogonality* is meant that the different extensions

to the ODMG standard should be coherent in isolation, so that, for example, the Tripod system should be effective as a historical database in which no spatial data is stored, or as a spatial database in which no use is made of the ability to record historical data. By *synergy* is meant that the system should allow the combined use of spatial and temporal capabilities in a seamless and complementary manner, so that full spatio-temporal applications benefit from integrated facilities without mismatches in the way different features are supported.

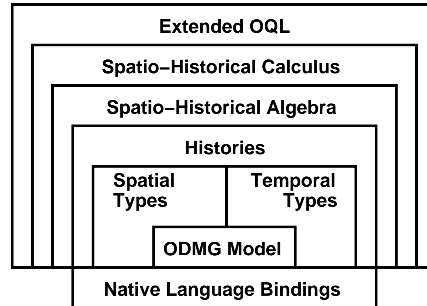


Figure 1. Tripod Layered Architecture

Figure 1 illustrates the relationships between the different components in the Tripod design. At the core is the ODMG object model, which is a subset of the Tripod object model. The ODMG model is extended with two new sets of primitive types, viz., spatial and temporal types. The spatial types used in Tripod are those of the ROSE algebra [18], i.e., `Points`, `Lines` and `Regions`. The temporal types supported in Tripod are one dimensional versions of the ROSE algebra types `Points` and `Lines`, and are known as `Instants` and `TimeIntervals`, respectively. The close relationship between the spatial and the temporal types increases consistency in the representation of the different kinds of data. Past states of all ODMG types, including the spatial and temporal types, can be recorded using histories. A *history* is a collection of *timestamp-value* pairs, where the timestamp is of a temporal type and the value is of any of the types in the extended ODMG model. Figure 1, from Histories inwards, represents a spatio-historical object model. The layers in Figure 1 from Histories outwards denote the interfaces that exist to populate, maintain and query instances of a Tripod database. Since the ODMG model does not define an object manipulation language, developers must use a programming language binding to create, update and delete objects. When the state of a database needs to be queried, developers can either write native language application programs or can issue declarative OQL queries. Tripod's ODL and OQL extend the ODMG ODL and OQL with spatial, historical and spatio-historical constructs. Queries are mapped to a spatio-historical calculus and then to a spatio-historical algebra. These mappings provide several opportunities for optimization using rewrite rules that are extensions to the techniques used by Fegaras and Maier for optimizing object query languages [6]. Tripod's language bindings and its extended OQL use the services provided by the extended object model to access and manipulate historical, spatial and aspatial data.

This paper is structured to reflect Tripod's layered data model, concentrating in particular on the ways in which each layer builds upon the functionality provided by lower levels of the architecture, and how each successive layer conforms to the constraints stemming from these lower layers. The remainder of this paper is structured as follows. Section 2 presents a motivating case study. Section 3 describes the spatial and temporal literal types that Tripod uses to enhance the ODMG type system. Section 4 defines a history as an abstract data type (ADT) underpinned by these temporal types. Section 5 brings together the concepts described in this paper under the unifying framework of a spatio-historical ex-

tension to the ODMG object model. Section 6 discusses related work. Finally, some conclusions are drawn in Section 7. It is assumed that the reader is familiar with the ODMG object model.

2 A Motivating Example

The longitudinal study (LS) links census and vital events data (e.g., births, deaths, cancer registrations) for a one percent sample of the population of England and Wales — about 500,000 individuals at any one census point. The study was started in 1974 with a sample drawn from the residents of England and Wales born on one of four dates each year and enumerated at the 1971 Census. Selection into the LS is by birth date. The study was designed as a continuous, multi-cohort study, with subsequent samples being drawn at each census, using the same selection criteria, and linked into the study. This study is of particular interest because of the spatial and aspatial changes that entities in the study can undergo. In particular, new members are included by virtue of birth on LS dates or by immigration (if born on LS dates) and excluded by death or emigration, and the enumeration districts in which members reside can change boundaries over time. A much simplified schema (adapting the notation in [3] with extensions to represent historical data) showing the main classes of interest in the LS is presented in Figure 2. Classes and relationships that can change over time are marked with a 'H' symbol (i.e., are historical). It should be noted that some of a member's attributes change over time (e.g., their marital status), whereas others (e.g., their place or date of birth) will not.

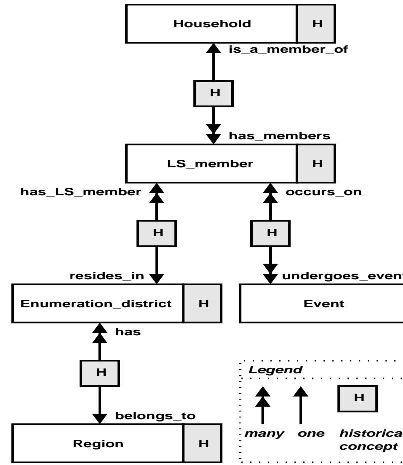


Figure 2. Simplified LS Schema

3 Spatial and Temporal Extensions to the Type System

This section shows how Tripod extends the ODMG type system with types to support the definition of spatial and temporal values. These extensions augment the set of ODMG structured literals, and support the definition of the mechanism by which Tripod maintains a history of changes in the state of modelled entities and relationships.

Operation		Signature
adjacent	:	Regions × Regions → bool
area_disjoint	:	Regions × Regions → bool
area_inside	:	Regions × Regions → bool
border_in_common	:	Regions × Lines → bool
common_border	:	Regions × Lines → Lines
encloses	:	Regions × Regions → bool
equal	:	Regions × Regions → bool
intersects	:	Regions × Lines → bool
length	:	Lines → float
meets	:	Regions × Lines → bool
not_equal	:	Regions × Regions → bool
on_border_of	:	Points × Lines → bool
on_border_of	:	Points × Regions → bool

Figure 3. Example Spatial Operations

Spatial Types Tripod extends the ODMG literal types with three new types for representing spatial data. These spatial data types (SDTs) are based on the ROSE (RObust Spatial Extensions) approach described in [18]. Underlying the ROSE approach is the notion of a *realm*. A realm is 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. These properties lead to an algebra with an efficient implementation [19].

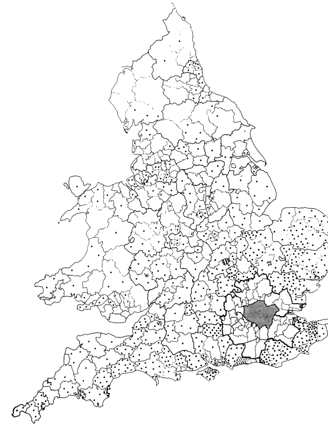


Figure 4. Spatial Values over a Realm

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. Figure 3 shows a small subset of the predicate operations. 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. Such collection-based SDTs are useful in modelling the LS, where areas need to be split. Figure 4 shows examples of spatial objects from the LS. Each LS Region is shown as a `Regions` value, and each LS Member is shown as a `Points` value. The figure illustrates migration destinations of a subset of the LS Members. At a finer scale, Figure 4 would also show LS Enumeration districts as `Regions` values contained within LS Regions.

