

CONCEPTS AND METHODS FOR MODELLING TEMPORAL AND SPATIOTEMPORAL INFORMATION

Agnar Renolen

NTNU, 1999

The work presented in this report is a thesis submitted to the Norwegian University of Science and Technology, NTNU (earlier known as the Norwegian Institute of Technology, NTH) as a partial fulfilment for the degree 'doktor ingeniør' (dr. ing).

To Jana

Abstract

This thesis deals with the problem of modelling information that varies over space and time. Four modelling frameworks are investigated. (1) By using *conceptual modelling languages*, and by extending some of these with temporal constructs, it is possible to describe temporal aspects of real world systems at the conceptual level. This way, structural, functional, behavioural, object-oriented and rule perspectives of real world systems can be described in a formal and illustrative way. (2) By using a *directed acyclic graph*, successor-predecessor relationships between objects and the changes that transformed an object from one state into a successive state can be represented. (3) By using *set-theory* it is possible to describe information using mathematics. Moreover, changes in a system can be described in a more formal manner. And finally, (4) by structuring real world entities according to the *object-oriented* model, which is considered to be the most expressive and powerful modelling framework. In addition to these four frameworks, three ‘byproducts’ are presented in this thesis. These includes generalization and data reduction in spatiotemporal data sets, implementation of bitemporal lifespans and temporal fuzzy regions.

ACKNOWLEDGEMENTS

This work is funded by the Research Council of Norway (NFR), grant 31387. Thanks to my supervisors professor Jan Terje Bjørke and professor Morten Dæhlen for their advice and encouragement during this thesis, and also to Terje Midtbø who stepped in to take care of the formal arrangements during the last six months of the project. Terje also proof-read one of my papers.

I would also like to thank professor Donna J. Peuquet and professor Alan M. MacEachren for allowing me to visit the Department of Geography at Penn State University during the 1997–1998 academic year. Thanks to my friends and colleagues at PSU; in particular to Monica Wachowicz and Liujian Qian who gave me invaluable feedback to my work. Taking part in many of the fruitful discussions were also Jeremy Mennis, Rob Edsall, Dan Haug and Ryan Baxter. Also thanks to Tereza, Maggie, Marco, Heejun, Thomas, Netra and Nalini, and all the other at the department and elsewhere at PSU who I learned to know during my visit. It was a real pleasure to meet yall.

Also thanks to Svein Erik Bratsberg and Ragnar Normann for reviewing and proof-reading my work, and to the reviewers of the articles and papers that I submitted to journals and conferences. I also had interesting and fruitful discussions with David Skogan, Bjørn Skjellaug and professor Finn F. Knudsen. Their comments have been valuable and have helped me to increase the standard of this work. Some helpful comments were also provided by professor Arne Sølvsberg.

Finally, I would like to thank my parents, Bjørg and Egil for everything they have done for me, and of course to Jana who really brightened things up for me during the last one and a half year of this project. I dedicate this thesis to our future family.

CONTENTS

1	Introduction	11
1.1	Time and Temporal GIS	11
1.1.1	The Time Domain	12
1.1.2	Temporal Data	13
1.1.3	Behaviour of Attributes and Attribute Values	14
1.1.4	The Space Domain	15
1.1.5	Spatiotemporal Data Models	15
1.2	Towards a Better Temporal GIS	18
1.3	Thesis Outline	21
1.3.1	Chapter 2: Conceptual Modelling in Spatiotemporal Information System Design	21
1.3.2	Chapter 3: Conceptual Modelling of Spatiotemporal Data using Graph Structures	22
1.3.3	Chapter 4: A Framework for Temporal and Spatiotemporal Modelling Based on Set-Theory	23
1.3.4	Chapter 5: On the Design of Temporal Object-Oriented GIS	25
1.3.5	Chapter 6: On Generalization and Data Reduction in Spatiotemporal Data Sets	25
1.3.6	Chapter 7: Indexing and Representing Bitemporal Lifespans Using Binary Trees	26
1.3.7	Chapter 8: Temporal Fuzzy Regions: Concepts and Measurements	26
1.3.8	How to Read this Thesis	27

2	Conceptual Modelling	29
2.1	Introduction	29
2.2	An Introduction to Conceptual Modelling	30
2.2.1	A Brief Review of Conceptual Modelling Perspectives	31
2.2.2	Information, Data and Model Domains	33
2.3	Structural Modelling and the Time Dimension	34
2.3.1	Entity Relationship Models	34
2.3.2	The ERT language	35
2.4	Modelling Processes	37
2.4.1	Data Flow Diagrams	37
2.4.2	Demos Activity Diagrams	40
2.5	Modelling Behaviour	42
2.6	Object-Oriented models	46
2.7	Rules, Knowledge and Business Policy	49
2.8	Concluding Remarks	51
3	History Graphs	53
3.1	Introduction	53
3.1.1	Change patterns	53
3.2	Events and States	55
3.2.1	The topology of time	55
3.2.2	Representing The State-Based Approach	56
3.2.3	The Event-Based Approach	57
3.2.4	Combining Events and States into One Graph	58
3.3	Extending the Model	60
3.3.1	Events do have Duration	60
3.3.2	Continuous Processes	61
3.3.3	Sudden Changes in Continuous Processes	62
3.4	Compound Objects	64
3.5	Temporal Topology in the Data Structure	66
3.6	Concluding Remarks	67
4	The Temporal Set-Theory	69
4.1	Introduction	69
4.2	The Time Model	70
4.2.1	Different Models of Time	70

4.2.2	Lifespans and Time Intervals	72
4.2.3	The Bitemporal Time Model	73
4.3	The Temporal Set Theory	74
4.3.1	Background and Related Work	74
4.3.2	The Classic (Naive) Set Theory	75
4.3.3	Attributes and Domains	76
4.3.4	Functions over Time	77
4.3.5	Temporal n -tuples and the Product of Temporal Domains .	78
4.3.6	Functions and Operators over Temporal Domains	80
4.3.7	Temporal Sets	81
4.3.8	Common Operations on Temporal Sets	83
4.3.9	The Temporal Cartesian Product and Relations on Tempo- ral Sets	86
4.4	Extending to the Bitemporal Domain	89
4.4.1	The Bitemporal Set Theory	89
4.4.2	Functions over bitemporal attributes	90
4.4.3	Bitemporal Sets	91
4.5	Tropology	92
4.5.1	Defining Tropology	92
4.5.2	Related Work	94
4.5.3	Valid Time Tropology	96
4.5.4	Causal Relationships	100
4.5.5	Conceptual Modelling of Tropology	101
4.5.6	Bitemporal Tropology	103
4.5.7	Transaction Types	104
4.5.8	Conceptual Modelling of Bitemporal Tropology	106
4.6	Implementing the Approach Using Abstract Data Types	107
4.6.1	Background and Related Work	107
4.6.2	Implementing Temporal Attributes	109
4.6.3	Implementing Temporal Sets and Sequences	110
4.6.4	Implementing Spatiotemporal Attributes	111
4.7	Concluding Remarks	113
5	On Object-Oriented Temporal GIS	115
5.1	Introduction	115
5.2	The Basic Object Model	117

5.3	Temporal OO-Databases and Temporal GIS	118
5.4	The Temporal Object-Oriented Model	122
5.4.1	Object and Attribute Behaviour	122
5.4.2	Observing Temporal Objects	123
5.4.3	Object Evolution and Mutators	124
5.4.4	Validations and Corrections	127
5.4.5	Class Evolution	127
5.4.6	The Basic Class Hierarchy	130
5.4.7	Spatial and Spatiotemporal Objects	132
5.5	A Temporal Map over Europe	133
5.5.1	Assumptions	133
5.5.2	Modelling the Temporal Database Conceptually	134
5.5.3	Designing the Interface for each Class	137
5.6	Querying the Database	141
5.7	Concluding Remarks	143
6	Generalization and Data Reduction	145
6.1	Introduction	145
6.1.1	Data Reduction vs. Generalization	146
6.1.2	Generalization Operators	147
6.2	Utilization of Generalization and Data Reduction	148
6.2.1	Generalization as a Tool for Data Retirement	148
6.2.2	Multiple Scale Temporal Databases	149
6.2.3	Map Animations	150
6.3	A Notation for Temporal Data	150
6.4	Methods of Generalization and Data Reduction	151
6.4.1	Spatial Generalization	152
6.4.2	Attribute Generalization	156
6.4.3	Temporal Generalization and Data Reduction	156
6.4.4	Cartographic Generalization and Animated Maps	159
6.5	Concluding Remarks	161
7	The BL-Tree	163
7.1	Introduction	163
7.2	Updates and Bitemporal Lifespans	166
7.2.1	A Conceptual Bitemporal Relation Schema	167

7.2.2	Examples	168
7.2.3	Transaction Types	170
7.3	The BL-tree	172
7.3.1	The Principle of the BL-tree	172
7.3.2	Searching a BL-tree	173
7.3.3	Building BL-trees	174
7.3.4	Using the Insertion Algorithm	177
7.3.5	Query Algorithms	177
7.3.6	Representing One Single Bitemporal Lifespan	181
7.4	Concluding Remarks	181
8	Temporal Fuzzy Regions	183
8.1	Introduction	183
8.2	Change and Time	183
8.3	Fuzzy Regions	184
8.4	Temporal Fuzzy Regions	186
8.4.1	Basic Concepts	186
8.4.2	Some Measurements of Change of Fuzzy Regions	187
8.5	Examples	188
8.5.1	Change of the snapshot of \mathcal{A}	188
8.5.2	Change of the Distance from p to \mathcal{A}	188
8.5.3	Change of the Area of \mathcal{A}	192
8.6	Conclusions and Future Work	192
9	Conclusion	193
A	Set-Theory and Related Topics	195
A.1	The Naive Set-Theory	195
A.1.1	Basic Notation	195
A.1.2	Operations on Sets	196
A.1.3	Functions	197
A.1.4	Relations and Cartesian Product	198
A.2	Point-Set Topology	199
A.2.1	Metrics and Metric Spaces	199
A.2.2	Open and Closed Sets	200
A.2.3	Some Types and Properties of Point Sets	200

A.2.4	Topological Spaces and Homeomorphisms	201
A.3	Fibre Bundles	202
A.4	Algebraic Topology	203
A.4.1	Simplexes and Complexes	203
A.4.2	Oriented Simplexes and Complexes	204
A.4.3	Simplicial Homology Groups	205
A.4.4	Polyhedrons and Manifolds	206
A.5	Fuzzy Set Theory	207
A.5.1	Basic Definitions	207
A.5.2	Basic Operations on Fuzzy Sets	208
A.5.3	Fuzzy Relations	209
A.5.4	Fuzzy Numbers and Fuzzy Regions	209
A.6	Graphs	209
A.6.1	Graph Types	210
A.6.2	Terminology of Graphs	210

INTRODUCTION

The main subject of this thesis is modelling of *spatiotemporal information*, information where both the position and the history of the information are of interest to the users. The term *spatial* is used rather than *geographical* to emphasize the fact that the location and geometric descriptions need not refer to a geodetic coordinate reference system. The aim of this thesis is to find frameworks that deal with all, or as many as possible, aspects of temporal and spatiotemporal information in order to describe and model some selected part of the real world, also known as the universe of discourse (UoD). Four frameworks for describing spatiotemporal information have been investigated. These are as follows:

- conceptual modelling languages,
- directed acyclic graphs,
- set-theory and
- the object-oriented paradigm.

In addition, some other issues related to temporal and spatiotemporal modelling have been studied.

The results are presented as separate reports given in the chapters 2 to 8.

1.1 *Time and Temporal GIS*

In this section, the basic concepts and definitions of time, temporal databases and temporal GIS will be given along with references to the most important work in these fields.

1.1.1 The Time Domain

An initial step in designing a temporal database is to choose a suitable time model. Since we in this thesis are concerned about geographical information, a Newtonian time model is considered sufficient. On the other hand, if nuclear particles or astronomical phenomena are being studied, a relativistic time model has to be implemented.

Conceptually, we distinguish between three structural models of time, *linear*, *branching* and *cyclic* [Ben83] [Wor95]. In linear time, time advances from the past into the future in a totally ordered manner. In branching time, time is considered linear until the present time, then time advances into several possible futures. In many applications, e.g. in archaeology, it also make sense to model several possible pasts. Cyclic time is associated with periodic patterns of time such as years, weeks and days.

We also distinguish between two models of time in terms of the *density* of the time line [Ben83]. The *continuous* model considers the time dimension to be isomorphic to the real numbers. Each point on the continuous time line is called an *instant* and, due to the density requirement, an instant can have no duration. The *discrete* model considers the time dimension to be isomorphic to the integers. Each integer correspond to an atomic unit of time called a *chronon*. A chronon is thus the smallest duration of time that can be represented in this time model. Finally, the time model can also be characterized by its boundedness, i.e. if there exist a beginning of time and possibly an end of time.

Even if time naturally conceptualizes into a one-dimensional domain, there are many types of times that can be associated with facts. First there is the time when a future fact is forecasted or decided. Then, there is the time when the fact become true in the reality, then there is the time when the fact is observed and measured, then there is the time when the observation is recorded in the database, then there is the time when the data in the database is used to derive new data, then we have the time when decisions are made upon the data, then there is display or print-out time time, and yet there is the time when data from one database is added to another database [JS96].

Thus, time has to be regarded as multidimensional. However, two orthogonal time dimensions are emphasized in temporal database research: *valid time* which is the time when a fact is true in the real world, and *transaction time* which is the time when the fact is current in the modelled reality, i.e. in the database [SA86]

[Sno92]. The valid time dimension is considered to start at some infinite past (the beginning of time), and progress into some infinite future (the end of time), whereas the transaction time dimension is considered to stretch from the time of creation of the database up to the present time. The other time dimensions can be captured by using time as an ordinary attribute. This is known as *user-defined time*.

1.1.2 Temporal Data

Temporal data differs from non-temporal data in one major aspect. Instead of having only one value, there is one value for each instant of time. Important in a large body of research on temporal data and databases is the dichotomy between *states* and *events*. Normally, a state is considered to hold over an extent of time, whereas events delimit states, and are considered to be instantaneous [JS96].

Suppose we have a record that represent the state of an object over an interval of time, this interval would then be the *lifespan* of the record. A lifespan can be a *time interval* which is a connected sequence of chronons (or instants). But we could also imagine that the record is valid during one interval, then to be invalid for some interval, and then valid again during some later interval. The lifespan of such a record is called a *temporal element* which is defined to consist of a finite set of disjoint time intervals.

Depending on the application, a data model may support valid time, transaction time or both. A model that supports only valid time is called a *historical* or *valid-time* model, a model that supports only transaction time is called a *roll-back* or *transaction model* and a model that supports both times is called a *bitemporal model*. A model that supports neither is known under the retronym *snapshot* model [SA86] [Sno92].

In bitemporal databases, there are three types of updates concerned with the time when updates are made. Consider that an update was made at transaction time TT concerning a change that occurred at valid time VT . A *retro-active* update is an update where $TT > VT$, i.e. the update is done after the change occurred in reality. If $TT < VT$, the update is called a *pro-active* update, and if $TT = VT$ we are speaking of a real-time updates.

In bitemporal databases, the lifespan of a record may be a *bitemporal interval*, which is a region in two-space of valid time and transaction time with sides parallel to the time axes [JCE⁺94]. In many cases, we do not know for how long a certain fact is going to hold in valid time. Lifespans in these cases, may stretch into infinity,

and the symbol ∞ is often used to hold this value. But, since the transaction time dimension has an upper bound at the present time, bitemporal periods are delimited by the symbol *now* or *utc* (until changed) which hold the present time.

Thus, when a record is updated in a database, a new record must be added, while *now* values of the old record's lifespan is replaced with the time of the transaction. This way, several versions of the same logical entity may be spread over several records in the database. In order to identify the records that are associated with the same entity, each record is equipped with a *surrogate key* that identifies the logical entity [JS96]. This way, it is possible to derive successions from the lifespans of the record, but this requires some computing effort. It is therefore of help to store explicit succession links in the data set [CT95].

1.1.3 Behaviour of Attributes and Attribute Values

An entity have many properties. A property that is considered to be constant over time is called a *time-invariant* property. A property that changes over time, is said to undergo *temporal variation*. We describe them using *temporal attributes*.

Figure 1.1 shows three kinds of behaviour of attribute values. Some types of attributes, such as salary, are *stepwise constant* and can only change through events that have a limited duration or no duration (instantaneous changes). *Continuously changing attributes* may also exhibit a wide range of behaviour. It may change smoothly, irregularly or uniformly. A third type of behaviour is called *discrete attributes*. Attributes of this type are stored in so-called *time sequences* [SS93] [MP93].

In some cases, successive records may be associated with non-contiguous lifespans, e.g. if the records represent observations that are made at certain points of time. In order to derive values between observations, i.e. for times when no value is explicitly stored in the database, a *derivation function* is used [JS96]. Three types of derivation functions exist: Attributes that are stepwise constant, are associated with a *stepwise constant* derivation function, attributes that are smoothly varying are associated with *interpolation functions*, whereas a *discrete* derivation function returns values only for those times where a value is explicitly stored in the database.