Temporal Types Tripod extends the set of ODMG primitive types with two temporal types, called `Instants` and `TimeIntervals`. The underlying domain of interpretation is a structure which we refer to as a *temporal realm* because it is defined to be a one-dimensional specialization of two-dimensional (spatial) realms [18]. Roughly, a temporal realm is a finite set of integers (whereas a spatial realm is a finite integer grid). There are several reasons why we adopt this viewpoint and terminology. Realm values are collections, which we find more suitable than scalars for the kind of set-a-time strategies that are prevalent in query processing architectures. Also, realm operations are well-defined and have a rich set of predicates and constructors with nice closure properties. In addition, we find it useful (for users, developers and researchers) to have realms as a unifying notion for the interpretation of operations on spatial *and* temporal values. This unified interpretation propagates upwards in the sense that the predicates and operation on realms are defined once and used (possibly after renaming) over both spatial and temporal values. Finally, this allows the reuse of implemented software components, such as those described in [15].

In a temporal realm, we can view a time-point as an integer. 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.

In Figure 5, ***A*** is a `TimeIntervals` value, and ***B*** and ***C*** are `Instants` values. Notice that ***B*** happens to be a singlet

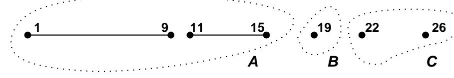


Figure 5. Example Tripod Timestamps

A temporal realm has an additional property to a spatial realm, namely a predefined ordering. In the ROSE algebra, there is no predefined notion of one `Points` value being ordered with respect to another `Points` value; any such notion of ordering must be defined within application programs that use the algebra. Temporal realms, however, must exhibit this property if they are to conform to our intuitions. The Tripod temporal algebra, therefore, extends the ROSE Algebra with ordering predicates based on the underlying order of the temporal realm's integer domain. In addition, the temporal realm uses a calendar that maps from the underlying integer domain to one more suited to human cognition.

Let $\mathbb{T}$ denote the set of all Tripod timestamps. Given $\tau \in \mathbb{T}$, representative predicates and operations defined on $\mathbb{T}$ are shown in Figure 6. It should be noted that predicates such as `contains_`ω are templates for a pair of signatures parameterized on an element of the set $\{\forall, \exists\}$, representing the two possible semantics of the predicate. For example, must every element of a timestamp `A` be contained by an element from another timestamp `B`, or just one? The operation names should give readers an intuitive understanding of their meaning based on operations on sets of integers (and integer pairs) and on classical definitions for temporal predicates (such as Allen's [1], and those defined by Ladkin [13] on sets of intervals). For full details, see the formal semantics in [10,11] (which follow [18] closely).

Operation	Signature	Result
`equal`	: $\tau \times \tau$	$\rightarrow$ `bool`
`before`	: $\tau \times \tau$	$\rightarrow$ `bool`
`starts_before`	: $\tau \times \tau$	$\rightarrow$ `bool`
`disjoint`	: $\tau \times \tau$	$\rightarrow$ `bool`
`common_points`	: $\tau \times \tau$	$\rightarrow$ `bool`
`meets_`ω	: `TimeIntervals` $\times$ `TimeIntervals`	$\rightarrow$ `bool`
`contains_`ω	: `TimeIntervals` $\times$ `TimeInstants`	$\rightarrow$ `bool`
`intersection`	: $\tau \times \tau$	$\rightarrow \tau$
`plus`	: $\tau \times \tau$	$\rightarrow \tau$
`minus`	: $\tau \times \tau$	$\rightarrow \tau$
`vertices`	: `TimeIntervals`	$\rightarrow$ `Instants`

Figure 6. Example Temporal Operations

The discrete domain underlying the temporal realm is bounded by two integer values that correspond to the earliest and latest representable temporal values. These are given the special names `beginning` and `forever`, and can be referenced as Tripod temporal literals. Additionally, many applications require the formation of `TimeIntervals` values whose end instant is as yet undecided (i.e., is *pseudo-open* [22]). For example, a person's period of employment could be from 1/1/1990 until some as yet unspecified future date. Tripod uses the special temporal value `until_changed` to allow such `TimeIntervals` values to be declared. When evaluated during query-processing, any value that references `until_changed` is bound to a distinguished `Instants` value representing the current time `now`, thus having the effect of closing any pseudo-open `TimeIntervals` values. When used as a variable, `now` acts as a temporal placeholder which is filled with the current system time if and when it is evaluated in some expression.

The Complete Tripod Type System The complete set of Tripod types Tripod$\mathcal{T}$ can thus be seen to augment the set of ODMG Object Model types *ODMG*, as follows. Tripod$\mathcal{T}$ = *ODMG* ∪ *TripodLT*, where *TripodLT* = *TripodSLT* ∪ *TripodTLT*, *TripodSLT* = {`Points`, `Lines`, `Regions`, `Point`, `Line`, `Region`}

is the set of Tripod spatial literal types and $TripodTLT = \{$`Instants`, `TimeIntervals`, `Instant`, `TimeInterval`$\}$ is the set of Tripod temporal literal types. Note that for each collection-based spatial and temporal type there is a corresponding singular type. A collection value can be cast into a set of values of its corresponding singular type. This facilitates the derivation of new values from existing ones by application programmers. Although Tripod timestamps can be used by application designers to complement 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 Histories

The Tripod history mechanism provides functionality to support the storage, management and querying of entities that change over time. A history models the changes that an entity (or its attributes, or the relationships 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. Each such value is called a *snapshot*. As a consequence of the possible value assignments that are well defined in the ODMG object model, a history can be kept in the Tripod object model for object identifiers, attribute values, and relationship instances. Thus, a history associates timestamps and snapshots drawn from the domain of exactly one of object identifiers (if the history is of an object), or attribute values (whose type can be any valid Tripod, and hence ODMG, type), or relationship instances (of any ODMG-supported cardinality). In the remainder of this section, a history is defined as an ADT.