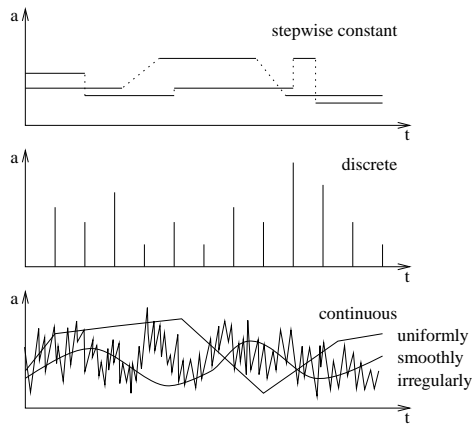


Figure 1.1: Types of changes to an attribute.

1.1.4 The Space Domain

Traditionally, the locational attributes of geographical information systems have utilized coordinate systems that are homeomorphic to the Euclidean plane. Using such coordinates systems locally in the map projection plane is sufficient. Thus, even if the topology of the earth is homeomorphic to the sphere, this thesis assumes the Euclidean space model.

1.1.5 Spatiotemporal Data Models

Before computers, geographical information was stored in the form of cartographic paper maps. The paper map model thus became the basis upon which the first data models for GIS were developed. Two data model paradigms emerged. One is grid-based or location-based and is known as the *raster paradigm*. The other is geometry-based or object-based and is known as the *vector paradigm*.

Al-Taha and Barrera [AB90] outline two types of models for temporal GIS. In the *change-based* model, facts are recorded to be valid during certain time intervals. Among the limitations of this model is the fact that all changes are considered to be instantaneous, such that continuous processes cannot be represented. In the *time-based* model only one basic change is recognized: the passage of time.

One of the simplest spatiotemporal models is the *snapshot model*, which is

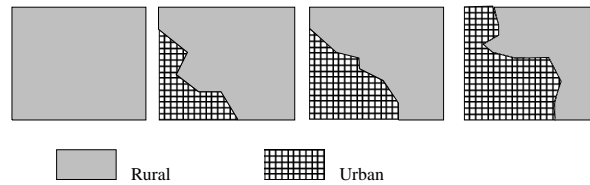


Figure 1.2: The snapshot model.

illustrated in Figure 1.2. In this model, the state of the UoD is given at regular or irregular intervals in different maps, one complete map for each time interval. The drawback with this model is that two snapshots contain much of the same data and that events in the real world cannot be identified. The problem of data redundancy is mostly a problem with vector-based data models. But for raster-based data models, such as remotely sensed images, the approach is useful since the data model correspond to the way such images are recorded. Hamre [Ham95] has implemented a temporal GIS which integrates such images together with other temporal information.

Another simple method is to use a model where each record is *timestamped* with a time interval indicating the period of time during which the record is valid. Hunter and Williamson [HW90] and Galetto and Viola [GV94] have implemented such an approach and show that time slices can be retrieved easily by simple queries. However, such a model spreads the different versions of the same object over several non-related tuples in the same table. This model can be improved by adding explicit references to preceding and succeeding versions of each object. Ramachandran, MacLeod and Dowers [RMD94] deal with this issue in an object-oriented model. They implement a *Temporal Change Object* which is an object that consists of a set of references to past (historic) and future (scheduled) versions together with a reference to the current version.

The *space-time composite* model, which is illustrated in Figure 1.3, has been proposed by Langran [LC88]. It is based on the principle that every line in space and time is projected down to the spatial plane and intersected with each other to create a polygon mesh. Each polygon in this mesh is associated with its own attribute history. Each new amendment is intersected with the already existing lines, and new polygons are formed with individual thematic histories. The methods can be implemented by storing the spatial component in an ordinary snapshot GIS,

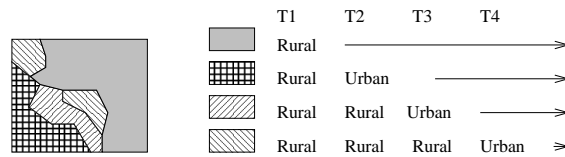


Figure 1.3: The Space time composite data model.

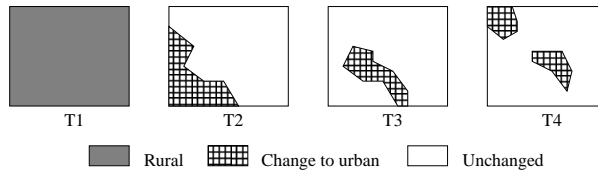


Figure 1.4: The amendment vector model.

while the thematic attributes can be stored in a temporal database. The results that were obtained looked promising, but only smaller data sets were tested.

However, this data model will also have some redundancy since two contiguous objects may have full or partial common history if they at some point were (or are) parts of the same logical object. Moreover, the model is vulnerable to fragmentation as the number of polygons can grow exponentially with time (Saafeld's work [Saa91] supports this theory).

An alternative approach is to represent the data only in terms of changes or events. Langran briefly discusses this *event-oriented* method as a vector approach (Figure 1.4), where the current state can be found by applying amendments from a base map or current state map [LC88]. In some cases, where the the number of changes are relatively small in relation to the number of objects, this can be a quite useful method. Ohsawa and Kim [OK98] implemented such a vector approach using an *inverse differential script*. In this method, the current state is represented by a complete snapshot, while historic time slices can be obtained by tracing through the change script in a backward manner.

A raster-based method for time sequence data has been implemented by Peuquet and Duan [PD95]. This model consists of a base state database and one compactly stored change image for each event (i.e. change). They show that the model is space saving compared to the snapshot model, and efficient in terms of common query types.

The most abstract way to implement a spatiotemporal data structure is in the form of a three- (x, y, t) or four-dimensional (x, y, z, t) polygon mesh. Such vector models have received little attention in the GIS community. Although, two alternative models have been described by Worboys. In [Wor92], regions are described by so-called simplicial T-complexes, where each complex is a collection of so-called T-simplexes. A 2-T-simplex is then a triangular prism covering a triangular area in space, and an interval in time. In [Wor94a], this model has been extended such that each T-simplex is associated with a bitemporal lifespan. However, these models has the same fragmentation problem as the space-time composite model.

The spatial objects of the vector model, may be included as the locational attribute of objects in an object-oriented data model. The advantage of the object-oriented paradigm, is that it allows us to encapsulate an entity and its entire history into one single object [KRS90]. Misund, Skogan and Bjørnås [Mis93][BS93] proposed such a model where different versions of cartographic objects can be accessed by indicating a time parameter as well as other parameters (such as map scale).

1.2 *Towards a Better Temporal GIS*

Most of the data models presented in the previous sections fall into the category of change-based models. Therefore, they are not good at capturing the semantics of the temporal real world, e.g. they are not able to handle information that is continuously changing over time. In order to improve this, we need a framework where such information can be conceptualized. This framework should be based on ontology which is the branch of philosophy that deals with modelling the real world.

One approach studied in this thesis, is to use conceptual modelling, which is a technique that have been developed within the field of information systems engineering. Over the last decade, we have seen an increasing interest in extending existing modelling languages to capture the semantics of geographical and temporal information. The so-called ERT model (entity-relationship-time model), which is an extended entity-relationship model designed for temporal information, is described in various sources [MSW92] [TL91b] [TL91a] [TLW91]. Hadzilacos and Tryfona [HT96] [HT97] have presented another extension of the entity-relationship model for geographical applications. Tveite [Tve92] has presented a similar exten-

sion that also provides some temporal constructs, while Spaccapietra and Parent et al. [SPZ98a] [PSZ⁺98] presents a more comprehensive formalism for spatiotemporal applications. However, these models are only dealing with the structural perspective of the UoD.

Interesting approaches can also be found in the field of artificial intelligence (AI). Since AI deals with knowledge representation, many applications of AI need to handle knowledge that is varying over space and time. In McCarthy and Hayes' situation calculus [MH69], knowledge is represented by a set S of *situations* (both real and hypothetical situations) that captures the state of the universe at instants of time. Then, information about the real world can be drawn in the form of *fluents* which are functions from S into S (situational fluent) or into the set of booleans (propositional fluent). Thus, a fact can be drawn from S by a propositional fluent. This means that the fluent $raining(p)(s)$ will return *true* if it is raining at position p in situation s . Changes and actions are modelled as situational fluents, consequently the fluent $open-door(d)(s)$ will return a new situation in which the door d is open. Hence, situational fluents represent functional relationships between different situations.

Allen's general theory of action and time [All84], which is partly based on [Mou78], describes the world by a set of temporally qualified assertions describing knowledge about past, present and future time. The static aspects of the world are captured by *properties* whereas the dynamic aspects of the world are captured by *occurrences*. Occurrences can again, according to [Mou78], be divided into *events* which focus on performances and achievements, and *processes* which focus on activities that occur over intervals of time. Allen's theory also provides means of representing causal relationships. Two types of causal relationships are discussed. *Agent causality* means that an occurrence was caused by some agent (usually a person or organization), while *event causality* means that an occurrence was caused by some other occurrence. Allen's theory is supported by a temporal logic (or tense logic) along with a set of relationships between time intervals [All83].

In recent research of temporal GIS we have seen an increasing interest in modelling the dynamic aspects of information. Yuan [Yua98] states that such an approach is necessary to enable temporal GISs to efficiently handle queries about static and dynamic aspects of the UoD. Claramunt and Thériault [CT95] recognizes the dichotomy between states and events, and have explicit representation of events in addition to state records in their data model. In Peuquet's Triad model [Peu94], spatiotemporal information is viewed from *where* (spatial), *when* (time)

and *what* (thematic) perspectives . A fourth corner can be added to represent the objects which are characterized by all those three perspectives and Cheng [CM98] adds the fifth corner, processes, which can be related to all the other four corners.

In the recent literature, several taxonomies of changes have been presented. Claramunt and Thériault [CT95] distinguish between three types of changes or processes: The first type of changes include those that concern the evolution of a single entity. This includes basic changes as appearance, disappearance and transformations. The second type of changes includes those that involve functional relationships between several entities. This includes replacement processes such as succession and permutation, and diffusion processes such as production, reproduction and transmission. The third type of changes includes those that are the evolution of spatial structures involving several entities. This includes restructuring processes such as split, merge and reallocation; for example of polygons in a polygon mesh.

The change-aspect of information can also be visualized using a conceptual language, such as the language developed by Hornsby and Egenhofer [HE97]. Using this language, a set of 21 different types of changes were identified among objects. A similar taxonomy has been presented for Voronoi diagrams [MAGM98].

Another interesting approach to temporal information modelling is to model features as *functions over time*. This can be done either in the form of abstract data types (ADT's) or in the form of object-oriented models. Wu and Dayal [WD92] presented such a temporal object-oriented model, together with a query language based on the OODAPLEX object-oriented data model. Erwig et al. [EGSV97] [ESG97] presented a model where real valued functions over time, called *moving reals*, as well as *moving points* and *moving regions* were studied. They suggest that such functions could be implemented by linear approximations using data points. They show that this model can help strengthen the expressive power of queries. E.g. suppose that we need to determine at which time two points were closest to each other. This can be achieved by first determining the distance between the two points as a function over time, and then find the minimum value of this function.

The work described in this section provides, in my opinion, a better framework for developing a future temporal GIS. The aim of this thesis, is to try and find a unified framework that could deal with as many aspects as possible. It has been done by studying conceptual modelling techniques in combination with a set-theoretical approach, while at the same time accommodating the concepts described above.

1.3 Thesis Outline

The thesis is written as independent reports, which are presented in the chapters 2 through 8. Nevertheless, comments and introductory stuff are therefore repeated throughout the thesis. Thus, each chapter can be read independently.

The main contribution of this thesis is in the four articles given in chapters 2 through 5 respectively. Additionally, three ‘byproducts’ are given in the Chapters 6 through 8.

Some of the articles are published as conference papers, while others are submitted to journals and conference proceedings. The comments from the reviewers have been of great value, and their comments have been incorporated in this thesis. In the remainder of this section, we give a brief overview of each article and how they are related.

1.3.1 Chapter 2: Conceptual Modelling in Spatiotemporal Information System Design

One of the main topics of the article is the importance of describing the UoD in a way that is closer to human perceptions of the real world, and without any concern about how to implement it. This type of modelling falls in under the branch of philosophy known as *ontology*, and is carried out using *conceptual modelling languages*. Moreover, conceptual modelling languages are often graphical and easy to understand, even by non-experts. A conceptual schema can therefore be understood by a broader audience, and several persons and potential users can contribute to the earlier stages of the GIS development. This is more important in temporal GIS than in non-temporal GIS as there are more aspects to the temporal world than there is to a single snapshot.

In the field of information systems engineering, a number of standardized modelling languages have been developed over the years. In addition, we have seen a number of special purpose languages that adds additional information about the UoD, such as spatial and temporal characteristics.

The conceptual modelling languages can be divided into a set of classes according to the perspective of the UoD they aim to describe. In [SK96], seven different perspectives of conceptual modelling are described. At least five of these are of special interest in the development of a temporal GIS. These are the structural perspective, the functional perspective, the behavioural perspective, the object-

oriented perspective and the rule perspective. These perspectives were studied with the utility in temporal GIS development in mind.

The article gives a brief introduction to conceptual modelling, and then describes each of the five perspectives in more detail with emphasis on temporal and spatiotemporal applications. Along with each perspective, a few examples on spatiotemporal systems are given. There is also an original contribution in the article in form of a generic behaviour model. In this model, objects are considered to be either alive or dead, and if they are alive, they are either in a state of (possibly gradual) change or they are in a static state. This model has provided a basis for many of the other models and frameworks presented in this thesis.

Additionally, the article briefly describes the history graph notation. This notation, which is considered to belong to the behavioural perspective is described in detail in Chapter 3.

The article has been accepted for publication in *Transaction in GIS* [Rened]. An earlier (and shorter) version of the article was presented at the ScanGIS conference in 1997 [Ren97].

1.3.2 Chapter 3: Conceptual Modelling of Spatiotemporal Data using Graph Structures

The main topic of the article is to extend the dichotomy between events and states to a more general framework. In this framework, successive relationships between *states* of objects are preserved through *event* or *change* objects. Both states and events can have a duration and are associated with a time-interval. Three cases are given. In the first case, the events are instantaneous whereas the states have duration, e.g. the splitting of parcels in a cadastral database. At the other extreme we find the case where the states have no duration and changes have duration, e.g. glaciers that are continuously changing. In the third case, which lies in between the two extremes, both events and states have duration, e.g. the position of a ship that is sailing between harbours.

Temporal relationships between states and events are preserved explicitly in the model and linked together in a directed acyclic graph called a *history graph*. This graph has been given a graphical notation as a conceptual modelling language. Based on the notation of history graphs, a set of seven basic change types were identified. These are creation, destruction, alteration, reincarnation, split, merge and reallocation. Furthermore, some other related issues are discussed in the ar-

ticle. These include the problem of compound objects, temporal topology and implementation.

Some related references are not cited in the article. These include Claramunt and Thériault [CT95] [CT96] and Hornsby and Egenhofer [HE97]. These references were not known to me at time of writing, or they were not yet published. Claramunt and Thériault both give a taxonomy of change types and present a model where changes and events are represented as explicit events. But, they do not timestamp events in the sense that they can have duration. Hornsby and Egenhofer presented a conceptual language and gave a taxonomy of change types that was more comprehensive than mine. Moreover, they also studied changes in compound objects, but they did not have an explicit representation of change objects. On the other hand, they had a better terminology which is partly incorporated in the final version of this article.

In Chapter 4, the history graph model has been extended into the bitemporal domain with concepts from [HE97], with set objects, and with causal relationships. Furthermore, it has been given a more formal semantics using the temporal set-theory. In Chapter 4 the term *tropology* also appears for the first time. This term is a more precise term than the term *temporal topology* that appears in Chapter 3 and in most other literature. The term temporal topology can also be understood as ‘changes in topology over time’, whereas tropology is more generally defined as ‘the study of changes’.

In general, the history graph model have been so fundamental in this research that it is repeated in many other articles. This applies to the Chapters 2, 4 and 6.

An early version of this article was presented at the BrnoGIS conference in 1996 [Ren96]. Some of the most important changes from the original BrnoGIS paper include change in terminology, particularly about names on change types.

1.3.3 Chapter 4: A Framework for Temporal and Spatiotemporal Modelling Based on Set-Theory

It is well known that the set-theory provides a general framework for many applications of computer science. The set-theory is the backbone of the relational database model; it is utilized in object-oriented database systems and in programming languages. Branches of the set-theory, such as point-set topology and algebraic topology have also been of great interest to the GIS community.

However, the temporal database theories that emerged in the database com-

munities were extended from the snapshot database theories, and not based on a *temporal set-theory*. But, a temporal set-theory was nowhere to be found in the literature. This article is an attempt to propose such a theory, and to discuss some of its applications in temporal GIS. The article is considered to be the main contribution of this thesis.

In the temporal set-theory, everything is considered to be functions over time: attributes, n -tuples, sets and relations. We can define functions where the arguments as well as the results are functions over time, e.g. if two reals are given as functions over time, their sum is also a function over time. Each function is associated with a *lifespan* which is the set of times during which the function returns a non-null value. If the functions were defined as partial functions, the lifespan would correspond to the domain of the functions. But, in order to avoid the problem of obtaining values for times outside the lifespan, the symbol \perp (null) is introduced, such that a value can be returned for all times. In addition, the article also includes some initial definitions of the *bitemporal set-theory*.

Using the temporal set-theory, tropology can be defined in a more formal way. The article presents four basic tropological operators. These are *creation*, *destruction*, *modification* and *causality*. By combining these operators, a selection of 18 different types of changes is defined. The tropology is associated with the history graphs and the history graph model have been extended to accommodate the new concepts introduced in the article.

Finally, the problem of implementing different functions over time is discussed. Data models for simple attributes, spatiotemporal attributes as well as temporal sets are presented. The results of this section is used in Chapter 5, where a model for a temporal object-oriented GIS is investigated.

A shorter version of this article was presented at the NewDB'98 workshop that was held in conjunction with the ER'98 conference in Singapore. The proceedings of this workshop has been published in volume 1552 of Lecture Notes in Computer Science [Ren98].

The main inspiration to this article was drawn from two research reports written by a research group at FernUniversität Gesamthochschule in Hagen, Germany [EGSV97] [ESG97]. Interestingly, one of these reports [ESG98] were presented at the NewDB'98 workshop in the same session as my paper.

1.3.4 Chapter 5: On the Design of Temporal Object-Oriented GIS

The article presents a basic object-oriented model for spatiotemporal data. The model has emerged by combining and integrating various concepts of temporal object databases presented in literature, together with concepts from the temporal set-theory.

The article comprises of three main parts. In the first part, the basic object model is presented together with a description of the concepts of temporal object-oriented databases. In the second part, the concepts of temporal object-oriented databases are exploited to make a generic class-hierarchy for an object-oriented temporal GIS. In this hierarchy, emphasis is on the interface and the expected behaviour of objects, and not on the internal implementation. Amongst the classes in the class hierarchy are *set*-classes and *temporal set*-classes, spatial atoms and spatially referenced objects.

The third part gives an example where the basic class hierarchy is utilized. In this example, which aims to describe a temporal map over Europe, many different aspects of temporal information is dealt with. We have the population of countries and cities which may vary continuously over time, while borders only changes instantaneously. A country can be a monarchy at one time, and a republic at another time. A city can be in one country at one time and in another at another time, and so on. The coastline, is considered to be a static entity, whereas multinational organizations, such as EU, are modelled by temporal sets.

The advantage of the object-oriented models are that they provide more modelling power and is closer to the way humans perceive the real world. This way, it is easier to capture the semantics of the real world in the data model. Another advantage is that operations can be added to each class as desired by the developer in order to enable specific queries to be answered or specific tasks to be performed. The problem of the model is that the internal structure of the objects is not known from outside, which means that it is hard for the query processor to optimize queries.

1.3.5 Chapter 6: On Generalization and Data Reduction in Spatiotemporal Data Sets

This article discusses four aspects of data reduction and generalization in spatiotemporal data sets. Generalization and data reduction in the time dimension, in the spatial dimensions, for visualization and animation purposes, and for the

utility of these approaches in GIS.

A clear distinction is made between data reduction and generalization, which are considered to be conceptually different issues. The distinction conforms to the way we distinguish between the terms data and information. In data reduction, the aim is to reduce the volume of the data without influencing the information that the data convey. In generalization, the aim is to reduce the amount of information conveyed by the data set. This, in turn, will often result in a data reduction, but not necessarily. If not, data reduction should be performed subsequent to generalization.

The approach of this article has been to take the spatial transformations discussed by McMaster and Shea [MS92], and study how these should be implemented in a spatiotemporal context. Most of the spatial transformations can also be migrated to the time dimension, e.g. in a map it is common to move two objects apart to emphasize their topological relationships if the symbols overlap. In the time dimension, the same operator can be utilized to move two successive events apart in an animation to emphasize that they do not occur at the same time.

This article was presented at the ScanGIS'99 conference in Aalborg, Denmark [Ren99b].

1.3.6 Chapter 7: Indexing and Representing Bitemporal Lifespans Using Binary Trees

This article is a result of a study of bitemporal data and the problem of representing bitemporal lifespans of objects in computers. The *BL-tree* introduced in this article is a result of the discovery that such lifespans could be represented by hierarchical data structures, more specifically by using binary trees. The name 'BL-tree' means Bitemporal Lifespan tree or Binary 'L'-tree, both equally descriptive.

The BL-tree has been implemented and work well. I believe that the BL-trees would be useful in some applications, although the method is more awkward in comparison to other methods.

1.3.7 Chapter 8: Temporal Fuzzy Regions: Concepts and Measurements

This article is an offspring from the temporal set theory that is a result from discussions between me and my supervisor, professor Bjørke. A fuzzy region can be

described as a function from the space $\mathbf{X} \times \mathbf{Y}$ into the unit interval $\mathbf{I} = [0, 1]$:

$$\mu : \mathbf{X} \times \mathbf{Y} \rightarrow \mathbf{I}. \quad (1.1)$$

Hence, a fuzzy region is a field where each point (x, y) is assigned a value $\mu(x, y)$ in the range $[0, 1]$ denoting the degree of membership the point (x, y) has in the fuzzy region, or the degree of which a point is considered to be ‘inside’ the region. A *temporal fuzzy region* can similarly be defined as a function

$$\mu : \mathbf{X} \times \mathbf{Y} \times \mathbf{T} \rightarrow \mathbf{I}. \quad (1.2)$$

The main topic of the article is to provide some basic definitions associated with such fuzzy regions.

The article is written together with professor Bjørke, and presented at the ScanGIS’99 conference [BR99]. Note that the article uses a different notation for fuzzy sets than this section and Appendix A do.

1.3.8 How to Read this Thesis

In principle, the articles can be read independently. However, it is recommended to read Chapter 3 before Chapter 4 and 6, since the history graph notation have been used or extended in these chapters. Moreover, it is also recommended to read Chapter 4 before Chapter 5 and 8.

Some of the chapters, in particular Chapter 4, contain some mathematical terminology that the reader might be unfamiliar with. Therefore, the most central definitions, in particular those that are related to the set-theory, have been given in Appendix A.

CONCEPTUAL MODELLING IN SPATIOTEMPORAL INFORMATION SYSTEM DESIGN

2.1 Introduction

Recent research has identified the need to handle historical information in geographical information systems (GIS). The various aspects of temporal GIS and spatiotemporal models have therefore been an active research field since the late 1980s, see [ASS94], [Lan92] and the references therein.

Historically, research in GIS has focused on applicative issues of digital cartography such as how to represent and manipulate spatial data structures in computers. The traditional representation schemes for geographical information system have utilized cartographic primitives such as points, lines, and areas. However, modern computers are capable of representing more information and knowledge about the real world than the paper map model is able to convey, such as the temporal perspective of spatial information.

In order to acquire and communicate the phenomena in the real world, it is necessary to describe these phenomena at the *conceptual level*. Within the field of information systems engineering, an abundance of conceptual modelling languages to describe various aspects of the real world have existed for a long time. But, 'Geographical information systems are often built without due considerations to this discipline . . . ' [HT96].

Wand and Weber [WW89] provide the following definition of an information system:

An *information system* is a human-created representation of a real world system as perceived by somebody, built to deal with information processing functions in organizations.

A *spatiotemporal information system* (STIS) is defined here as an information system where the spatial location and the temporal history of the real world system is of interest to the organizations.

Early research on spatiotemporal models for GIS has focused on the development of computer models that are based on simplified concepts such as those that only timestamp records. These so-called *change-based* approaches have drawbacks, such as the lack of ability to model continuously changing objects [AB90]. One example of a change-based model has been described by Worboys [Wor94a], where spatiotemporal objects have spatial and (bi-)temporal extent.

However, to create models that to a larger extent capture the semantics of the real world, a better understanding of these aspects is required. In the literature, we have seen important work on creating conceptual frameworks for modelling spatiotemporal phenomena ([LC88] [Peu94] [CT95] [SPZ98a] [PSZ⁺98]). This article describes work done on the modelling of various temporal (and spatial) aspects of a real world system, in the sequel referred to as the *universe of discourse* (UoD), using standard and specialized conceptual modelling languages already described in the literature.

The remainder of this paper is organized as follows: The next section briefly reviews some issues of conceptual modelling, and the following sections discuss the most important models with respect to the design of an STIS. The paper closes with some concluding remarks.

2.2 An Introduction to Conceptual Modelling

Computers and computer languages are generally abstract. Larger systems are compiled from thousands of statements in written source code and it is virtually impossible for even a skilled programmer to get a general view of a large program without any visual support in the form of figures and diagrams. During the development of computer systems it is important that all participants understand the problem domain. In general, software designers and software users represent different level of knowledge about programming, and communication between them may easily be distorted due to misunderstandings and lack of insight. The key to

achieve a successful communication among participants is therefore to make them share relevant conceptual knowledge about the domain of discourse. This can usually be achieved by developing so-called *conceptual models* [SK93].

A conceptual model is usually in a diagrammatic form ('boxology') with a grammar which consists of boxes and links between them. Sølvsberg and Krogstie provide the following definition of a conceptual model [SK96]:

A conceptual model is the phenomenon of a domain at some level of approximation externalized in a semi-formal or formal language.

In information systems engineering, a conceptual model serves as a tool for sense-making, as a vehicle for communication and as documentation and basis for design and implementation. However, conceptual modelling techniques are not only useful in the design and development of computer data structures, they have also proven to be a valuable methodology in the acquisition of knowledge of real world phenomena.

Interestingly, cartographic maps and conceptual models have much in common. A map can be defined as a selective, symbolic, generalized image of an object presented in a given scale [Bjø95]. Comparing this definition of a map with the definition of a conceptual model given above, one could conclude that a map is a conceptual model. A map represents a phenomenon of a domain (a part of the real world seen from above), it represents some level of approximation (it is selective and generalized), and it is presented in a formal language (a symbolic image where the symbols are defined through some legend). This may explain why the paper map model has been such a popular basis for spatial computer models in GIS.

2.2.1 *A Brief Review of Conceptual Modelling Perspectives*

Throughout the history of computers and computer systems development, an abundance of different conceptual modelling languages has been presented. Two well-known modelling languages are the entity-relationship diagrams and data-flow diagrams. In general, modelling languages can be divided into classes according to the *structural principle* or the *perspective* of the language. In [SK96], the following seven perspectives are described:

The structural perspective: The focus of the structural perspective is on data and data modelling. The main components of structural models are entities, re-

relationships, attributes and constraints on relationships (i.e. cardinalities). It was the development of the *entity-relationship* language of Chen [Che76] that represented the breakthrough of this modelling perspective.

The functional perspective: The functional perspective focuses on *processes* rather than on objects and physical entities. The best known conceptual modelling language with a process perspective is the *data flow diagram* (DFD) which describes the UoD in terms of external entities, processes, data stores and (data-) flows between these.

The behavioural perspective: The basic concepts of the behavioural perspective are *states* and *transitions* that transform the system from one state to another. One of the problems with this perspective is that larger and complex systems quickly become unmanageable with an almost infinitely number of possible states. To overcome this, some languages such as Statecharts add hierarchical abstraction mechanisms in the form of AND and XOR decompositions [Har87].

The rule perspective: The main application of the rule perspective is in knowledge systems and artificial intelligence (AI). In general, a rule has the form:

$$\text{if}\langle\text{condition}\rangle\text{then}\langle\text{consequence}\rangle.$$

Rules are both utilized to describe knowledge about the real world e.g. in knowledge databases, and to express constraints on other conceptual models, e.g. on an ER-model. One drawback with rules is that conditions are expected to be either true or false, while in cartography as in many other applications, natural conditions often seem to have a fuzzy nature.

The object-oriented perspective: The object perspective has basically emerged as a result of the need to support object-oriented programming languages like SmallTalk, C++ and Eiffel. However, object-oriented analysis and design have truly become a branch on its own, applying the same concepts that were introduced in the object-oriented programming languages (i.e. the object-oriented paradigm).

The communication perspective: The communication perspective is based on the assumption of language/action theory developed by Austin and Searle

called the *speech act theory* ([Aus62] [Sea69] [Sea79]). A few modelling languages exist such as the *action workflow* diagrams [MWFF92].

The actor role perspective: The actor and role perspective is based on ideas developed during work on object-oriented programming languages and intelligent agents in AI. Basic constructs of this perspective are actors, roles and agents.

At first glance, it is quite evident that several of these perspectives are useful in the development of an STIS. Behavioural models may help us understand how objects change over time and functional models may possibly support us in that process. Structural models on the other hand normally represent the real world in a *static* fashion, although extensions to the ER-language that support changes over time — such as the ERT-model [MSW92][TLW91] — have been developed. The object-oriented approach is closely related to the structural approach since it describes relationships between objects, but since the approach incorporates the functionality of the objects, it is also possible to draw relations to the functional approaches. Moreover, object-oriented models may further be supported by *Objectcharts* [CHB92] which is an object-oriented adaption of Statecharts [Har87].

The rule perspective is also of great value in the development of temporal information systems in general. Rules that are coupled with business policy may change over time, and historical data should be viewed in context with the current policy at the time in question. The ability to express policy in the form of explicit rules is therefore critical.

2.2.2 Information, Data and Model Domains

The terms *data* and *information* are often used interchangeably in the literature. However, a clear distinction between those two concepts should be maintained. *Data* should be considered as a collection of symbols represented in computer-readable form. Data exists in form of bits and bytes in computer files, whereas *information* involves some kind of human interpretation, e.g. a coastline may be represented as a stream of points connected with straight line segments. This piece of data conveys some information that a user may interpret as fjords, bays and peninsulas, when presented in a graphical format. In other words, information is associated with a higher perception level, whereas data is associated with low level computer representation.

Along this line of perception levels, three domain levels of information system modelling have been identified [SK93]. These are:

The subject domain: concerns itself with information of the real world. Focus is on physical entities or abstract concepts such as persons, parcels, roads or legislations.

The interaction domain: Concerns itself with the way information in the information system is to be presented and perceived by the system users.

The implementation domain: Concerns itself with the low level implementation of information systems. Focus is on data, communication protocols, data access and algorithms.

A top-down approach to information systems modelling begins with the analysis phase and the study of the real world with the aim of creating a subject domain model. Then the analysis phase moves on with the user interaction modelling, and ends up with the design of the implemented system. In the GIS literature, we have seen the opposite, i.e. a bottom-up approach. First the data model has been designed, and then the user has to fit the real world into the confines of this model. The same trend can be seen in the suggested spatiotemporal data modes such as the space-time composite vector model [LC88] or the event oriented spatiotemporal model [PD95]. In general terms, focus has been on the data rather than on the information.

2.3 *Structural Modelling and the Time Dimension*

2.3.1 *Entity Relationship Models*

The Entity Relationship model, or in short the ER-model, developed by Chen [Che76] was not the first language for semantical data modelling, but it certainly became the most popular one. The main reasons for its popularity is the simple diagrammatic representation and the easy transition to tables of relational databases. Although the intension of the ER-modelling language was to describe the structure of (relational) databases, the ER-language is also appropriate for modelling general knowledge about the real world (i.e. not only the part to be stored in the database).

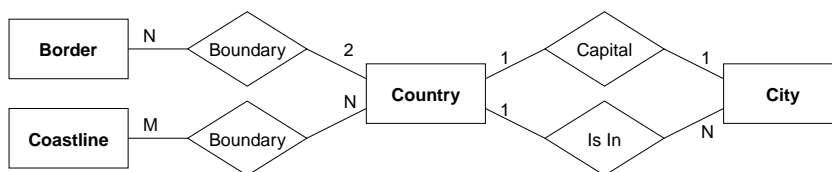


Figure 2.1: A sample ER model describing cities and countries of the world

There are two basic constructs of the ER-model: *entities* and *relationships*. Chen defined an entity as ‘a thing that can be distinctly identified’ and a relationship as ‘an association among entities’. This wide definition is a strength of the ER-model. Each entity is characterized by a set of *attributes* which is common to all entities of the same type.

Consider a temporal database of countries and cities of the world. In this model countries are bounded by borders and coastlines. A coastline may also bound small islands, hence a country may be associated to more than one coastline whereas one coastline may bound several countries. Borders separate two countries, and each country has a capital and a number of other cities. Figure 2.1 shows an ER-diagram of this schema.

However, for modelling concepts in the real world, the ER-language has some shortcomings. One problem is the lack of support of attributes in the original language. It is therefore common to see the the EAR-model (entity-attribute-relationship) in the literature. Other extensions, such as the EER model (extended entity-relationship) also includes concepts such as sub-typing and association; concepts that today are known from of the object-oriented model [TYF86].

2.3.2 The ERT language

A natural consequence of the increasing research on temporal databases, is the emergence of accompanying conceptual modelling languages. Since much of the research on temporal databases has been focused on extending the relational model, it is not surprising that most conceptual modelling languages supporting time are extensions to the ER or the EER-model. One such language is the *ERT* modelling language (Entity-Relationship with Time) [MSW92][TLW91]. In this language, which is based on the EER-model, support for temporal concepts has been applied by the use of *time-stamping* of entities and relationships.