4.1 The Structure of 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 collection of pairs, called *states*, of the form $\langle \tau, \sigma \rangle$, where τ is a timestamp and σ is a snapshot. In the rest of the paper, let $\mathbb{T}$ denote, as before, 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 Tripod history, a collection 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, $states_H : \tau \in \mathbb{T}_H \to \sigma \in \mathbb{V}_H$. Therefore, the following invariants hold, for any history $H = \langle V, \theta, \gamma, \Sigma \rangle$:

1. Every timestamp occurring in Σ is of type $\theta \in \{$`Instants`, `TimeIntervals`$\}$ and has granularity γ.
2. For every snapshot σ occurring in Σ, $\sigma \in V$.
3. A particular timestamp is associated with at most one snapshot (note that snapshots can be collections), i.e., a history does not record different values as valid at the same time. For example, an LS member cannot be a member of more than one household at the same time.
4. A particular snapshot is associated with at most one timestamp, i.e., all value-equal snapshots within a history are merged together to form a single state with a collection-based timestamp. This process is called *coalescing*.

The notation used later in the paper is as follows. A `TimeIntervals` value is notated as $[t_1^s - t_1^e, \ldots, t_n^s - t_n^e]$, where each element is a *closed* interval (i.e., the delimiting instants are included). Simple integer values are used rather than calendar-based dates. Although all examples in this paper use the `TimeIntervals` type, they are equally applicable to the `Instants` type. A single state is notated as $\langle[1-5, 6-9], \mathtt{r_1}\rangle$, where $\mathtt{r_1}$ is, e.g., a snapshot value from the domain of `Regions` that holds between the granules 1 to 5 and 6 to 9, inclusively. A history is notated as exemplified by $\{V, \theta, \gamma, \langle[1-3, 7-9], \{\mathtt{r_1}\}\rangle, \langle[4-5], \{\mathtt{r_1}, \mathtt{r_3}, \mathtt{r_4}\}\rangle, \langle[10-14], \{\mathtt{r_1}, \mathtt{r_4}\}\rangle\}$, with $V = \mathtt{bag}\langle\mathtt{Regions}\rangle$, $\theta = \mathtt{TimeIntervals}$, and $\gamma = \mathtt{DAY}$.

4.2 History Operations

The operations defined for the history ADT can be classified into *constructor*, *update*, *query*, *mutator*, and *merge* operations. A sample of each such group is given in Figure 7. It is exemplified below how they can be given a precise semantics.

Operation	Domain	Range
create:		$\rightarrow \mathbb{H}$
create:	$\mathbb{H}$	$\rightarrow \mathbb{H}$
DeleteTimestamp:	$\mathbb{H} \times \mathbb{T}$	$\rightarrow \mathbb{H}$
DeleteSnapshot:	$\mathbb{H} \times \mathbb{V}$	$\rightarrow \mathbb{H}$
DeleteState:	$\mathbb{H} \times \mathbb{S}$	$\rightarrow \mathbb{H}$
DeleteAll:	$\mathbb{H}$	$\rightarrow \mathbb{H}$
InsertState:	$\mathbb{H} \times \mathbb{S}$	$\rightarrow \mathbb{H}$
UpdateTimestamp:	$\mathbb{H} \times \mathbb{T} \times \mathbb{V}$	$\rightarrow \mathbb{H}$
UpdateSnapshot:	$\mathbb{H} \times \mathbb{T} \times \mathbb{V}$	$\rightarrow \mathbb{H}$
IsEmpty:	$\mathbb{H}$	$\rightarrow$ bool
ContainsTimestamp_ω:	$\mathbb{H} \times \mathbb{T}$	$\rightarrow$ bool
ContainsSnapshot:	$\mathbb{H} \times \mathbb{V}$	$\rightarrow$ bool
ContainsState_ω:	$\mathbb{H} \times \mathbb{S}$	$\rightarrow$ bool
FilterByTimestamp_ω:	$\mathbb{H} \times \mathbb{T}$	$\rightarrow \mathbb{H}$
FilterBySnapshot:	$\mathbb{H} \times \mathbb{V}$	$\rightarrow \mathbb{H}$
FilterByState_ω:	$\mathbb{H} \times \mathbb{S}$	$\rightarrow \mathbb{H}$
EarliestState:	$\mathbb{H}$	$\rightarrow \mathbb{S}$
LatestState:	$\mathbb{H}$	$\rightarrow \mathbb{S}$
GetLifespan:	$\mathbb{H}$	$\rightarrow \mathbb{T}$
SubsetOf:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow$ bool
StrictSubsetOf:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow$ bool
Equals:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow$ bool
Brackets:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow$ bool
$\mathtt{dissect}^I$:	$\mathbb{H}$	$\rightarrow 2^{\mathtt{Instant} \times \mathbb{V}}$
$\mathtt{dissect}^T$:	$\mathbb{H}$	$\rightarrow 2^{\mathtt{TimeInterval} \times \mathbb{V}}$
$\Cup$:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow \mathbb{H}$
$\Cap$:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow \mathbb{H}$
$\setminus\!\!\setminus$:	$\mathbb{H} \times \mathbb{H}$	$\rightarrow \mathbb{H}$

Figure 7. Operations on Histories

In what follows, the dot notation is used to denote the individual elements of a particular state. For example, the timestamp of a particular state s is denoted by $s.\mathtt{t}$, and the corresponding snapshot by $s.\mathtt{v}$. Where reference is made to an operation defined on the underlying temporal structured literal types, the operation name is underlined. Let H (possibly primed or subscripted) range over $\mathbb{H}$. Given a history, let S^H denote its state set. Since the state set of a history is a function, its domain is the set of timestamps occurring in it, denoted by S_T^H, and its range the set of snapshots, denoted by by S_V^H. The semantics of an operation Ω is sometimes characterized by writing: $post(\Omega) \Rightarrow \{p_1, \ldots, p_n\}$, where each p_i is a predicate that evaluates to true after Ω is carried out. Alternatively, the semantics of Ω is sometimes characterized by a rewriting: $\Omega \equiv (\sigma)[\Omega_1, \ldots, \Omega_n]$, where each Ω_i is an operation (defined elsewhere) over elements generated in σ. [10,11] give, in detail, the semantics of all operations on histories.

Constructor Operations A history is empty when created. It is also possible to create a history from the state information contained in another history. The signatures for the `create` operations are given in Figure 7. Their semantics are given by $post(H := \mathtt{create}()) \Rightarrow \{\mathtt{isEmpty}(H)\}$ for the empty constructor, and by $H := \mathtt{create}(H') \equiv H := \mathtt{create}() \Cup (\forall s \in S^{H'})[\mathtt{InsertState}(\mathtt{create}(), s)]$, for the non-empty creator, where $\Cup$ and `InsertState` are defined below.

Update Operations The update operations in the history ADT provide the ability to insert, delete and update states of a history. These operations preserve the invariant properties previously described. Some examples are shown in Figure 7, and some of these are now described in detail.

$\texttt{InsertState}$ 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 $\Sigma' = \Sigma \setminus \{\langle \tau, \sigma \rangle\} \cup \{\langle \tau_+, \sigma \rangle\}$. 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 $\Sigma' = \Sigma_+ \cup \{\langle \tau', \sigma' \rangle\}$. The $\texttt{InsertState}$ operation can therefore be seen to be a *destructive*, and hence non-commutative, operation. For example, if $\langle \tau', \sigma' \rangle = \langle [5-8], \texttt{o}_2 \rangle$ and $\Sigma = \{\langle [1-6], \texttt{o}_1 \rangle\}$, then $\Sigma' = \{\langle [1-5], \texttt{o}_1 \rangle, \langle [5-8], \texttt{o}_2 \rangle\}$, and if $\langle \tau', \sigma' \rangle = \langle [5-8], \texttt{o}_1 \rangle$ and $\Sigma = \{\langle [1-6], \texttt{o}_1 \rangle\}$, then $\Sigma' = \{\langle [1-8], \texttt{o}_1 \}$.

$\texttt{DeleteTimestamp}$ takes a history $H = \langle V, \theta, \gamma, \Sigma \rangle$ and a timestamp τ of type θ and yields a new history $H' = \langle V, \theta, \gamma, \Sigma' \rangle$. The operation maps Σ into a state set Σ' 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', otherwise Σ remains unchanged. For example, if $\tau = [2-3]$ and $\Sigma = \{\langle [1-6], \texttt{o}_1 \rangle\}$, then $\Sigma' = \{\langle [1-2, 3-6], \texttt{o}_1 \rangle\}$. More formally, $H' := \texttt{DeleteTimestamp}(H, \tau) \equiv H' := \texttt{create}() \Cup (\forall s \in S^H \mid \underline{\texttt{common_points}}(s.\texttt{t}, \tau))[H \setminus\!\!\setminus_\tau \{s\} \Cup \texttt{InsertState}(\texttt{create}(), \langle \underline{\texttt{minus}}(s.\texttt{t}, \tau), s.\texttt{v} \rangle)]$. It can be seen that if the input timestamp has common points with the timestamp of any existing state, then this state is initially deleted (using the specialized difference operator $\setminus\!\!\setminus_\tau$ on histories, defined below), and a new state is inserted whose timestamp is computed using $\underline{\texttt{minus}}$, and whose snapshot is the same as in the deleted state. This new state is merged with the history using the merge operator $\Cup$ on histories, defined below.

The $\texttt{DeleteSnapshot}$ operation deletes all states within a given history whose snapshot is equal to the input operand.

The remaining update operations assign to existing states either a new timestamp value, τ, or a new snapshot value, σ. For example, the $\texttt{UpdateTimestamp}$ operation updates all states whose timestamps have common points with the input timestamp. There can be several such states and each has its overlapping temporal components updated to the new value.

Query Operations Several operations can be used by the higher layers of the Tripod architecture to query a given history. Such operations can be classified into Boolean, retrieval and filtering operations. Signatures for some of these are shown in Figure 7.

Boolean operations leave the a history unchanged and return a Boolean value to indicate the truth or falsity of the predicate they denote. Filtering operations return a (possibly empty) history.

The $\texttt{ContainsTimestamp_}\omega$ and $\texttt{FilterBySnapshot}$ operations are now defined to exemplify how query operations can be given a precise semantics. The former is defined as the template $\texttt{ContainsTimestamp_}\omega(H, \tau) \equiv (\exists\, \tau' \in S_T^H)[\omega(\tau, \tau')]$. Note that $\texttt{ContainsTimestamp_}\omega$ is a template for a collection of signatures parameterized on any predicate operation on Tripod timestamps. Given that $\underline{\texttt{before}}$ and $\underline{\texttt{after}}$ are members of that set, they can instantiate the template. For example, if the state sets of two histories H_1 and H_2 both with $V = \texttt{Regions}$, $\theta = \texttt{TimeIntervals}$ and identical γ, are $\Sigma_1 = \{\langle [1-6], \texttt{r}_1 \rangle, \langle [9-11], \texttt{r}_3 \rangle\}$ and

$\Sigma_2 = \{\langle[5-10], \mathtt{r}_2\rangle, \langle[13-20], \mathtt{r}_4\rangle\}$ then $\mathtt{ContainsTimestamp_\underline{before}}(H_1, [9-10]) = \mathtt{true}$ and $\mathtt{Contains\ Timestamp_\underline{after}}(H_2, [21-22]) = \mathtt{false}$.

In contrast to $\mathtt{ContainsTimestamp_}\omega$, which queries a history for a true/false reply, the FilterBySnapshot operation exemplifies operations that query histories for a reply that is itself a history. It is defined as $H := \mathtt{FilterBySnapshot}(H', v')$ $\equiv H := \mathtt{create}()\ \uplus\ (\forall s \in S^{H'} \mid s.\mathsf{v} = v')[\mathtt{InsertState}(\mathtt{create}(), \langle s.\mathtt{t}, v'\rangle)]$. For example, if H_1 and H_2 are as above, then to check whether H_1 and H_2 (perhaps recording the history of change to the boundaries of two enumeration districts) contain a particular value $\mathtt{r}_1$, one can use $\mathtt{FilterBySnapshot}(H_1, \mathtt{r}_1) = H_1' = \langle V, \gamma, \theta, \{\langle[1-6], \mathtt{r}_1\rangle\}\rangle$ and $\mathtt{FilterBySnapshot}(H_2, \mathtt{r}_1) = H_2' = \langle V, \gamma, \theta, \{\}\rangle$.

The history ADT also provides several binary Boolean operations to test the relationship between two given histories. For example, it is possible to test if one history is a subset or a strict subset of another history, or whether one history is equal to another history using the SubsetOf, StrictSubsetOf and Equals Boolean operations. Finally, the Brackets operation takes two histories H_1 and H_2 (with identical θ, γ, and V) as arguments, and returns true if the lifespan of H_2 is contained within or equal to the lifespan of H_1. Formally, let $H_1^s := \mathtt{EarliestState}H_1.\mathtt{t}$, $H_1^e := \mathtt{LatestState}H_1.\mathtt{t}$, $H_2^s := \mathtt{EarliestState}H_2.\mathtt{t}$, and $H_2^e := \mathtt{LatestState}H_2.\mathtt{t}$, then $\mathtt{Brackets}(H_1, H_2) \equiv (\underline{\mathtt{starts_before}}(H_1^s, H_2^s) \vee \underline{\mathtt{equals}}(H_1^s, H_2^s)) \wedge (\underline{\mathtt{starts_after}}(H_1^e, H_2^e) \vee \underline{\mathtt{equals}}(H_1^e, H_2^e))$. For example, if the state sets of two histories H_1 and H_2 are, respectively, $\{\langle[1-6], 12\rangle, \langle[9-11], 14\rangle\}$ and $\{\langle[2-11], 12\rangle\}$ then $\mathtt{Brackets}(H_1 H_2) = \mathtt{true}$.