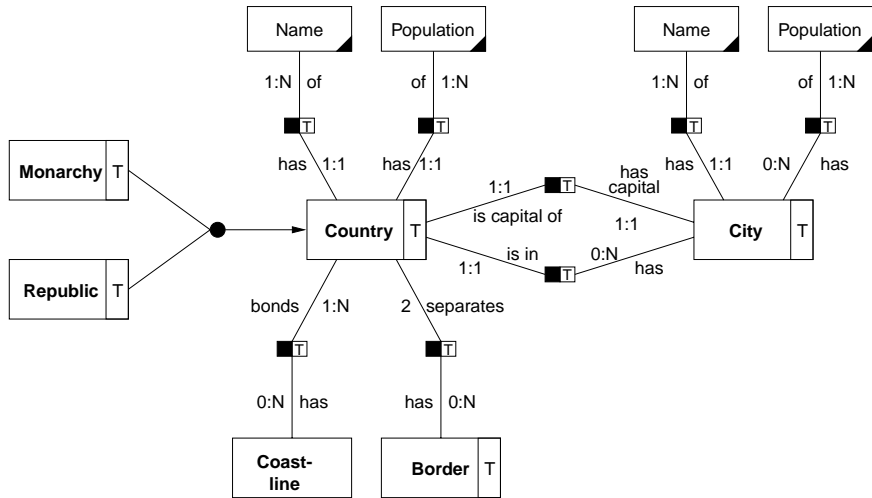


Figure 2.2: An example of an ERT model showing a temporal map over Europe

The basic constructs of this language are the *entity class* (denoted by a rectangle) which denotes a set of objects which share the same set of attributes, the *value class* (denoted by a rectangle with a black corner) which is used to describe an entity's attributes, and the *relationship* (denoted by a line with small black square) which describes the associations between an entity and a value class or another entity. Furthermore, the inheritance extension is represented by a relationship with a circular join. If the circle is solid, then the sub-classes are disjoint or total; if open, then the sub-classes are overlapping or partial.

To implement the temporal dimension in the ERT diagrams, entity classes and relationships may be either **T**- or **H**-marked. If an entity class is **T**-marked it means that the entity only exists at certain times (or ticks) in our UoD, meaning that the entity is undergoing *temporal variation*. If a relationship is **T**-marked, it means that the relationship between the two entities it involves exists for only a subset of the time (number of ticks) for which both the entities it associates exist. If a relationship between two entities of which at least one is **T**-marked is not **T**-marked, it means that the relationship exists as long as both entities co-exist in our UoD.

The **H**-mark is used to indicate that a relationship has a *historical perspective*.

This means that a relationship involvement may exist between two entities that do not co-exist in time. For example, we may say that one *person* has a grandparent that is another *person*, but the two persons did not co-exist in our UoD if the grandparent died before the grandchild was born. However, in the grandparent example, we might want to say that the grandparent is related to its grandchild from the time that the grandchild begins to exist. For this purpose, the **TH**-mark may be used.

Figure 2.2 shows an improved model of the temporal map over Europe using the ERT-language. In this model, we have distinguished between two types of countries: monarchies and republics. For simplicity, we assume that all countries are either monarchies or republics, hence the generalization link is a solid circle. Furthermore, we have also added names and population of cities and countries as attributes and added the **T**-marks wherever appropriate. Because the ERT model both supports temporal aspects and sub-typing it is particularly interesting in the design of an object-oriented system. Furthermore, the notation of the ERT-language allows us to read out the relationships, as for example the relationship between countries and its capital: A country *has* one capital, while a city can be the *capital of* one country.

2.4 Modelling Processes

There are several reasons why it is desirable to model processes in the real world. One is because they involve human interaction, and that these processes need to be automated (e.g. monitoring and management of ships in a harbour), or because the process is to be simulated in the computer (e.g. the melting and accumulation of glaciers).

This section presents two modelling languages that belong to the functional perspective, viz. *data flow diagrams* and *demos activity diagrams*.

2.4.1 Data Flow Diagrams

There are two commonly used notations for data flow models. One was proposed by deMarco [DeM78] and the other was proposed by Gane and Sarson [GS78]. The two languages contain exactly the same concepts, but provide different symbols for them. For practical reasons, we use the notation of Gane/Sarson, which is shown in Figure 2.3.



Figure 2.3: The functional perspective: Symbols of the DFD language

Data flow diagrams have many applications in STIS. In contrast to the ER-diagrams they do not show how things are, but how things are done or how things happen. This is valuable information in STIS.

Consider that a building department is maintaining a database for all buildings in a municipality. In order to set up new buildings, pull down or renovate existing buildings, or to change the use of buildings, the proprietor has to apply and get permission from the department to do so. The application is received and verified for conformance with the area development plan, and notifications are sent out to the neighbours for reactions. Upon approval (or rejection) of the application, the necessary documents are sent back to the proprietor. When a new building is built or an existing one is changed, it is inspected to verify that it conforms with the original application and the final position of the house is surveyed. In some cases, houses get damaged (partly or totally) due to fire or other natural causes (e.g. landslides and avalanches). Notification about this is received from the insurance companies. This way, the building department will have a complete inventory of all buildings in their municipality at all times.

Figure 2.4 shows a data flow diagram of the system described above. In this diagram the external entities are the proprietors, their neighbours and the insurance companies. The main data store is the building database, but also a cadastral database and an area development plan is needed. The latter two databases may be maintained by other departments. The processes shown in the figure are described as follows:

P1: Receive application: An application is received from a proprietor. The application may be either for setting up a new building, to extend or restore an existing one, or to change the use of a building (e.g. to change from private residence to an office building). The application is checked whether all necessary documentation has been included, and if it is OK, it is forwarded for

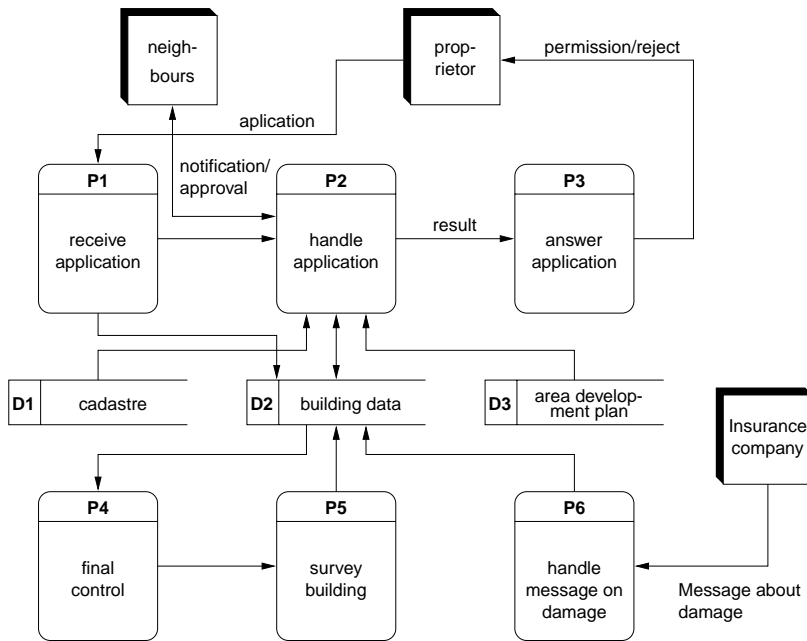


Figure 2.4: A data flow diagram of the activities in a municipal building department

processing.

P2: Handle application: In this process, the neighbours to the building site are determined from the cadastral database, and a notification is sent out to them. If the neighbours have objections, they will have a deadline to make these. If there are no objections and the application is in conformance with the area development plan, it is accepted, and forwarded for final notification to the proprietor. If a new building or a wing is to be set up, preliminary coordinates for the position of the house is put to the building database.

P3: Answer application: The application is accepted, and the necessary certificates and permissions are issued to the proprietor.

P4: Final control: When the building is finished, it is inspected in order to verify that the building, restoration or change in use is according to the application, and the original intentions.

P5: Survey Building: If a new building or wing has been set up, the new building is surveyed and its exact location is determined. The results are stored in the building database.

P6: Handle message on damage: Sometimes, houses are damaged, partly or totally, due to fire, landslide, avalanche, or other reasons. Notification about this is received from the respective insurance companies, and the building database is updated accordingly.

In the diagram, it is easy to see which processes contribute to the updating of the databases, and which processes only need to read information from databases. By further specification, it is also possible to describe exactly what information is updated by these processes.

2.4.2 *Demos Activity Diagrams*

Another application of the functional perspective is where real world processes are to be simulated in computers. In fact, computer simulation is itself an active research field, and designers of STIS may have a lot to learn from this field. An early simulation language, Simula [DN66], has been extended with a package called

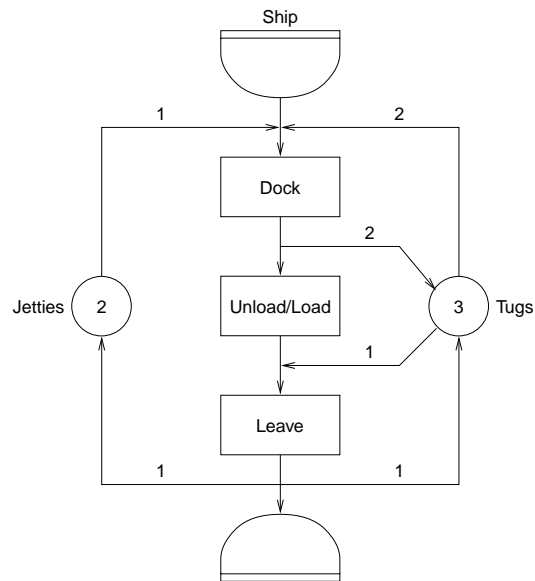


Figure 2.5: Demos activity diagram for a harbour

Demos [Bir79] to ease the implementation of simulation programs. Demos programs, can be visualized using so-called *Demos activity diagrams*, and the original notation has been extended a number of times through [Bir79] and [HR85].

An activity digram shows how an object enters the UoD, goes through different activities, acquiring resources, cooperates with other objects, interrupts other activities or is interrupted by other activities, before it leaves the UoD. In addition to activities, activity diagrams also incorporate entities such as resource objects, bin objects, conditionals and wait activities.

Figure 2.5 shows a sample activity diagram for a harbour which is managed by the harbour administration by the help of an STIS. The harbour administration has two jetties and three tug boats which are modelled as resources (indicated by circles) . Two tugs are required to dock a ship whereas one is sufficient when a ship is leaving. When a ship arrives, i.e. enters the UoD (indicated by a lower half circle) it must acquire two tugs and one jetty. If these resources are not available, the ships have to wait until they become available, and they are served on a first come, first served basis. The processes of the system, such as the docking process,

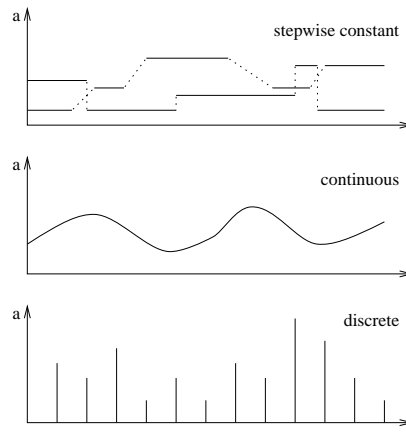


Figure 2.6: Temporal behaviour of an attribute

unloading/loading process and the undocking process are indicated by rectangles, and these processes are considered to take some arbitrary amount of time. When the docking is complete, the two tugs are released, and the unloading and loading process can begin. When the loading is complete, one tug is acquired before the ship can leave. The ship is towed to sea, upon which the jetty and the tug is released, and the ship can leave the UoD (indicated by a upper half circle).

The example given above, only illustrates a subsets of the concepts of the demos activity diagrams. Nevertheless, the demos activity diagrams also have a lot in common with the behavioural perspective; the example above could to some extent also be modelled using Petri-nets.

2.5 Modelling Behaviour

Understanding behaviour is one of the most fundamental issues of STIS engineering. Most spatiotemporal models so far are extensions of existing data models, and a common solution is to simply timestamp the data when it is updated. Such systems, are then only capable of representing changes as sudden events. However, we know that many changes in the real world have duration. In general, features in the real world exhibit a wide range of temporal behaviour. In general, three basic types of behaviour have been identified [SS93] [MP93]. These are as illustrated in

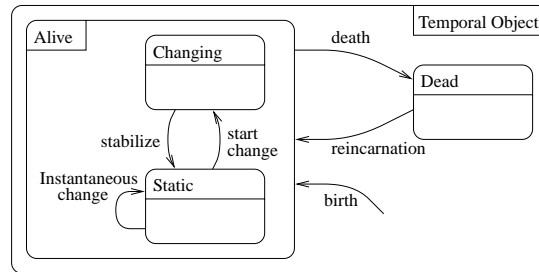


Figure 2.7: Generic behaviour of temporal objects using the Statechart notation

Figure 2.6:

Stepwise constant: A feature of this type is considered to be static and changed by events. These events may be instantaneous, such as the division of a parcel, or they may have duration such as the building of a road, or the change in position of a ship that is sailing from one harbour to another.

Continuously changing: Features of this type are always considered to be in a changing state. The population of the country or city, or the expansion and retreat of glaciers are examples of such behaviour

Discrete values: Features of this type are considered to be associated with specific times or time intervals. The amount of precipitation per day, and the gross domestic product of a country are examples of such behaviour

To model temporal behaviour, it seems natural to use a language such as Statecharts [Har87] or Objectcharts [CHB92]. However, these languages are similar to finite state machines which are based on the idea that the system is always in one state and that transitions between each state are instantaneous. Although, many objects in the real world exhibit such behaviour, it would be of great value if we could model gradually changing objects as well. However, if we introduce the *state of change* as a distinct state, we may obtain a generic Statechart model (i.e. a meta-model) for spatiotemporal objects as shown in Figure 2.7. According to this model, an object is either *alive* or it is *dead*. An object that is alive, may die and enter the state of being *dead* and later become alive again by a *reincarnation* event. If the object is alive, it may either be in a static state, or in a state of continuous

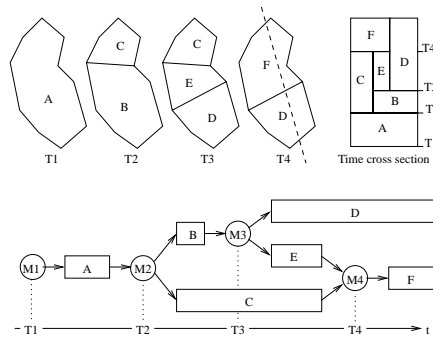


Figure 2.8: The story of a land area (above) shown in the history graph notation (below)

change. An object that is in a changing state may *stabilize* and enter the state of being static, and then later it may start to change again. An object that is in a static state may change instantaneously and continue to be in the static state.

It was the idea of this model that led to the definition of the *history graph* notation [Ren96]. In this language, which in some way is similar to Petri-nets (which also is a modelling language that belongs to the behavioural perspective, [Pet62]), the objects of a data set may be described through a series of consecutive states (i.e. static states) and changes (i.e. changing states). The states are denoted by a square rectangle, while the changes are denoted with boxes with circular ends. Both the states and changes are associated with a time interval, and the boxes are stretched to mimic the time interval they are associated with. Objects that change suddenly would then be described by transitions with zero duration (i.e. events), while objects that change continuously would be described by version with zero duration (i.e. snapshots) describing intermediate states. An object that is dead, is denoted with a rectangular box with a dashed outline. Figure 2.8 shows a sample story where a region is split and merged. Since the changes in this story are considered as sudden events, the transitions are shown as circles. A similar language has been introduced in [HE97], but it does not consider changes as distinct entities.

Studying Figure 2.8, one can identify a number of distinct change types, such as the splitting and merging of objects. As illustrated in Figure 2.9, a total of seven different types of changes were identified. These are as follows (a refinement of these change types can be found in [HE97] and [CT95]):

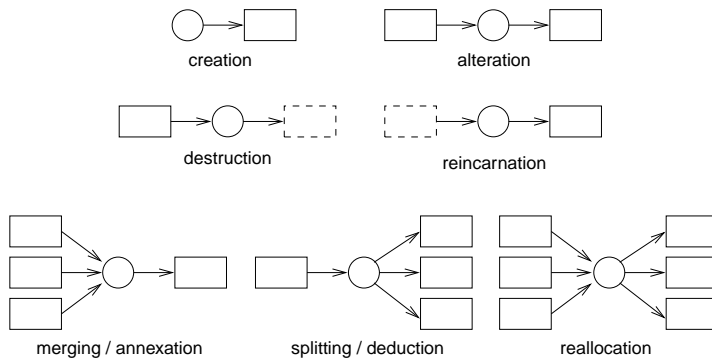


Figure 2.9: Seven basic types of changes (Shown in history graph notation)

Creation: An object is created.

Alteration: An object is changed or modified.

Destruction: An object is destroyed or removed.

Reincarnation: An object that previously has been destroyed or removed is reintroduced, possibly with a new state and location.

Split/Deduction: An object is subdivided in two or more new objects or one or more objects is deducted from an existing object.

Merge/Annexation: Two or more objects are joined together to form a new object or one or more objects are ‘swallowed’ into another object.

Reallocation: Two or more objects are merged together and two or more different objects result from the change.

Although the history graph notation helps us understand temporal behaviour in particular cases, it does not allow us to describe the general behaviour of certain types of features. For example, changes to a country can be one of the following types:

1. Two countries may merge to form a new country, e.g. East and West Germany merge to form Germany.

2. One country splits to form two countries, e.g. Czechoslovakia splits to form the Czech Republic and Slovakia.
3. A region in one country gains its independence and forms a new country, e.g. Estonia, Latvia and Lithuania withdraw from the Soviet Union.
4. A country is annexed into another country, e.g. Iraq invades Kuwait.
5. A border between two countries is adjusted according to an agreement between the two countries.
6. The population is changed. Apart from the continuous change of population by natural birth and mortality, all of the events above will have an effect on the population of the involved countries.
7. When changes occur in territories, some cities will change country, and the involved countries will have a jump in population.
8. The capital of a country is moved to another city, e.g. Germany moves the capital from Bonn to Berlin.
9. A country changes name, e.g. the Soviet Union becomes Russia.