Mutator Operations Tripod provides a collection of functions that transform a given history into another collection that represents a particular *view* over the history. Such views can be used (for example) by the Tripod query calculus [9] to provide different ways of iterating over a particular history. For example, while some users will want to view a history in its previously described canonical form, other users may want to view a history as a collection of states that are timestamped with individual temporal values. The collections returned by such operations may be ordered according to the semantics of the mutator function and the nature of the resultant collection.

```
R := {};
for each ⟨τ,σ⟩ in Σ do
      for each τ' in disassemble(τ)
      R := R ∪{⟨τ',σ⟩};
```

Figure 8. The dissect Operation

For example, the dissect mutator function returns a set of snapshot/timestamp pairs, with the timestamp consisting of *individual* (i.e., Instant or TimeInterval) temporal elements. Given $H = \langle V, \mathtt{Instants}, \gamma, \Sigma\rangle$, $\mathtt{dissect}^I(H)$ returns a set of pairs of the form $\langle\tau', \sigma\rangle$, where $\sigma \in V$ and the type of τ' is Instant, obtainable as shown in Figure 8. The views created by such mutator functions can be used by the Tripod spatio-historical calculus to provide different forms of iteration over a history. For example, expressions of the form $s \leftarrow H$ can be constructed, where $H : \mathbb{H}$ and $s : \mathbb{S}$, such that H is the *domain generator* from which bindings for s are drawn. However, if a calculus expression needs to range over states of the form $\langle s : \mathbb{V}, t : T\rangle$, where T is either Instant or TimeInterval, then dissect can be used, yielding generators like $s \leftarrow \mathtt{dissect}(H)$.

Merge Operations The history ADT provides several binary operations that, based on different criteria, generate a new history from two others. The operations are shown in Figure 7.

Given $H_1 = \langle V, \theta, \gamma, \Sigma_1 \rangle$ and $H_2 = \langle V, \theta, \gamma, \Sigma_2 \rangle$, $H_1 \Cup H_2 = H_3 = \langle V, \theta, \gamma, \Sigma_3 \rangle$, where $\Sigma_3 = (state_{H_1} | (dom(state_{H_1}) \setminus dom(state_{H_2})) \cup state_{H_2})$, where, given a function $f(x)$, its *restriction to the set* E, denoted by $f|E$, is the set of pairs $\langle x, y \rangle$ such that $y = f(x)$ and $x \in E$. In other words, taking the union of two histories 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 invariants that characterize histories. For example, if the state sets of two histories H_1 and H_2 are as exemplified above, then the state set of $H_3 = H_1 \Cup H_2$ is $\Sigma = \{\langle [1-5], \mathtt{r}_1 \rangle, \langle [5-10], \mathtt{r}_2 \rangle, \langle [10-11], \mathtt{r}_3 \rangle, \langle [13-20], \mathtt{r}_4 \rangle\}$.

The $\Cap$ operation is the history ADT's equivalent of the set intersection operation, in that, for two histories H_1 and H_2, its result contains the states that are members of both H_1 and H_2. This operation however has two variants: one, denoted by $\Cap_\tau$, tests for equality between states based on the timestamp values; the other, denoted by $\Cap_\sigma$, based on the snapshot value. Once again, any states from H_2 that are value equivalent take precedence in the result. For example, if the state sets of two histories H_1 and H_2 are as exemplified above, then the state set of $H_3 = H_1 \Cap_\tau H_2$ is $\Sigma = \{\langle [5-6, 9-10], \mathtt{r}_2 \rangle\}$, and the state set of $H_3' = H_1 \Cap_\sigma H_2$ is $\Sigma = \{\}$.

The $\setminus\!\!\setminus$ operation corresponds to set difference, in that, for two histories H_1 and H_2, its result contains all elements of H_1 that are not present in H_2. There are, again, two versions of this operation: one denoted by $\setminus\!\!\setminus_\tau$, based on equal timestamp values; the other, denoted by $\setminus\!\!\setminus_\sigma$, based on equal snapshot values. For example, if the state sets of two histories H_1 and H_2 are as exemplified above, then the state set of $H_3 = H_1 \setminus\!\!\setminus_\tau H_2$ is $\Sigma = \{\langle [1-5], \mathtt{r}_1 \rangle, \langle [10-11], \mathtt{r}_3 \rangle\}$, and the state set of $H_3' = H_1 \setminus\!\!\setminus_\sigma H_2$ is $\Sigma = \{\langle [1-6], \mathtt{r}_1 \rangle, \langle [9-11], \mathtt{r}_3 \rangle\}$.

5 A Spatio-Historical ODMG Object Model

The Tripod Object Model (OM) provides the ability to record the history of change that entities and relationships undergo over time. It extend the ODMG model with the ability to track the changes caused by assignment operations on any assignable construct that is declared to be historical. In other words, any construct denoted by the left hand side of an assignment operation (i.e., an *l-value*) can have a record kept of the different values assigned to it over time. More specifically, the history ADT makes it possible to track the changes caused by the creation and deletion of objects, assignments to object attributes, assignments to object relationships, and assignment to named collections. For example, for the LS there is a need to track changes made to an LS member's marital status, the life events they undergo, and the districts they live in. For each ODMG Object Model concept that is value assignable (i.e., atomic object types, attributes, relationships and collection object types), the Tripod OM provides a *historical* counterpart.

Although histories can be thought of as collections, they are not denoted by a type constructor. Rather the keyword `historical`, as a modifier within the ODL, indicates that a history should be maintained of the modified concept. When such declarations are made, the OM internally creates an instance of the history ADT for each historicized concept. This approach has been taken because if the history mechanism were to appear as a type, then declarations such

as history<history<set<history<String>>>>, would be syntactically valid, despite their semantics being unclear.

In addition to specifying that a database concept should have its history maintained, the designer can also specify certain defaulted properties of the history, viz., its *granularity* and its *temporal type*. Thus, if a history has granularity γ, then the changes maintained by the history cannot vary more than once for each granule of γ (the default is the CHRONON granularity [5]). Any attempt to specify a granularity finer than γ will cause the value to be converted to granularity γ. Also, each change is associated with either an `Instants` or a `TimeIntervals` timestamp. The default is `TimeIntervals`.

```
historical(TimeIntervals, DAY) class LS_member            // spatio-historical class
( extent LS_members )
{      attribute    enum {male, female} sex;              // aspatial and not historical
       attribute    TimeInstant         date_of_birth;    // temporal but not historical
    historical(TimeIntervals, DAY)                        // aspatial historical attribute
       attribute    enum {single,married,divorced,widowed} marital_status;
    historical(TimeIntervals, DAY)                        // spatio-historical relationship
       relationship Enumeration_district resides_in
       inverse      Enumeration_district :: has_LS_members;
};

historical(TimeIntervals, DAY) class Enumeration_district  // spatio-historical class
( extent Enumeration_districts )
{      attribute    string              name;             // aspatial and not historical
    historical(TimeIntervals, DAY)                        // spatio-historical attribute
       attribute    Regions             boundary;
    historical(TimeIntervals, DAY)                        // spatio-historical relationship
       relationship set<LS_member>      has_LS_members
       inverse      LS_member           :: resides_in;
};
```

Figure 9. Historical Type Definitions

For example, Figure 9 shows how the model in Figure 2 can be declared (but some classes in the latter have been omitted). Note that the boundary attribute of an enumeration district implies a spatio-historical property for every LS member through the `resides_in` relationship.

Historical Atomic Object Types In the Tripod OM any user-defined type can be declared as historical. This causes an implicit historical attribute called `lifespan` to be maintained for each instance of that type. The lifespan attribute records whether the object is active or inactive at particular times in the modelled reality. This capability allows objects to be de-/reactivated within a particular database, as opposed to being simply inserted and deleted. For example, through the Tripod language bindings, an `LS_member` object $\mathtt{ls}_1 : \mathtt{T}$ can be created (using an extended `new` operator) to exist during the time period $[10-20]$, and can be subsequently (logically) deleted (using an extended version of the `delete` operator) during the period $[12-15]$, where the `lifespan` attribute for T denotes a `TimeIntervals` history of snapshots drawn from a domain $\mathtt{Status} = \{\mathtt{active,\ inactive}\}$, thus giving ls_1 the history: $o_1.\mathtt{lifespan} = \{\langle[10-12, 15-20],\ \mathtt{active}\rangle, \langle[12-15],\ \mathtt{inactive}\rangle\}$. With specific reference to the LS, this feature allows us to logically delete an `LS_member` when they undergo an *emigration* life event. If an object type is non-historical, then the type can still have historical properties, however its will not have a `lifespan` attribute and deletion will remove current and past states.

Historical Properties Any property (attribute or relationship) of an atomic object type can be declared as historical. For example, in Figure 9, `boundary` is a historical attribute and `resides_in` is a historical relationship. In the ODMG object model, the integrity of a relationship is automatically maintained by the ODBMS. This is also the case with historical relationships, although such maintenance is inherently more complex. For example, when a new state is added to the `resides_in` relationship, described above, the inverse history defined by the `LS_member` type must be maintained so that the constraints specified by the Tripod spatio-historical object model are satisfied (as defined in Section 5). Such maintenance is the responsibility of the Tripod kernel.

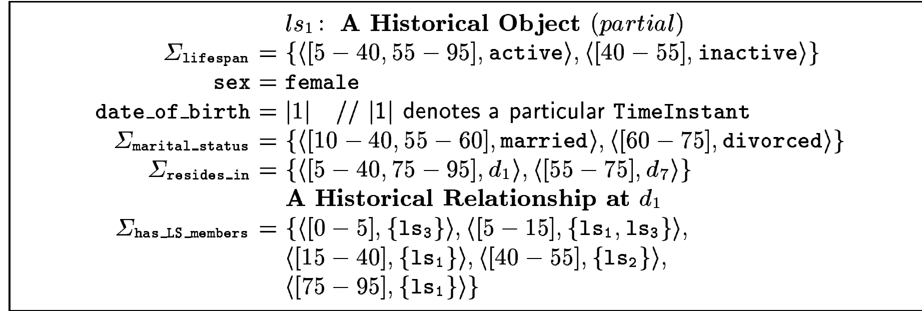
Collection Object Histories An instance of a collection object type is a named object whose component elements are of the same type. This type can be an atomic object type, another collection, or a literal type. A historical collection object type uses the history mechanism to allow the history of change of its composition to be recorded. For example, a named historical collection object called `largest_Enumeration_districts` could be created, where each state is a structure $\langle\tau, \sigma\rangle$, where $\sigma \in$ `set`$\langle$`Enumeration_district`$\rangle$ denotes the `Enumeration_districts` with area above a given limit at each timestamp τ.

Inheritance The ODMG object model supports inheritance-based type-subtype relationships, in which the subtype can specialize, and add to, the state and behaviour of the supertype. This ability gives rise to several important considerations in the Tripod OM regarding specializations of types with historical declarations. Since inheritance is defined to allow the *specialization* of a type, thus providing more specific information in the subtype, it follows that in the Tripod OM, a subtype may only have a more specialized historical nature than its supertype. Therefore, in the Tripod OM a historical type cannot be specialized into a non-historical type, but a non-historical type can be specialized into a historical one. The properties of a type may be added to by a subtype, or they may be refined according to some restrictions. For the same reasons as stated above, it is only possible for a non-historical property to be specialized to a historical property (on the same domain or a more specific one) in a subtype. The reverse is not possible since this will result in loss of information in the subtype. The substitutability property of the ODMG object model requires that any attribute refined in a subclass is viewable with respect to its superclass. Thus, in the Tripod OM, any non-historical property that is specialized in a subclass to become historical, is viewable as a non-historical property in its supertype. Since the value of a non-historical attribute is implicitly valid at the present time (now), the Tripod OM by default accesses a historical type/property snapshot at *now* in such circumstances. This function uses the history ADT's `FilterByTimestamp_equals(now)` operation to compute the current value of a historical property. Note that this value could be undefined if every value in the subtype history exists only in the past.