On the basis of these changes, one can make transition specifications. Formally in a Statechart, a transition specification comprises the initial and the final state of the transition and the service name for the transition, together with a precondition and a postcondition.

A problem with the Statechart is that it poorly expresses the interaction between objects, e.g. how can we comprehensively describe a deduction. A deduction generally means the creation of one object and the alteration of another. Obviously, such issues have not been addressed, although the history graph notation to a large extent can assist in such specifications.

2.6 Object-Oriented models

A popular approach in GIS modelling in general but also in spatiotemporal modelling is the *object-oriented* approach, and a number of data models have been presented [RMD94] [Ham95] [Wor94b]. However, the object-oriented model has

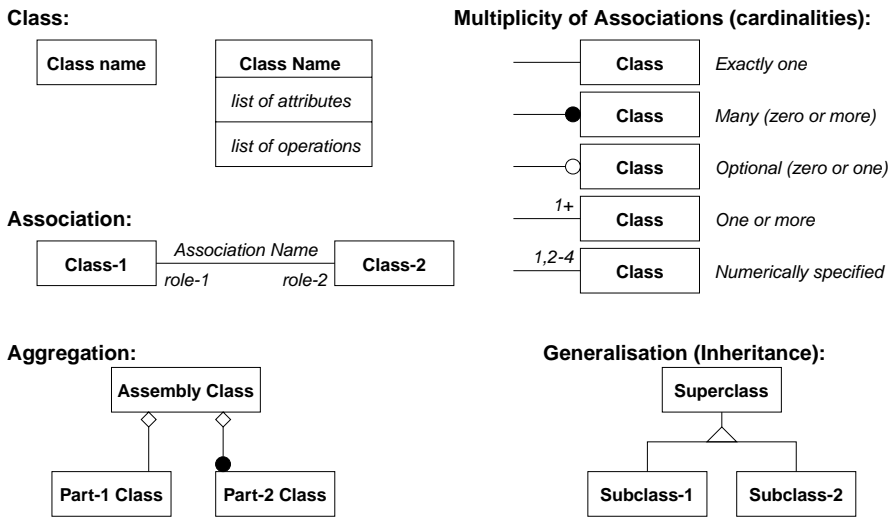


Figure 2.10: The most common symbols in the OMT-language

been used in GIS development as a mere wrapper around the vector model, rather than a fundamental approach according to which geographical knowledge could be modelled. Apart from the advantages of the object-oriented model in traditional GIS, such as increased modelling power. Käfer lists four main advantages of an object-oriented model in a temporal database [KRS90]:

1. The complete history of an entity can be encapsulated into one single object.
2. Since the complete history of an entity can be represented as a single object, queries become less complicated, because they do not consider the dispersion of the entity over many tuples.
3. Since complex object queries are executed efficiently, the corresponding temporal data should be handled efficiently as well.
4. Handling of temporal and non-temporal data can be done in a uniform way.

The object-oriented approach provides concepts as object class, association, aggregation, and generalization. Several object-oriented modelling languages exist [Boo96] [Mar93] [EHS93], but the language of the object modelling technique

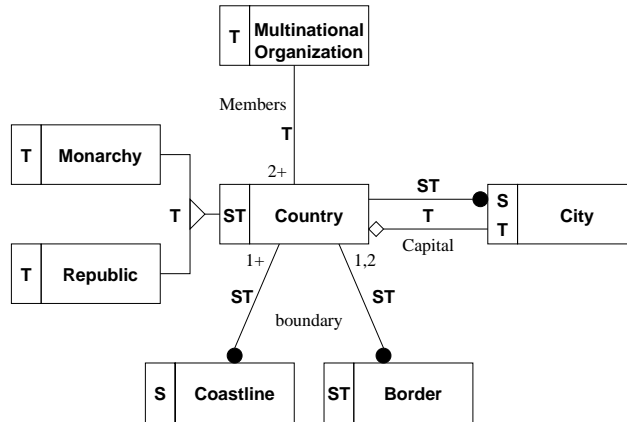


Figure 2.11: An Object Oriented model of a temporal map of Europe with temporal extensions

(OMT) [RBP⁺91] seems to be the most popular one. The most common symbols of this language is shown in Figure 2.10.

Some research has been seen where the object-oriented model has been used to design a temporal data schema, without embedding temporal constructs into the modelling language itself. One model is the multimodel and metamap schemas by [Mis93] [BS93], where spatial and non-spatial information can be accessed via parameters such as time.

Since the early 1990s we have seen an increasing activity in the design and construction of temporal object databases. Some ongoing research deals with incorporating concepts from these models into an object-oriented modelling language. Of particular interest is MADS (for Modelling of Application Data with Spatiotemporal features) which is described in [SPZ98a] and [PSZ⁺98]. MADS is so far the most comprehensive modelling language for geographical and spatiotemporal information within the object-oriented paradigm. MADS provides the standard object-oriented concepts which can be marked with various icons to indicate certain spatial and temporal properties of these concepts.

A simple ERT-like extension of the OMT can also be found in [Ren99a]. Figure 2.11 shows an improved model of the countries of the world. Here, we have hidden the attribute information, but added a new type of objects, viz. multinational organizations. A multinational organization has a number of member countries,

and over time it may receive new members and existing members may secede from the organization. A country can also be a member of several such organizations. The markings on this figure are defined as follows: a class that is **T**-marked has properties that varies over time. An object class that is **S**-marked is a spatial object and thus has a location. If an object class is **ST** marked, it means that the location of such objects may also vary over time. Hence, an object class may be both **T**-marked and **S**-marked, but not **ST**-marked if it has a static location, but has other properties that vary over time. Subclasses inherit all markers from their superclasses, and are only marked if additional properties that deserves a mark are added to the object class.

A generalization link is **T**-marked if the instance of the subclasses may change in type, e.g. if a monarchy changes to become a republic. An aggregation or association link may be **T**-marked if the link only exists during a part of the time for which both involved objects co-exist. If, on the other hand, a link between two temporal object classes is not **T**-marked it means that the link exists as long as both objects co-exist. If an aggregation or association link is **S**-marked it means that the link is spatially dependent, i.e. is a topological link. Thus, a city can only be related to the country in which it is topologically inside.

Objects may have several properties that vary independently over time. Some properties may vary suddenly, such as the location and name of a country, whereas other properties may vary continuously, such as the population of countries and cities. Additional markers can be added to attributes if they are provided in the model.

2.7 *Rules, Knowledge and Business Policy*

In cartography, rules have mostly been applied to issues related to cartographic generalization. In conceptual modelling on the other hand, rules can be used to express constraints on conceptual schemas (such as cardinality constraints).

Most often, such rules are implemented directly in source code. In information systems, this represents potential problems since business policy may change over time, making current information systems obsolete [MNP⁺91]. Historical data should always be viewed in context with the rules that were current at the time in question. For example, to become a member in the EU, a country has to present a national budget with a deficit that is less than a certain percentage of the gross

domestic product. However, many countries that already are members of the EU, did not meet this requirement, but they still became valid members because the rule did not exist at that the time when they became members.

A possible solution is to explicitly express rules by an *external rule language* (ERL) and store them together with the system. There are two approaches to this problem. One is to store the rules in a temporal database, such that rules can easily be obtained for specific points or periods of time. Another approach is to let the time validity be an inherent part of the rule if the rule is expressed as

$$\mathbf{when}\langle time \rangle \mathbf{if}\langle condition \rangle \mathbf{then}\langle consequence \rangle$$

The rule can then be viewed as describing some logical constraint on the model, which must hold at every moment (or tick) whilst the information system is active [MNP⁺91].

Any system of rules must contain a set of predefined predicates that are understood by the system. In tense logic, four temporal predicates have been proposed [RU71]. These are as follows:

$$\begin{aligned} F(p) & : \text{ it will be that } p, \\ P(p) & : \text{ it has been that } p, \\ G(p) & : \text{ henceforth, always } p, \\ H(p) & : \text{ hereto-forth, always } p, \end{aligned}$$

where p denotes any proposition. Similar constructs have been proposed by Alagic [Ala97] in a temporal constraint language as a high-level, declarative database programming paradigm for object-oriented databases.

In the field of artificial intelligence, rules have played a central role in the representation of knowledge about the real world. McCarthy's *situation calculus* [MH69], has been a major inspiration source for other research in this area. In this theory, a *situation* is defined to hold the complete state of the universe at an instant of time, and the set S is defined to contain all situations including both real and imagined situations. A *fluent* is a function from S into S or into the set of booleans $B = \{true, false\}$. This means that a fluent $at(x, p, s)$ is true if an object x is at the position p in the situation s . Using the tense operators above, we can express causality. In order to express that in a situation s , if a person x is located at position

p and it is raining at position p , then x will become wet, we could write:

$$\forall p \forall x \forall s (\text{raining}(x, s) \wedge \text{at}(x, p, s) \rightarrow F(\text{wet}(p, s'), s))$$

Allen [All84] presents a further development of the situation calculus where events, processes, activities and causality are expressed as predicates occurring over time intervals along with binary operators over temporal intervals [All83]. Consequently, if an object x moved from location $p1$ to a location $p2$ over a time interval t , this is expressed as the formula

$$\text{OCCUR}(\text{CHANGE_POS}(x, p1, p2), t)$$

2.8 *Concluding Remarks*

This article has presented a selection of conceptual modelling approaches and demonstrated some of their applications in the development of an STIS. This has made it possible to describe complex aspects of real world systems. A simple basic model has been presented, shown in Figure 2.7, that recognizes three basic states of objects. On top of this underlying meta-model, there is a need to create application-specific domain models which precisely describe a domain in the real world. This model can be created by using ERT-models or object-oriented models together with data flow diagrams, behaviour models and rule-based constraints.

By using conceptual modelling languages, it is possible to describe and uncover the characteristics of real world systems in a way that is understandable even to non-experts. Thus, a broader audience can participate in the development of a system and the users will have a system that more closely represents their own concepts of reality.

CONCEPTUAL MODELLING OF SPATIOTEMPORAL DATA USING GRAPH STRUCTURES

3.1 *Introduction*

The need for representing historical information in databases has led to the idea of temporal databases, databases where time is an explicit dimension. Recent research in temporal GIS has shown that modelling real world phenomena in space and time is a non-trivial task, and several different data models have been presented. To be able to construct efficient temporal databases, it is critical to understand temporal phenomena in the real world. For this purpose, conceptual models are necessary since they provide a vehicle for communication and a basis for design and implementation [SK96].

The model introduced in this article has much in common with state-transition diagrams and Petri-nets ([Pet62]). The focal point of this model is the dichotomy between states and events, and the durational aspects of these concepts.

3.1.1 *Change patterns*

Changes on spatiotemporal objects may occur at any level, on the attribute level as well as the geometrical and topological level. Most commonly, changes made by humans are considered as events, e.g. a new road is opened, a new building is built, or parcel has been divided in two. It is common that such events are considered as instantaneous occurrences, but many of the changes in the real world are results of continuous processes such as glacial fluctuations, expansion of deserts

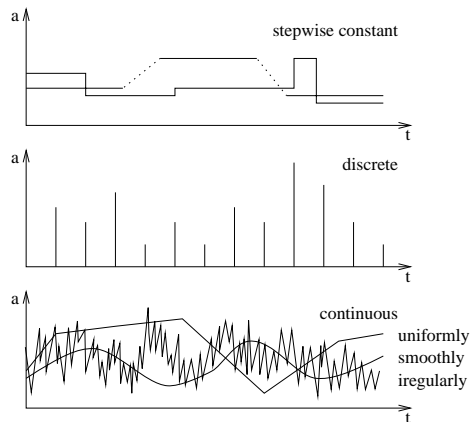


Figure 3.1: Types of changes in an attribute

and erosion of rivers. Although most types of changes are either of a sudden nature or are results of continuous processes, some types of changes also fall in a category between the two.

Entities can be divided into several categories according to their temporal behaviour or change patterns. As illustrated in Figure 3.1, three main types of behaviour can be identified: stepwise constant, discrete and continuous [SS93]. *Stable values* are exposed to instantaneous events and have stepwise constant values. *Discrete values* are collected on a regular or irregular time intervals and contains values that have been accumulated over this interval (e.g. daily rain fall). *Continuously changing values* can be divided into three sub-categories: *uniformly*, *smoothly* and *irregularly* changing values [MP93]. Additionally, some entities are static and never change while other entities may be measured or depend on time itself [Lan92]. An example of the latter type may be a scheduled opening date for a new road, or the opening hours of a ferry.

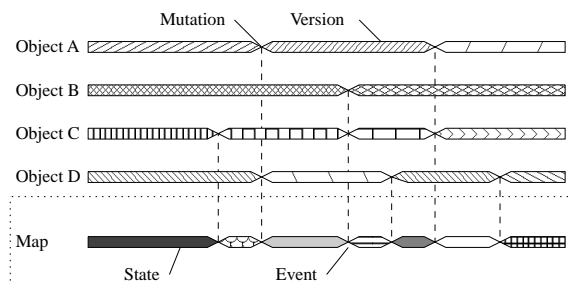


Figure 3.2: The relationships between events and mutations, map states and object versions in a spatiotemporal system (from Langran 1992)

3.2 Events and States

3.2.1 The topology of time

Basically, modelling a temporal system is about managing objects and their changes. As stated, the term ‘event’ is usually considered to denote some instantaneous occurrence that transform objects from one state to another. However, one event may transform several objects and their subcomponents into new states. As illustrated in Figure 3.2, Langran therefore suggested to distinguish between *map state* and *event* as terms pertinent to an overall database- or map-level, and *versions* and *mutations* as terms pertinent to the object-level. This relationship between versions or states and mutations or events describe a temporal topology where temporal neighbours are predecessors and successors of each other. Each object forms a line starting with the object’s birth, going through its life cycle which consists of a series of contiguous versions separated by mutations and eventually ends in the death of the object [LC88].

This dichotomy between events and states (or versions and mutations) is fundamental in temporal data modelling. Consequently, there are two main approaches to data modelling, the *state-based* approach where the history of an object is viewed as a sequence of states or versions, and the *event-based* approach where the history of an object is viewed as a sequence of events or mutations. This article is also concerned about the durational aspects of states and events, and we will show that both states and events can be associated with durations of time. However in this article, we use a different terminology than Langran. We use the

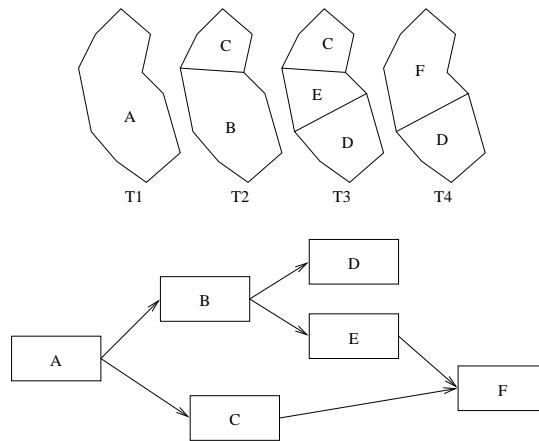


Figure 3.3: The story of an area, and the temporal relationships between objects in the story

term *object-state* and *map-state* (or *database-state*) for Langran's version and state, respectively, and the terms *event* and *change* for Langran's event and mutation, respectively.

3.2.2 Representing The State-Based Approach

Consider the area depicted in the top of Figure 3.3. At time T_1 the area consists of one single object, region A . At time T_2 , A is split into two regions, B and C . Later, at time T_3 , B is split into E and D , and finally at T_4 , C and E are merged into a new region F . As shown in the bottom of Figure 3.3, this indicates a general structure of a spatiotemporal objects where the states of each object are linked together in a directed acyclic graph (DAG).

This graphical view of a story focuses on the states of each object and will be referred to as the *state-based* approach. The strength of this approach is that it is easy to obtain time-slices at certain times as each object state may be stamped with a time interval. The disadvantage of the approach is that it is not possible to obtain direct information of *what* happened or *why* it happened. In other words, the changes can only be obtained in terms of their effects rather than as explicit information. This means that the state-based approach is strong on queries like

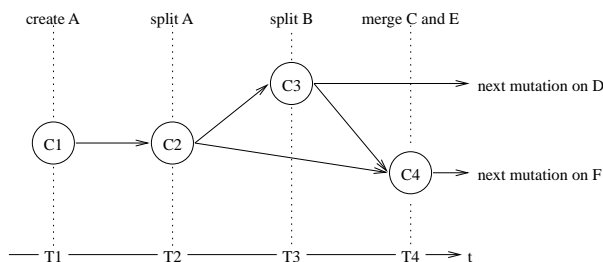


Figure 3.4: The relation between changes in the event-based approach

‘*What was the state of . . .*’ and weak on queries like ‘*What happened to . . .*’

3.2.3 The Event-Based Approach

An alternative to the state-based approach is the *event-based* approach, which represents the data in terms of *events* rather than states. With this approach, the state of an object can be obtained by tracing through the events associated to that object. Langran outlines such an approach called the *amendment vector* approach where a base state (or a final state) is overlaid with amendment maps, which represent the events in the database [LC88]. Another similar model (ESTDM) is implemented by Peuquet and Duan [PD95]. In this raster model each event is represented by a compactly stored change image.