Containment Constraints The Tripod OM defines a set of containment constraints on historical objects and their properties. For example, the lifespan of a particular historical object instance must always contain the timestamps associated with each snapshot of its historical properties. If this were not the case then a database would contain historical information that is arguably incorrect. These constraints are defined as follows. For an instance o of an historical type, with a set of historical object-valued attributes $attrs$, the lifespan of o must bracket

the lifespan of each of its attributes, i.e.: $\forall\ a \in attrs$, $\texttt{Brackets}(o.\texttt{lifespan}, a.\texttt{lifespan}) = \texttt{true}$. For an instance o of an historical type, with a set of historical relationships $rels$, the lifespan of o must bracket the lifespan of each of its relationships, i.e.: $\forall\ r \in rels$, $\texttt{Brackets}(o.\texttt{lifespan}, r.\texttt{lifespan}) = \texttt{true}$. For a historical relationship r, with state set $\Sigma = \{s_1, \ldots, s_n\}$, where the snapshot value of each s_i is an instance of an historical type, the timestamp associated with each state must be a subset of the lifespan of its corresponding historical object, i.e.: $\forall\ s_i \in \Sigma$, $\texttt{ContainsTimestamp_\underline{surrounds}}\ (s_i.\texttt{v.lifespan}, s_i.\texttt{t}) = \texttt{true}$. For a historical collection object type O, with state set $\Sigma = \{s_1, \ldots, s_n\}$, the lifespan of O must bracket the timestamp in each state: $\forall\ s \in O$, $\texttt{ContainsTimestamp_\underline{surrounds}}\ (O.\texttt{lifespan}, s.\texttt{t}) = \texttt{true}$. Note that the above containment constraints apply only when both concepts are historical. These constraints are invariant properties on historical objects.

An Example For the LS_member type declared in Section 5, each LS_member object will have a history for each of its historical properties. For example, the marital_status attribute is associated with a history with the structure $H_{\texttt{marital_status}} = \langle V, \theta, \gamma, \Sigma\rangle$, where $V \subseteq \{\texttt{single}, \texttt{married}, \texttt{divorced}, \texttt{widowed}\}$, $\theta = \texttt{TimeIntervals}$, $\gamma = \texttt{DAY}$, and Σ maps each different marital_status value to a TimeIntervals value.



ls_1: **A Historical Object** (*partial*)

$\Sigma_{\texttt{lifespan}} = \{\langle[5-40, 55-95], \texttt{active}\rangle, \langle[40-55], \texttt{inactive}\rangle\}$
sex = female
date_of_birth = |1| // |1| denotes a particular TimeInstant
$\Sigma_{\texttt{marital_status}} = \{\langle[10-40, 55-60], \texttt{married}\rangle, \langle[60-75], \texttt{divorced}\rangle\}$
$\Sigma_{\texttt{resides_in}} = \{\langle[5-40, 75-95], d_1\rangle, \langle[55-75], d_7\rangle\}$

A Historical Relationship at d_1

$\Sigma_{\texttt{has_LS_members}} = \{\langle[0-5], \{\texttt{ls}_3\}\rangle, \langle[5-15], \{\texttt{ls}_1, \texttt{ls}_3\}\rangle, \langle[15-40], \{\texttt{ls}_1\}\rangle, \langle[40-55], \{\texttt{ls}_2\}\rangle, \langle[75-95], \{\texttt{ls}_1\}\rangle\}$

Figure 10. Historical, Aspatial Properties

Since LS_member is a historical type, each LS_member instance also has an implicit lifespan attribute which is a history of the form $H_{\texttt{lifespan}} = \langle\{\texttt{active}, \texttt{inactive}\}, \texttt{TimeIntervals}, \texttt{DAY}, \Sigma\rangle$. An LS member ls_1 could have value assignments as shown in Figure 10.

Note that $\texttt{ls}_1$ resided in enumeration district $\texttt{d}_1$, then emigrated (i.e., was logically deleted), then lived in enumeration district $\texttt{d}_7$, then returned to enumeration district $\texttt{d}_1$. The inverse of the resides_in relationship would be defined for each of the referenced objects. For example, for the object $\texttt{d}_1$, has_LS_members would be instantiated as shown in Figure 10.

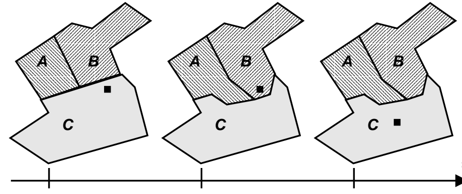


Figure 11. A Historical, Spatial Property

Figure 11 illustrates the capabilities of the Tripod OM when considering spatial properties of LS objects as they change over time. It shows a single LS Region split into three enumeration districts whose boundaries change between the three states. In addition, the LS member object (associated with a `Points` value) can change location independently of changes that enumeration districts undergo.

$\mathtt{ls}_1$ satisfies the containment constraints of Section 5. Note that the implicit `lifespan` attribute of this object indicates that $\mathtt{ls}_1$ had been (logically) deleted from the database during the period $[40 - 55]$ (due to a life event that removed them from the LS). If subsequently the database were updated to reflect that $\mathtt{ls}_1$ had a `marital_status` of `widowed` during $[42 - 53]$, then the stored data must still reflect the OM's containment constraints. It is an implementation issue as to whether this is achieved by either (a) rejecting this new information as inconsistent with the object's lifespan, or (b) accepting the new information, and updating the object's lifespan accordingly. Equally, if $\mathtt{ls}_1$ were (logically) deleted during the period $[20 - 30]$, then this operation can either be rejected or the object's properties can be modified according to the policy set down by the higher layers.

6 Related Work

The contributions reported in this paper build upon the large body of research in the general area of spatial and temporal databases, in both relational and object settings over the last three decades. Much of the recent work in this area has been surveyed in [7]. In general, there are two ways of providing support for spatio-temporal entities. The first of these is to provide tightly-coupled mechanisms for spatio-temporal support, the other is to provide a more general means of supporting the temporal aspects of data, of which spatial entities are just one type. The former of these approaches is exemplified by [25], where a spatio-temporal model and algebra are proposed whose entities are spatial objects (simplicial complexes) that have an integral temporal dimension. While such approaches provide highly specialized mechanisms for maintaining the history of purely spatial entities, these mechanisms have not been extended to aspatial entities. The remainder of this section focuses on data models that use more general mechanisms for storing changes not only to spatial but also to aspatial data, since that is the focus of this paper.

Current research into temporal object models stems from similar work in the relational setting during the 1980s. Such models typically used either tuple or attribute timestamping mechanisms to record the valid time of stored data. The property timestamping approach adopted in this paper can be seen to be an adaptation to an object-oriented setting of the methods employed by, e.g., Gadia [8], who modelled attribute values as functions from temporal elements onto the attribute's domain, where a temporal element is a union of disjoint time intervals. Such attribute timestamping methods have also been used in a purely spatial setting by researchers such as Langran [14], who proposed a spatial vector model in which stored line segments are used as primitives to produce stored polygons. Each of these polygons is then timestamped with its own attribute history using discrete semantics. These techniques however have not been extended to aspatial data. The operations defined over Tripod temporal types exploit constructs first proposed by Güting and Schneider in the early 1990s in a spatial setting. Our temporal predicate operations have their foundations in the interval

algebra defined by Ladkin [13], which in turn extends that of Allen [1] to unions of convex intervals. There are many proposals of temporal relational models, however few have been implemented (see the excellent survey in [12] for details).