Looking back at the story outlined in Figure 3.3, an *event-dependency graph* may be created. The graph that would look like the one in Figure 3.4 is a bit harder to interpret, but the changes labelled *C1* through *C4* are as follows:

- C1*: *A* created.
- C2*: *A* is split into *B* and *C*.
- C3*: *B* is split into *D* and *E*.
- C4*: *C* and *E* merged into *F*.

With this approach, the states of each object can be obtained for any time. It cannot be obtained directly, but computed from an initial state and summing up all the events applied to the object. The advantage of this approach is that explicit information about *what* happened to the objects is stored in the database. Data models based on this approach should be capable of tracing events both forwards and backwards. Additionally a current state database should be given to provide a

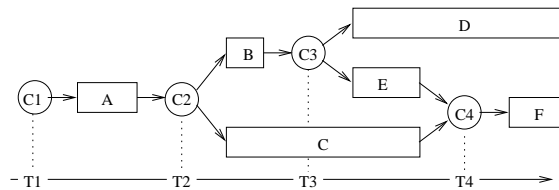


Figure 3.5: The different objects and the events that relates them

complete map of the most current situation and to provide shorter paths to recent and usually more frequently accessed states.

3.2.4 Combining Events and States into One Graph

Although the event-based approach and the state-based approach both have their advantages and disadvantages, the two approaches complement each other. Hence, it is natural to suggest that temporal databases should manage both events and objects in their data sets. Therefore, we will combine object states and changes into one graph as shown in Figure 3.5. This graph clearly illustrates the dichotomy between states (shown as rectangles) and events or changes (shown as circles), and will be called a *history graph*. The primary application of this graph is to describe a limited extent in time and space called a *story*.

As can be seen from Figure 3.5, three types of events can be identified: $C1$ which creates or adds A , $C2$ and $C3$ which split one objects in two, and $C4$ which merges two object into one. Generally, four more types of changes can be identified by studying the number of ‘input’ and ‘output’ states of the event. As illustrated in Figure 3.6, this gives us a total of seven different types of changes:

Creation: A new object is created.

Alteration: An object is modified, either by change in attribute or by change in geometry.

Destruction: An object has been destroyed and does no longer exist in the universe of discourse. All objects that are destroyed are potential items of a reincarnation.

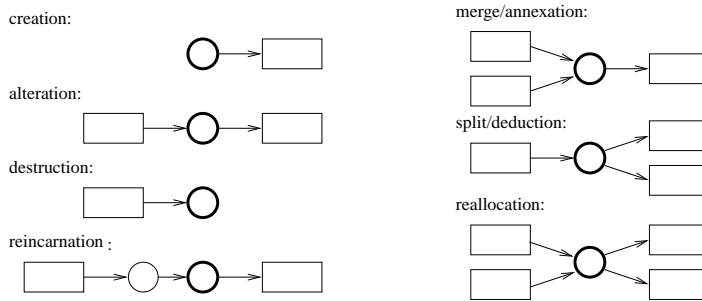


Figure 3.6: The seven basic classes of changes

Reincarnation: An object that has been destroyed, is reincarnated. A reincarnation can only succeed a destruction, and has only one succeeding object state. This is a type of event that can, if omitted, be replaced by a destruction and a creation of an entirely new object. In that case, there is nothing that relates the two objects to each other, although we are speaking of the same real world entity.

Split: An object is subdivided in two or more fragments. One of the succeeding objects may have the same identity as the original object indicating that the other successors were *deducted* from the original object. To indicate this, the edge between the change and the succeeding state may be indicated with a fat line, while normal thin lines indicate creation of new objects with new identities.

Merge: A new object has been constructed from a number of existing objects. The identity of the object may be new, but the new object may also inherit its identity from one of the former objects, indicating that some of the preceding objects are *annexed* into the new object. Again, persistence of identity can be indicated by drawing the edge using a fat line.

Reallocation: A group of objects have been changed and reallocated in a different configuration. Alternatively, it is possible to represent such an event by two consecutive merge and split events. Again, drawing some of the edges using a fat line may indicate persistence of identity.

As can be seen in Figure 3.6, the basic difference between these event types

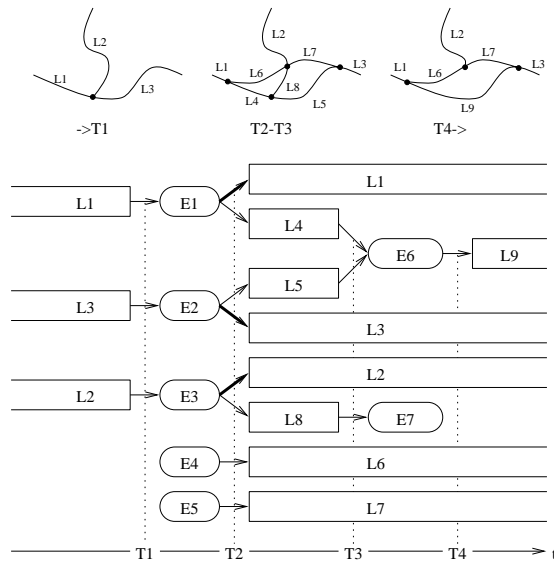


Figure 3.7: Three stages in the history of a road, and the associated history graph

is the number of preceding and succeeding states. While the creation has no preceding state, splitting, destruction, and alteration requires exactly one predecessor. Merging on the other hand requires at least two predecessors while a reincarnation can only precede a destruction.

3.3 Extending the Model

3.3.1 Events do have Duration

The conceptual model that emerged in the preceding section is based on the assumption that events are instantaneous and therefore have no duration. Generally, this is not the situation, e.g. it may take several years to build a new road. Although in many applications the opening date is sufficient as a ‘creation date’, other applications would take advantage of information that indicates construction or planning activity in the period before the opening.

The natural way to accommodate this in our model would be to apply duration to the events as well. In other words, not only the objects are time-stamped, but

also the events. Consider the roads in Figure 3.7. Between $T1$ and $T2$ a new road segment is built including a new crossing of $L2$. The new road is opened at $T2$ and the situation remains constant until $T3$. From $T3$ to $T4$ the segment $L8$ is destroyed merging $L4$ and $L5$ into a single line: $L9$. The history graph of this story is given at the bottom of Figure 3.7. Here, the event of building the new road segment composed of $L6$ and $L7$ is represented as the changes $C1$ through $C5$. The destruction of the small segment between the new and the old road segment, in the figure labelled $L8$, is an event represented by the changes $C6$ and $C7$.

This story is given at the geometric level with line segments as the main entity type. A higher level story based on the concept of whole roads can be presented as well. That is a matter of decomposition of elements and will be described in Section 3.4. However, we have shown that events that have duration can be modelled using history graphs by extending the event-oriented paradigm. We do this by drawing the events as oval-shaped boxes, and the states as rectangular boxes. The shapes are stretched along the horizontal time line to mimic the duration of these entities.

3.3.2 Continuous Processes

So far, we have assumed that changes in the world occur in the form of events that have a finite duration in time. However, there are also changes in the world that are caused by continuous processes. The fluctuations of a glaciers, coastal erosion, air pollution and weather are examples of such processes. The most natural way to model these phenomena is to record states at regular or irregular intervals, and to interpolate between two states in order to find intermediate states.

In fact, such processes can easily be described by the model already outlined in the previous sections. In opposition to the event-based way of thinking where the states span all space in time and the events have zero duration, continuous processes can be described in a diametrically opposite way, by states having zero duration and changes spanning all space in time. However, at this point we need to extend the terminology a little. We suggest using the term *process* instead of events, but consider the term *change* still applicable. Furthermore, we use the term *snapshot* to denote a state that have no (or very short) duration.

For example, consider a melting glacier as illustrated in Figure 3.8. Three snapshots of the glacier(s) are given at times $T1$, $T2$ and $T3$ respectively. We can see that, as the ice melts, the glacier disintegrates into smaller ice patches. The history

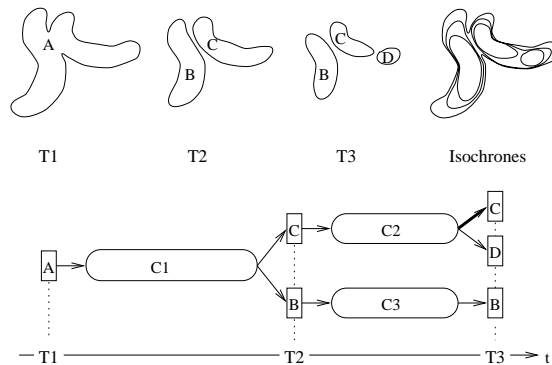


Figure 3.8: Three stages in the melting of a glacier

graph shown in the figure shows the intermediate states, or the snapshots, depicting the situations at the actual times. Although the snapshots have zero duration, they are drawn as narrow rectangles. This is mainly due to graphical reasons, but one may also state that the rectangle representing the snapshot gives a sufficiently accurate description of the state within the time interval indicated by the extent of the rectangle.

There is also another modelling issue worth noticing at this point. Looking back at Figure 3.7, we see that the changes $C1$ through $C5$ are all part of the same event of building a new road, and that each change is associated with the same time interval as the main event. On the other hand, the process in Figure 3.8 includes $C1$ as well as $C2$ and $C3$ and spans the full history of the objects being involved. In other words, each of the changes is associated with a subinterval of the total time interval that the main process is associated to.

Additionally, the story of Figure 3.8 also reveals a number of questions related to topology: When did A split into B and C ? and When did E separate from C ? From the snapshots, we cannot exactly tell, but satisfactory approximations can be computed using interpolation methods.

3.3.3 Sudden Changes in Continuous Processes

Many types of features may change continuously in a smooth and regular manner, but some may also experience abrupt changes of a sudden nature. The glaciers in

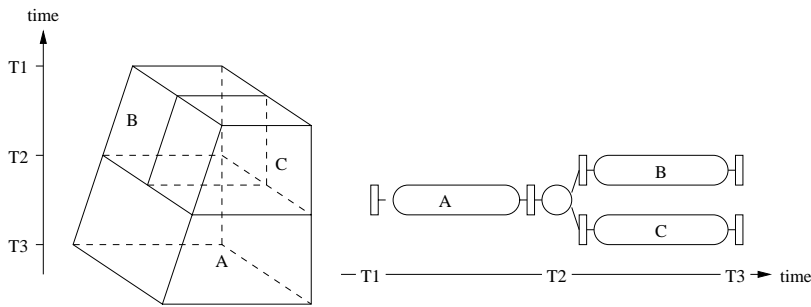


Figure 3.9: An instantaneous topological change in otherwise continuous changing features

the previous section illustrates this where we see a sudden change in topology of the ice patches. Another typical example of this is sudden changes in population. Imagine the population of Russia (or the former Soviet Union). At a large scale, we may assume that the population changes regularly and smoothly, but when e.g. the Baltic counties separated from Russia, Russia experienced an instantaneous jump in its population.

Consider the situation illustrated in Figure 3.9. It shows a parcel situated by a river bank that is continuously shrinking due to erosion by the river. Then at a certain point, the parcel is split in two, and at the time of the split, geodetic surveys was made to determine the coordinates of the new boundary line. At the same time, the river bank was measured to determine the current size of the parcels.

In order to model this using history graphs, we need two sets of states to determine the situation immediately before and immediately after the division of the parcel. These states are in principle associated to the same time, the time when the split occurred. But between these two sets of states, there is an event, also with no duration that represent the splitting of the parcel. Hence, as illustrated in Figure 3.9, such a change is modelled using a snapshot - instantaneous event - snapshot sequence where all three elements are associated with the same time, the time of the sudden change.

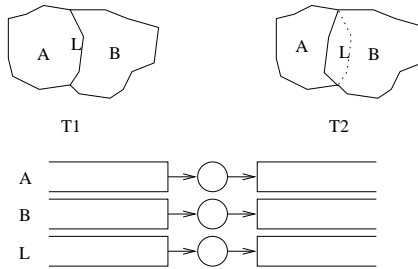


Figure 3.10: Both polygon A and polygon B are changed as a consequence of the event that moved the line between them

3.4 Compound Objects

Generally, there are two types of spatial objects in a spatial database. On the one hand, there are atomic objects such as points and lines, while on the other hand there are compound objects that consists of, or are formed by other objects such as polygons and volumes. So far, we have dealt with objects at the more individual level. In this section, we are going to investigate the implication of topological interrelationships between objects.

Using object-oriented terminology, if two objects are tightly bounded by a part-whole-relationship, it is called *aggregation*, while the term *association* is used for more loosely related objects [RBP⁺91]. In a polygon mesh, a polygon can be seen as an aggregation of the nodes and the lines surrounding the area. A higher level object, such as a complex area, may again be aggregated from several polygons, one (or more) describing the outer bounds, and zero or more others describing holes in the area. Yet, one line can be part of two polygons, and both these polygons are thus topologically associated to each other.

Consider the line L in the story pictured in Figure 3.10. Because the line is altered, the polygons of which the line is part of, is also altered. There are two basic ways to represent this in a history graph. One method is to treat the line and the two polygons as individual objects placing them side by side in the graph as shown in the figure. The other approach would be to try and show the interrelationships between the objects in the same graph. To demonstrate this, we will revisit the example in Section 3.2.2 and have a closer look at the relation between the lines and the patches (i.e. polygons).

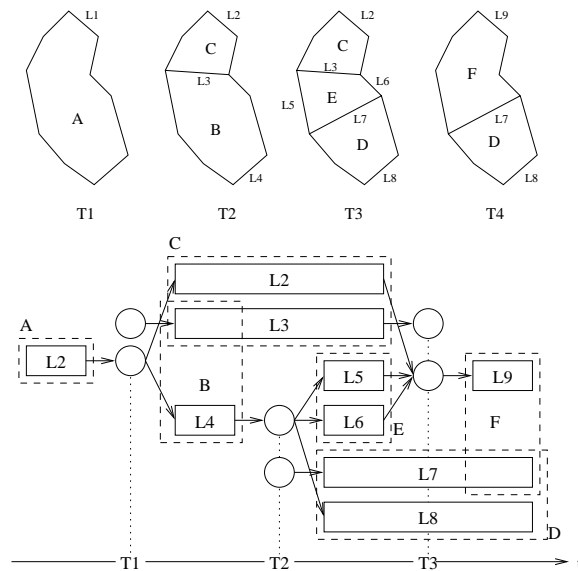


Figure 3.11: The story from Section 3.2.2, now with numbered lines

As we can see from Figure 3.11, polygon *A* is formed by a single line *L1*. To split *A* into *B* and *C*, *L1* is split into *L2* and *L4*, and *L3* is added. To split *B* into *E* and *D*, *L4* is split into three pieces: *L5*, *L6* and *L8* while *L7* is added. And finally, to merge *C* and *E* into *F*, *L3* is removed and *L5*, *L2*, and *L6* are joined into *L9*. As the figure illustrates, the history graph can be quite complex. If we were to include the relationships between the patches in the same graph (shown as dashed rectangles), the graph would be virtually unreadable. It is therefore wise to present each structural level of the data set in separate graphs.

As another example, let us revisit the example in Section 3.3.1 illustrated in Figure 3.7. Assume that the road is constructed from several line segments. That is, at time *T1* road 1 (*R1*) consists of the segments *L1* and *L3*, and road 2 (*R2*) consists of *L2*. From *T1* to *T2*, *R1* includes the new segment *L6* and *L7* in addition to the old segments *L1* and *L3*. The old part is now defined as a separate road (*R3*) consisting of *L4* and *L5*, while *R2* consists of *L2* and *L8*. After *T4*, when *L8* is removed, *R2* consists of *L2* only, while *R3* consists of *L9* which is the join of *L4* and *L5*. As the history graph in Figure 3.12 shows, *R3* is seen as being separated

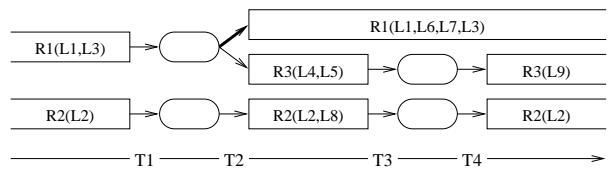


Figure 3.12: A higher level view of the story outlined in Section 3.3.1

from $R1$ in a splitting process.

3.5 Temporal Topology in the Data Structure

Topology is the study of those properties of geometrical forms that remain invariant under certain transformations such as bending, stretching, etc. (Webster's dictionary). In other words, we are speaking of relationships between objects such as neighbour of, overlap, disjointness and insiderness. Temporal topology would include time as another spatial dimension and involve temporal relationships between objects. Neighbours in time would either be predecessors or successors while various forms of co-existence are relevant relationships between objects that are not temporal neighbours [All83].

The advantage of the history graph is that these temporal relationships can be represented explicitly in the data structure. Temporal neighbours are states of objects that are linked together by a single event or change. Even relationships between events and changes can be determined as these are related through the states of the objects (Figure 3.4).

The model that we have presented in this paper is primarily a conceptual model, but by using abstract data types or an object-oriented programming language, it is possible to implement the graph structure. The advantage of this approach is that the programmers job is simplified as the data structure can be visualized in a history graph, and that the temporal topology is explicitly embedded in the data structure. Price [Pri89] have implemented a temporal cadastral data structure based on *parcels* and *transactions* as abstract data types. He also accommodates the persistence of identity throughout a transaction.

Using the object-oriented methodology, it is possible to take advantage of inheritance when implementing the model. Each of the seven types of changes may

be implemented as a different subclass of a generic change class.

However, the question is, how will such an implementation perform? It is reasonable to believe that it may perform well if equipped with proper indexing methods providing short accessing paths to the data. Moreover, an implementation based on both processes and states should be well suited for most types of queries, not having the deficiencies of the event-based or the state-based model. Hence, implementability must be further investigated and prototypes must be tested to determine the performance of the models.

3.6 *Concluding Remarks*

In this paper a general conceptual model for spatiotemporal data, called *history graph* is presented. The model is useful to visualize temporal relationships between objects and their states, and may therefore aim at solving specific problems at hand. However, there are more issues to be studied, and the model may not be suitable to explain all issues. It is reasonable to say that the model is fairly crude, and thus, it is necessary to extend the model even further to accommodate other relevant issues. This will also bring us closer to implementing the model using abstract data types or an object-oriented programming language. Also, further research must be done to obtain suitable indexing methods in three-dimensional, and possibly four-dimensional, space designed for spatiotemporal data. Langran have implemented a model called the *space-time composite* model and tested it with a number of indexing methods [Lan92]. It will be worth comparing Langran's result with the results of a history-graph based implementation.

A FRAMEWORK FOR TEMPORAL AND SPATIOTEMPORAL MODELLING BASED ON SET-THEORY

4.1 Introduction

The set-theory provides the most general and basic concepts of mathematics and applications of computer science. The relational database model is based on the set-theory, and it is likely that this, in turn, is the key reason to the success of relational database systems today. The set-theory also plays an important role in object-oriented database systems and in spatial modelling. In the latter case through the branch of set-theory known as *point-set topology*.

However, when the need for *temporal database theories* emerged, these theories were extended from the corresponding non-temporal database theories. Instead, we suggest that such theories should be based on the temporal set-theory in the same way that snapshot databases are based on the set-theory. We believe that a mathematical foundation for temporal database theories aids in ensuring the comprehensiveness and a logical soundness of these theories. Unfortunately, we could not find a temporal set-theory in the literature, so we decided it was worth investigating it on our own.

The objective of this article is therefore to propose the temporal set-theory and to discuss its applications in temporal databases and in spatiotemporal data modelling. Moreover, we also propose a new term called *tropology* that accompany the temporal set-theory. The term ‘tropology’ stems from the Greek word ‘tropos’, meaning ‘to change’ or ‘to turn’. Tropology can therefore generally be defined as the *study of changes*. We propose this term because we believe that a full fledged

temporal database theory must take all aspects of time into consideration; not only dealing with properties of objects over time, but also changes amongst objects. Using the temporal set-theory, it should be possible to describe and define changes and occurrences in a more formal manner.

We do not attempt to study the temporal set-theory and its properties from a strict mathematical point of view. We are more interested in the concepts behind the theory, and how to apply the results in the context of temporal databases and spatiotemporal modelling problems.

The remainder of this article is organized as follows: The next section discusses the time domain and suggest a continuous model of time where the elements are instants. Section 4.3 introduces the temporal set theory, and Section 4.4 briefly introduces extensions into the bitemporal setting. In Section 4.5 we make an attempt to introduce topology and to describe different types of changes, and in Section 4.6 we propose an implementational model of temporal and spatiotemporal data. Finally, Section 4.7 provides some concluding remarks.

4.2 *The Time Model*

4.2.1 *Different Models of Time*

There are many aspects to be considered when choosing a suitable time model. We use the symbol \mathbb{T} to denote the set of all times. Initially, we assume a continuous model of time which according to [Ben83] is:

1. transitive, i.e. if $t_1 < t_2$ and $t_2 < t_3$ then $t_1 < t_3$.
2. irreflexive, i.e. if $t_1 < t_2$ then $t_2 \not< t_1$
3. linear, i.e. for all t_1 and t_2 , either $t_1 = t_2$, $t_1 > t_2$ or $t_2 < t_1$.
4. successive, i.e. for all t_1 there exists a t_2 such that $t_1 < t_2$ and a t_3 such that $t_3 < t_1$.
5. dense, i.e. for all t_1 and t_2 with $t_1 < t_2$, there exist at least one t_3 such that $t_1 < t_3 < t_2$.

Because of the density requirement, each member of \mathbb{T} can have no duration, and the members of \mathbb{T} are therefore called *instants*. In order to compute the duration of time intervals, we introduce a *metric* which is a non-negative number $\delta(t_1, t_2)$ that computes the *time difference* between any two instants such that:

$$\delta(t_1, t_2) = |t_2 - t_1| \quad (4.1)$$

Let \mathbb{T}_R denote the continuous model of time. This model is isomorphic to the real numbers. The continuous model is of course beautiful in the mathematical sense, but falls short in practical applications [EGSV97]. Indeed, there are several practical arguments for a discrete model. At least we need some kind of discrete encoding [Sno92]. In the discrete model of time, we replace the density axiom (axiom 5), with a new axiom [Ben83]:

5. discrete, i.e. for all t_1 and t_2 , $t_1 < t_2$ implies that there exist a t_3 such that $t_1 < t_3$ and there is no t_4 with $t_1 < t_4 < t_3$. And, for all t_1 and t_2 , $t_1 < t_2$ implies that there exist a t_3 such that $t_3 < t_2$ and there is no t_4 with $t_3 < t_4 < t_2$

This axiom alone, opens up for the possibility of a start and an end point. In many situations this makes sense. However, Axiom 4 ensures that no such end points exist. Therefore, the discrete model is isomorphic to the integers. However, we may imagine two discrete models of time. In the *discrete instant model*, denoted \mathbb{T}_i , each member of \mathbb{T}_i is an instant. It is therefore obvious that

$$\mathbb{T}_i \subseteq \mathbb{T}_R \quad (4.2)$$

However, in the *discrete chronon model*, denoted \mathbb{T}_c , each member of \mathbb{T}_c is a time interval with a fixed duration called a *chronon*. \mathbb{T}_c can therefore be looked upon as a family of intervals from \mathbb{T}_R , such that each chronon has the same duration. The intervals are pairwise disjoint, such that the union of all intervals equals \mathbb{T}_R .

It is the discrete chronon model that has been adopted in the temporal database community. This is because it has a simple semantic, and because it is intuitive and conforming to the way people treat time in daily life. However, because we are dealing with mathematics in this article, we will stick to the continuous model of time, and we use the symbol \mathbb{T} to denote the continuous time domain. However, the theory will also apply to a discrete model of time, although we might have to change the axiomatic schema.











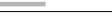

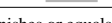

Relation	Symbol for		X	Y
	Symbol	Inverse		
X before Y	<	>		
X equal Y	=	=		
X meets Y	m	mi		
X overlaps Y	o	oi		
X during Y	d	di		
X starts Y	s	si		
X finishes Y	f	fi		
X in Y	\subseteq	\supseteq	during, start, finishes or equals	

Figure 4.1: Relationships between time intervals

4.2.2 Lifespans and Time Intervals

In many occasions we are interested when a certain condition is true. Let p be a proposition. The *lifespan* of the proposition p is the times for which the proposition holds. It is obvious that a proposition may not hold for all times in \mathbb{T} but some subset of \mathbb{T} . This subset may not be connected if the proposition holds for some time period, then does not hold for some subsequent time period, then to hold again at a later time.

According to [JCE⁺94], such a lifespan is called a *temporal element* and is defined as a finite union of time intervals. However, since we are dealing with the temporal set-theory, and since sets are said to contain ‘elements’, there is a risk of causing confusion of terminology if a set contains elements that varies over time (and therefore would be called ‘temporal elements’ as well). Therefore, we avoid using this term here.

In many cases, the lifespan of a proposition may be restricted to be a *connected* subset of \mathbb{T} . Such a lifespan is called (as already indicated) a *time interval*. Time intervals play an important role in the temporal set-theory. Allen [All83], has proposed a set of 13 operators to describe different relationships between time intervals. In order to mathematically define these, we use the notation \underline{X} to denote the greatest lower bound of the time interval X and \overline{X} to denote the least upper bound of X . In order to not complicate matters, we assume that all intervals are closed intervals, thus two intervals that meet have exactly one time instant in common.

Figure 4.1 illustrates Allens’s operators, whereas as the mathematical definitions follows below:

$$\mathbf{X} < \mathbf{Y} \iff \overline{\mathbf{X}} < \underline{\mathbf{Y}} \quad (4.3)$$

$$\mathbf{X} = \mathbf{Y} \iff \overline{\mathbf{X}} = \overline{\mathbf{Y}} \wedge \underline{\mathbf{X}} = \underline{\mathbf{Y}} \quad (4.4)$$

$$\mathbf{X} \text{ m } \mathbf{Y} \iff \overline{\mathbf{X}} = \underline{\mathbf{Y}} \quad (4.5)$$

$$\mathbf{X} \text{ o } \mathbf{Y} \iff \underline{\mathbf{X}} < \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} > \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} < \overline{\mathbf{Y}} \quad (4.6)$$

$$\mathbf{X} \text{ d } \mathbf{Y} \iff \underline{\mathbf{X}} > \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} < \overline{\mathbf{Y}} \quad (4.7)$$

$$\mathbf{X} \text{ s } \mathbf{Y} \iff \underline{\mathbf{X}} = \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} < \overline{\mathbf{Y}} \quad (4.8)$$

$$\mathbf{X} \text{ f } \mathbf{Y} \iff \underline{\mathbf{X}} > \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} = \overline{\mathbf{Y}} \quad (4.9)$$

$$\mathbf{X} \subseteq \mathbf{Y} \iff \underline{\mathbf{X}} \geq \underline{\mathbf{Y}} \wedge \overline{\mathbf{X}} \leq \overline{\mathbf{Y}} \quad (4.10)$$

$$(4.11)$$

If \mathbf{X} is a time interval, then the duration of \mathbf{X} , denoted $|\mathbf{X}|$, is the metric distance between the greatest lower bound and the least upper bound of \mathbf{X} . In other words:

$$|\mathbf{X}| = \delta(\underline{\mathbf{X}}, \overline{\mathbf{X}}) = \overline{\mathbf{X}} - \underline{\mathbf{X}} \quad (4.12)$$

Since a lifespan is a finite set of time intervals, it is also possible to compute the duration of any lifespan. Let \mathbf{L} be a lifespan and let Φ be a function that transforms a lifespan into a family of disjoint time intervals, then the duration of \mathbf{L} , denoted $|\mathbf{L}|$ is given by

$$|\mathbf{L}| = \sum_{\mathbf{X}_i \in \Phi(\mathbf{L})} |\mathbf{X}_i| \quad (4.13)$$

4.2.3 The Bitemporal Time Model

In databases, there are normally a difference between the time when something occurs in the real world, and the time when the occurrence is recorded in the database. We therefore distinguish between *valid time* which is pertinent to when facts are true in the real world, and *transaction time* which is pertinent to the time when facts are current in the database. These two times are considered to be orthogonal, and therefore, time is said to be two-dimensional [SA86] [Sno92]. A model that includes both valid time and transaction time is said to be a *bitemporal* model.

In this article, we will both present a model which is based on a one-dimensional representation of time (i.e. mono-temporal) and its bitemporal extension. For most of the time, the mono-temporal model will be pertinent to the valid time dimension. When it is necessary to distinguish between the two time dimensions, we use the symbol \mathbb{T}_v to denote the set of valid times and the symbol \mathbb{T}_t to denote the set of transaction times. Then, the set of bitemporal times is given by:

$$\mathbb{T}^2 = \mathbb{T}_t \times \mathbb{T}_v \quad (4.14)$$

4.3 The Temporal Set Theory

4.3.1 Background and Related Work

As we already have stated, we have not found any proposed temporal set-theory in literature. On the other hand, several text books exist on the topic of *temporal logic* or *tense logic* (such as [RU71] [Ben83] [McA76]), and logic often goes hand in hand with set theory; e.g. given a predicate $p(x)$ which determines whether some condition holds for an object x , we can build a set A consisting of all objects which satisfy $p(x)$

$$A = \{x | p(x)\}. \quad (4.15)$$

In [RU71], a special type of logic, called *topological logic*, is discussed. Given a topological space X and a proposition p , the expression $P(x, p)$ is to be understood as the proposition p realized at the position x . In principle, we could replace the proposition p with a predicate $p(x)$ where x is a position in X .

Since time is a topological space, the topological logic can be utilized in temporal logic. Furthermore, the time space has also a total order, which is not entirely the case for Euclidean space, at least for dimensions of two and higher. This has led to the definition of four *tense operators*:

$$Fp \quad \text{sometimes in the future } p \quad (4.16)$$

$$Pp \quad \text{sometimes in the past } p \quad (4.17)$$

$$Gp \quad \text{it will always be the case that } p \quad (4.18)$$

$$Hp \quad \text{it has always been the case that } p \quad (4.19)$$

It is worth noticing that $Gp = \neg F\neg p$ and $Hp = \neg P\neg p$.

4.3.2 The Classic (Naive) Set Theory

The classic set theory has provided a basis for many areas within computer science, such as database systems and programming languages. Naturally, it will provide a basis for the temporal set theory as well. Therefore, we present a brief review of the most basic definitions of the set-theory that we need in the development of the temporal set theory.

- A *set* is a collection of objects called *elements* or *members* of the set. A set is finite if it contains a finite number of elements. A set is infinite if it is not finite.
- The set A is said to be a *subset* of a set B , denoted $A \subseteq B$, if and only if every member of A is also a member of B .
- If A is a set, the *power set* of A , denoted $P(A)$, is the set of all subsets of the set A . In other words,

$$P(A) = \{B \mid B \subseteq A\} \quad (4.20)$$

Note that both the set \emptyset and the set A itself are members of $P(A)$.

- If A and B are sets, then the *Cartesian product* of A and B , denoted $A \times B$, is the set of all *ordered pairs* $\langle a, b \rangle$ where $a \in A$ and $b \in B$.
- If A_1, \dots, A_n are sets, the Cartesian product $B = A_1 \times \dots \times A_n$ is the set of all *n-tuples* $\langle a_1, \dots, a_n \rangle$ in B such that $a_i \in A_i$ for all $i = 1, \dots, n$.
- If A and B are sets, then a *function* f from A to B , denoted $f : A \rightarrow B$, is an assignment of a unique element of B to each element of A . We write $f(a) = b$ if b is the unique element of B assigned by the function f to the element a of A . A is called the *domain* of f while B is called the *codomain* of f .
- If A and B are sets, then a *binary relation* from A to B is a subset of $A \times B$. In other words, a binary relation from A to B is a set of ordered pairs where the first element of each ordered pair is a member of A and the second element is a member of B . Likewise, an *n-ary relation* over the sets A_1, \dots, A_n is a subset of the Cartesian product $A_1 \times \dots \times A_n$.

- A *relation* on the set \mathbf{A} is a relation from \mathbf{A} to \mathbf{A} .
- If \mathbf{A} and \mathbf{B} are sets, the set of functions from \mathbf{A} into \mathbf{B} , i.e. the set of functions with \mathbf{A} as domain and \mathbf{B} as codomain, is denoted $\mathbf{B}^{\mathbf{A}}$, and is formally defined by

$$\mathbf{B}^{\mathbf{A}} = \{f \mid f : \mathbf{A} \rightarrow \mathbf{B}\} \quad (4.21)$$

4.3.3 Attributes and Domains

Let a be any object, say, an attribute and let \mathbf{A} be a set containing all legal values for a . Then, the set \mathbf{A} is called the *domain* of a . We may also say that the set \mathbf{A} provides a *state space* for a .

As with programming languages, we wish to provide a set of standard domains for primitive objects, such as integers, reals and so on. Most attribute domains will be subsets of these. We will use black-board bold fonts to denote these standard domains, and for the remainder of the article we define the following domains, and their respective sub-domains:

- The set of Booleans $\mathbb{B} = \{true, false\}$. This is a fundamental set containing two elements, *true* and *false*. In the literature, this set is often denoted $\mathbf{2}$, which contain the elements 0, and 1. Hence, we may also refer to this set as the *binary set*. However, \mathbb{B} and $\mathbf{2}$ are equivalent, but we will use \mathbb{B} in the remainder of this article.
- The set of real numbers, \mathbb{R} . A commonly used subset of \mathbb{R} is the unit interval $\mathbb{I} = [0, 1]$. The unit interval has its application in the fuzzyset-theory and probability. Another common subset of \mathbb{R} is $[-1, 1]$ which is used in spectral analysis.
- The set of integers, \mathbb{Z} . The set of natural numbers \mathbb{N} , the set of positive integers \mathbb{Z}^+ and so on, are subsets of \mathbb{Z} . Note that we treat the set of integers as a subset of the set of reals, i.e. $\mathbb{Z} \subseteq \mathbb{R}$.
- The set of complex numbers, \mathbb{C} . The set of complex numbers may not at first sight look interesting, but in temporal analysis they play an important role in relation to the study of periodicity and the related Fourier analysis.

- The set of characters, \mathbb{V} . Today, a plethora of character sets have been defined such as the ASCII set. For the sake of being able to describe words in most languages using Latin letters, we may choose to adhere to the ISO8859-1, or the unicode character set. For other languages, we may have the Cyrillic characters set, the Hebrew character set and so forth.
- The set of all strings, \mathbb{W}_V , which is the set of all sequences (or n -tuples) of characters from \mathbb{V} . \mathbb{W} has no inherent order, but a lexicographical order can be defined if a total order is defined over \mathbb{V} .
- The set of all spatial elements, \mathbb{S} . As we want to model the spatial part of objects as an attribute of the object, we need to define this domain for these kinds of attributes. This set need to be more closely specified depending on the number of dimensions modelled, and the type of objects modelled.
- The set of all times, \mathbb{T} , with the corresponding valid time domain \mathbb{T}_v , transaction time domain \mathbb{T}_t and bitemporal domain \mathbb{T}^2 .