There are also relatively few proposals for temporal object models (see the survey in [21]). Of these, the historical object model proposed by Bertino et al. [2] most closely relates to our work. They propose a temporal extension to the ODMG object model, called T_ODMG, that uses property timestamping. The value for each T_ODMG object property is a function of time, using interval-based timestamps that closely resemble the primitive Tripod `TimeInterval` values. Particular attention is paid to modelling objects that migrate to another type during their lifetimes. While the structural component of T_ODMG is well documented, the behavioural aspects of temporal domains is less so. The details of such behaviour are a necessary precursor to the definition of a query language and corresponding optimizer, and without that it is difficult to compare the contributions of this paper with those of [2] in greater detail. There are even fewer proposals for spatio-temporal object models (see the survey in [23]). Of these the MADS model [16] reflects many of the concerns identified by the Tripod object model, including the orthogonal treatment of spatial and aspatial concerns. While the MADS model provides facilities to conceptually model spatio-temporal types and relationships, it does not consider the operations necessary for updating and querying.

Both our data model and that of [2] use a *discrete* model of time. Other proposals exist for data models that capture objects whose properties (spatial and aspatial) are continuously changing. These models are typified by the *moving object* approach adopted in [20] and [17]. Such models 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. This provides an expressive mechanism for the representation of moving points and polygons ([17] only considers points). Querying moving object databases is achieved by extending an existing database algebra through a process called *lifting*. This allows non-temporal kernel algebra operations to be applied to temporal types. It should be noted, however, that 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 by the Tripod data model. 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; for example cadastral, cartographic, and demographic applications.

7 Summary

The aim of the Tripod project is to design and prototype a complete spatio-temporal database system. This paper presents the core spatio-historical data model which forms the foundations of this work. In particular, this paper has presented a collection of primitive temporal types whose foundations lie in the existing spatial ROSE algebra [18]. The intrinsic relationship between these temporal and spatial types promotes consistent and complementary facilities for representing time and space. The paper has also described how these temporal types can be used to underpin the notion of a history, as a generalized mechanism through which both spatial and aspatial change can be recorded over

time, and shown how Tripod's spatial and temporal types and the notion of a history can be used to orthogonally extend the ODMG object model to form a spatio-historical object model. Finally, through the use of an example application, the paper has illustrated how these core modelling concepts are used within a spatio-historical database architecture to provide a formal description of the data structures and operations that are necessary to underpin both a query calculus and native language bindings.

Acknowledgments: Discussions with John Stell and Chris Johnson helped shape the contributions of this paper. Support by the UK Engineering and Physical Sciences Research Council (EPSRC) is gratefully acknowledged. Figure 4 is Crown copyright material reproduced under Class Licence Number C01W0000487 with the permission of the Controller of HMSO and the Queen's Printer for Scotland.

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. *Proc. ECOOP'98*, pp. 41–66, 1998.
3. R. G. G. Cattell, editor. *The Object Database Standard: ODMG 3.0.* Morgan Kaufmann, 2000.
4. C. X. Chen and C. Zaniolo. SQL^{ST}: A spatio-temporal data model and query language. *Proc. ER 2000*, LNCS 1920, pp. 96–111, 2000.
5. C. S. Jensen et al. The Consensus Glossary of Temporal Database Concepts. *Temporal Databases: Research and Practice*, LNCS 1399, pp. 367–405, 1998.
6. L. Fegaras and D. Maier. Optimizing Object Queries Using an Effective Calculus. *ACM TODS*, 25(4):457–516, 2000.
7. A. U. Frank et al. CHOROCHRONOS, A Research Network for Spatiotemporal Database Systems. *SIGMOD Record*, 28(3):12–21, 1999.
8. S. K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM TODS*, 13(4):418–448, 1988.
9. T. Griffiths, A.A.A. Fernandes, N. Djafri, and N.W. Paton. A Query Calculus for Spatio-Temporal Object Databases. In *Proc. TIME'01*, pp. 101–110, 2001.
10. T. Griffiths, N.W. Paton, and A.A.A. Fernandes. An ODMG-Compliant Spatio-Temporal Data Model. `http://pevepc13.cs.man.ac.uk/PrePrints/index.htm`.
11. T. Griffiths, N.W. Paton, and A.A.A. Fernandes. Realm-Based Temporal Data Types. `http://pevepc13.cs.man.ac.uk/PrePrints/index.htm`.
12. L. E. McKenzie Jr and R. T. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4):501–543, 1991.
13. P. Ladkin. *The Logic of Time Representation.* PhD thesis, University of California at Berkeley, November 1987.
14. G. Langran. *Time in Geographical Information Systems.* Taylor and Francis, 1992.
15. V. Müller, N. W. Paton, A. A. Fernandes, A. Dinn, and M. H. Williams. Virtual Realms: An Efficient Implementation Strategy for Finite Resolution Spatial Data Types. *Proc. SDH'96*, pp. IIB.1 – IIB.13, 1996.
16. C. Parent, S. Spaccapietra, and E. Zimányi. Spatio-temporal conceptual models: Data structures + space + time. *Proc. ACM-GIS '99*, pp. 26–33, 1999.
17. D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel Approaches in Query Processing for Moving Object Trajectories. *Proc. VLDB 2000*, pp. 395–406, 2000.
18. R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4(2):243–286, 1995.
19. R. H. Güting, T. de Ridder, and M. Schneider. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types. *Proc. SSD'95*, LNCS 951, pp. 216–239, 1995.
20. R. H. Güting, M. H. Böhlen, M. Erwig, C. S. Jensen, N. A. Lorentzos, M. Schneider, and M. Vazirgiannis. A Foundation for Representing and Querying Moving Objects. *ACM TODS*, 25(1):1–42, 1000.
21. R. T. Snodgrass. Temporal Object-Oriented Databases: A Critical Comparison *Modern Database Systems: The Oobject Model, Interoperability and Beyond*, pp. 386–408. Addison-Wesley/ACM Press, 1995.
22. R. Snodgrass. *Developing Time-Oriented Database Applications in SQL.* Morgan Kaufmann Publishers, 2000.
23. A. Pavlopoulos and C.Theodoulidis. Review of Spatio-Temporal Data Models. Technical report, Department of Computation, UMIST, United Kingdom, October 1998. `http://www.crim.org.uk/`.
24. C. Theodoulidis and P. Loucopoulos. The time dimension in conceptual modelling. *Information Systems*, 16(3):273–300, 1991.
25. M. Warboys. A Unified Model for Spatial and Temporal Information. *The Computer Journal*, 37(1):25–34, 1994.