4.3.4 Functions over Time

In the literature temporal attributes of some domain \mathbf{A} are often modelled as subsets of the Cartesian product $\mathbf{A} \times \mathbb{T}$. However, since we assume a linear model of time, we can allow only one value of \mathbf{A} for each value of \mathbb{T} . It therefore makes sense to express everything that varies over time, as functions over time.

A function f over time is a function from the time domain \mathbb{T} into some arbitrary codomain \mathbf{A} :

$$f : \mathbb{T} \rightarrow \mathbf{A}. \quad (4.22)$$

Because we assert that such functions return a value even for times when the attribute is not defined, or does not exist, we introduce the symbol \perp (null) which is used to indicate that an object did not exist at a particular time. Thus, any domain \mathbf{A} has a corresponding *temporal domain*

$$\mathbf{A}^T = \{f | f : \mathbb{T} \rightarrow (\mathbf{A} \cup \{\perp\})\} \quad (4.23)$$

which is the set of all functions with \mathbb{T} as domain and $(\mathbf{A} \cup \{\perp\})$ as codomain. For convenience, we introduce the notation \mathbf{A}_\perp to denote that the null object has been

added to the set \mathbf{A} . In other words,

$$\mathbf{A}_\perp = \mathbf{A} \cup \{\perp\} \quad (4.24)$$

To each function $f(t)$ we assign a *lifespan* $LS(f)$, which according to [CC87] [JCE⁺94], can be any subset of \mathbb{T} , representing the times at which the attribute is defined. Formally, we define the lifespan of a function over time as a function from a temporal domain \mathbf{A}^T into the power set $P(\mathbb{T})$,

$$LS : \mathbf{A}^T \rightarrow P(\mathbb{T}), \quad (4.25)$$

such that

$$LS(f) = \{t \in \mathbb{T} \mid f(t) \neq \perp\}. \quad (4.26)$$

In some cases, we restrict a lifespan to be a *connected* subset of \mathbb{T} , i.e. a *time interval*. A time interval can be identified by an ordered pair $\langle t_1, t_2 \rangle$ in $\mathbb{T} \times \mathbb{T}$, where $t_1 \leq t_2$. Since the distinction between lifespans and time intervals is important, we will define the set $I(\mathbb{T})$ to be the set of all time intervals, and $P(\mathbb{T})$ to be the set of all lifespans. Thus

$$I(\mathbb{T}) = \{\mathbf{X} \mid \mathbf{X} \subseteq \mathbb{T} \text{ and } \mathbf{X} \text{ is connected}\} \quad (4.27)$$

From this definition it is clear that $I(\mathbb{T}) \subset P(\mathbb{T})$.

If the lifespan of a function contains only one instant, the duration is zero. The semantics of this is significant, since it allows us to represent instantaneous facts such as events (instantaneous changes) or snapshots (instantaneous states).

4.3.5 Temporal n -tuples and the Product of Temporal Domains

An object is usually described by a set of attributes, and can therefore be said to be an n -tuple. If we assume that the attributes of a temporal object are given as functions over time, then also the object itself can be given as a function over time. Such a function is in this article referred to as a *temporal n -tuple*. Temporal objects may also contain time-invariant attributes which complicates matters a little. However, initially we are going to assume that all attributes of an object is given as functions over time, since time-invariant attributes can be represented as functions that return the same value for all times.

Let $\mathbf{A}_1, \dots, \mathbf{A}_n$ be (non-temporal) sets. A temporal n -tuple over the product $\mathbf{A}_1^T \times \dots \times \mathbf{A}_n^T$ is given by

$$F(t) = \langle f_1(t), \dots, f_n(t) \rangle \quad (4.28)$$

However, we must distinguish between the lifespans of each entry in the tuple, and the lifespan of the tuple itself, e.g. a 3-tuple at one time $F(t_1) = \langle \perp, \perp, \perp \rangle$ is semantically different from $F(t_2) = \perp$ at another time. Hence the product described here is not really a Cartesian product. However, we define the product of temporal domains as the set of *temporal n -tuples* which are functions from \mathbb{T} into the Cartesian product $(\mathbf{A}_{1,\perp} \times \dots \times \mathbf{A}_{n,\perp})$ plus \perp . More specifically,

$$\mathbf{A}_1^T \times \dots \times \mathbf{A}_n^T = \{F|F : \mathbb{T} \rightarrow (\mathbf{A}_{1,\perp} \times \dots \times \mathbf{A}_{n,\perp} \cup \{\perp\})\} \quad (4.29)$$

As already indicated above, it is often the case that an attribute of an object is not defined for the entire lifespan of that object. In some cases, attributes may even be maintained and updated for objects that do not logically exist, e.g. physicians may keep records of babies even before they are born. This is quite problematic to deal with. Hence, we assume that no entry in a temporal n -tuple can exist unless the n -tuple itself exist. In other words, we assert that

$$LS(f_i) \subseteq LS(F) \quad (4.30)$$

for all $i = 1, \dots, n$.

Until this point, we have assumed that all entries in a temporal n -tuple are functions over time. In practice, it would be more logical that time invariant attributes (such as identifiers) were not represented as functions over time. Therefore, we should be able to construct products of a mixture of temporal and non-temporal domains.

If at least one of the entries in an n -tuple is a function over time, then also the n -tuple itself is a function over time, since it has different values for different times, e.g. consider the real functions $f(t) = 2t$ and $g(t) = t^2 - 3$, and the constant $c = 5$. Let the temporal n -tuple $F(t)$ be given by

$$F(t) = \langle c, f(t), g(t) \rangle, \quad (4.31)$$

then the value of $F(t)$ for $t = 4$ is given by

$$F(4) = \langle 5, 8, 13 \rangle \quad (4.32)$$

4.3.6 Functions and Operators over Temporal Domains

Recall that a function $f : \mathbf{A} \rightarrow \mathbf{B}$ is an assignment of a unique element of \mathbf{B} to each element of \mathbf{A} . If \mathbf{A} is a single set, the function is a unary function. If \mathbf{A} is the Cartesian product of two sets, the function is a binary function. Both unary and binary functions are often expressed in terms of operators, such as $-b$ or $a + b$, but a strict functional notation is also possible such as $neg(b)$ or $sum(a, b)$. Hence, we will stick to the term *function* in the remainder of this article, even if we are speaking of operators.

Let Ω be the set of all functions over non-temporal domains and let ω be any function in Ω . The functions in Ω are characterized by their domain and codomain, e.g. we may specify the following functions:

$$+ : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{R} \quad \text{e.g.} \quad 3 + 4 = 7 \quad (4.33)$$

$$> : (\mathbb{R} \times \mathbb{R}) \rightarrow \mathbb{B} \quad \text{e.g.} \quad 4.5 > 6.7 = \text{false} \quad (4.34)$$

$$\text{StrLength} : \mathbb{W} \rightarrow \mathbb{N} \quad \text{e.g.} \quad \text{StrLength}(\text{"Norway"}) = 6 \quad (4.35)$$

To the set Ω , there is a corresponding set Ω_T , and for every function ω in Ω there is a corresponding *temporal function* ω_T in Ω_T , usually with the same symbol as ω , such that if

$$\omega : \mathbf{A} \rightarrow \mathbf{B}, \quad (4.36)$$

then,

$$\omega_T : \mathbf{A}^T \rightarrow \mathbf{B}^T. \quad (4.37)$$

This means that if we can add two integers and get another integer, we can also add two temporal integers (i.e. integers that are expressed as functions over time) and get another temporal integer, e.g.

$$f_a(t) + f_b(t) = f_c(t). \quad (4.38)$$

Hence, if $\omega(a, b) = c$ then $\omega_T(f_a, f_b) = f_c$ means that for every t such that $a = f_a(t)$, with $a \neq \perp$, and $b = f_b(t)$, with $b \neq \perp$, there exist a $c = f_c(t)$ with $\omega(a, b) = c$. Usually, if for any t , $f_a(t) = \perp$ or $f_b(t) = \perp$, then also $f_c(t) = \perp$. This ensures that $f_c(t)$ is only defined for those values of t where both $f_a(t)$ and

$f_b(t)$ are defined. However, this may not apply to all functions, e.g. the function that takes two sets and returns their union, may return a non-null result if one of sets are null.

Now, what happens if we add a non-temporal integer to a temporal integer, say $a + f_b$. It is obvious that the result is another temporal integer f_c , e.g. if $a = 2$ and $f_b = 3t+4$, we get $f_c = 3t+6$. This problem corresponds to the problem described in Section 4.3.5. This way, if one of the arguments of a function is a function over time, the result is also a function over time. In other words, if $f : \mathbf{A} \times \mathbf{B} \rightarrow \mathbf{C}$ then,

$$f : \mathbf{A}^T \times \mathbf{B} \rightarrow \mathbf{C}^T \quad (4.39)$$

$$f : \mathbf{A} \times \mathbf{B}^T \rightarrow \mathbf{C}^T \quad (4.40)$$

$$f : \mathbf{A}^T \times \mathbf{B}^T \rightarrow \mathbf{C}^T. \quad (4.41)$$

In addition, there are, of course, many other functions which take functions over time as argument, and/or return a function over time as a result that are not members of Ω_T (since they have no corresponding function in Ω). However, such functions can be freely defined in any algebra associated with the temporal set-theory.

4.3.7 Temporal Sets

A set may contain a number of members. Over time, a set may receive new members while existing members may secede their membership. The members of a temporal set can be a non-temporal object, a function over time, or another set or temporal set. A *temporal set* is a structure that keeps track of which elements were members of the set at which times.

There are many ways of expressing temporal sets using classic set theory:

- *As a relation:* Given any domain \mathbf{A} and the time domain \mathbb{T} , the temporal set τ_R is a subset of $\mathbf{A} \times \mathbb{T}$. Then, for all ordered pairs $\langle a, t \rangle$ in τ_R where a is an element of \mathbf{A} and t is an element of \mathbb{T} , a is said to be a member of τ_R at time t .
- *As a function into \mathbb{B}^T :* Let \mathbf{A} be any domain. Then, a temporal set τ_μ is a function $\tau_\mu : \mathbf{A} \rightarrow \mathbb{B}^T$ that for every element a of \mathbf{A} returns the *temporal*

characteristic function for a . Let $\mu = \tau_\mu(a)$ then $\mu(t) = \text{true}$ if a is a member of τ_μ at time t , and $\mu(t) = \text{false}$ otherwise.

However, since we have chosen to express temporal attributes as functions over time, we also choose to define a temporal set as a function over time: given any domain \mathbf{A} and the time domain \mathbb{T} , a temporal set τ_F over \mathbf{A} can be described as a function from \mathbb{T} into the power set $P(\mathbf{A})$

$$\tau_F : \mathbb{T} \rightarrow P(\mathbf{A}) \quad (4.42)$$

that for every time t in \mathbb{T} returns a subset of \mathbf{A} containing the elements that are members of τ_F at that time. Since the expression $\tau(t)$ denotes a normal non-temporal set (we omit the subscripts from now on) we use the notation

$$a \in \tau(t) \quad (4.43)$$

to indicate that the element a is a member of the temporal set τ at a time t . Equivalently, we use the expression

$$a \notin \tau(t) \quad (4.44)$$

to express that a is not a member of τ at time t .

The *membership lifespan* of an element a with respect to a temporal set τ , denoted $LS(a \in \tau)$, is the set of times at which a is a member of τ . In other words,

$$LS(a \in \tau) = \{t \in \mathbb{T} | a \in \tau(t)\} \quad (4.45)$$

Following this definition, it is clear that the assertion

$$LS(a \in \tau) \subseteq LS(\tau) \quad (4.46)$$

must hold. Another assertion that may be introduced is that a set cannot contain dead members, in other words

$$LS(a \in \tau) \subseteq LS(a). \quad (4.47)$$

However, this axiom should only be optional as it is possible that some types of sets may contain dead members, e.g. some members of, say, the GIS hall of fame, may no longer be alive.

To clarify the notion of a temporal set, an example can be enlightening. Let EU be the set of all members in the European Union (formerly EEC). Then the set $EU(1958) = \{\text{Germany, France, Italy, Belgium, Netherlands, Luxembourg}\}$ while the set $EU(1999)$ contains all the countries that are members of the European Union in 1999.

4.3.8 Common Operations on Temporal Sets

Equality and the Equality Function

Two temporal sets τ_1 and τ_2 are *equal* if they have the same lifespan, i.e. if $LS(\tau_1) = LS(\tau_2)$, and if they for all times in this lifespan, contain the same elements.

However, this construct is rather restrictive, and in many cases we want to express that two sets are equal at some times while not equal at some other times. For this purpose we define the equality function, denoted $=_T$, as a function in \mathbb{B}^T . Let τ_1 and τ_2 be temporal sets, the *equality function* for temporal sets is defined such that for all t in \mathbb{T} :

$$(\tau_1 =_T \tau_2)(t) = \begin{cases} \perp & \text{if } \tau_1(t) = \perp \text{ or } \tau_2(t) = \perp \\ \text{true} & \text{if } \tau_1(t) = \tau_2(t) \\ \text{false} & \text{otherwise} \end{cases} \quad (4.48)$$

This definition clearly indicates that the result of $\perp =_T \perp$ is undefined. It is natural that many would question this, but an example should help. Let $f(t)$ and $g(t)$ denote the salary of two persons living in the 20th century. Then it is obvious that the result of $f(t) =_T g(t)$ must be undefined for $t = 1600$, because it does not make sense to compare the salaries of persons that are not alive.

In other situations, we may only be interested in the *times* at which two temporal sets are equal. Since the statement $\tau_1 = \tau_2$ is a proposition, then we can obtain the lifespan of this proposition, $LS(\tau_1 = \tau_2)$, by the formulae

$$LS(\tau_1 = \tau_2) = \{t \in \mathbb{T} \mid \tau_1(t) = \tau_2(t)\}. \quad (4.49)$$

Note that $LS(\tau_1 = \tau_2)$ is not the same as $LS(\tau_1 =_T \tau_2)$

Empty Temporal Sets

A temporal set is *always empty* if it never contained any elements during its lifespan.

We may also define the *temporal empty set*, denoted \emptyset^T , as the set whose lifespan is \mathbb{T} and which is empty at all times. Then we use this set together with the equality function to check at which times a temporal set is empty or not.

The Temporal Characteristic Function

Given any domain \mathbf{A} and the time domain \mathbb{T} , the *temporal characteristic function* μ_τ of an element a of \mathbf{A} with respect to a temporal set τ , is a function

$$\mu_\tau : \mathbf{A} \rightarrow \mathbb{B}^T \quad (4.50)$$

that returns a binary function over time. This function is true for all times when a is a member of τ , and false for all times when a is not a member of τ . A nice property with (4.50) is that it can easily be extended to a *temporal fuzzy set* by replacing \mathbb{B}^T with \mathbb{I}^T .

Temporal Subsets

If τ_1 and τ_2 are temporal sets with respect to some domain \mathbf{A} , then τ_1 is a *temporal subset* of τ_2 if $LS(\tau_1) \subseteq LS(\tau_2)$ and for all t in $LS(\tau_1)$,

$$\tau_1(t) \subseteq \tau_2(t) \quad (4.51)$$

To indicate the times at which τ_1 is a subset of τ_2 we can use the expression $LS(\tau_1 \subseteq \tau_2)$.

The subset relation expressed as a function over time determine the truth values for which times a subset relation between two sets holds. Thus, the subset relation as a function over time, denoted \subseteq_T , can be defined as

$$\subseteq_T : \tau_1 \times \tau_2 \rightarrow \mathbb{B}^T \quad (4.52)$$

where,

$$(\tau_1 \subseteq_T \tau_2)(t) = \begin{cases} \perp & \text{if } \tau_1(t) = \perp \text{ or } \tau_2(t) = \perp \\ \text{true} & \text{if } \tau_1(t) \subseteq \tau_2(t) \\ \text{false} & \text{otherwise} \end{cases} \quad (4.53)$$

The definition above naturally raises a couple of questions. The questions are, assuming that \mathbf{A} is a non-null set, whether $\perp \subseteq \mathbf{A}$ and whether $\perp \subseteq \perp$. Let us assume the example of the EU and EMU (the European Monetary Union). Then the first question would correspond to whether the EMU countries are a subset of the EU countries in 1986? Since the EMU did not yet exist in 1986, it does not make sense to ask the question, hence $\perp \subseteq \mathbf{A}$ is undefined. For the same reason, it is obvious that also $\perp \subseteq \perp$ is undefined.

Union, Intersection, Difference and Complement

By representing temporal sets as functions over time, most of the common set operations for non-temporal sets can easily be extended to temporal sets. Let τ_1 and τ_2 be temporal sets with respect to some domain \mathbf{A} , the operations union, intersection, difference and complement can respectively be defined as functions over time, such that for all t in \mathbb{T} :

$$(\tau_1 \cup \tau_2)(t) = \begin{cases} \tau_1(t) \cup \tau_2(t) & \text{if } \tau_1(t) \neq \perp \text{ and } \tau_2(t) \neq \perp \\ \tau_1(t) & \text{if } \tau_2(t) = \perp \text{ and } \tau_1(t) \neq \perp \\ \tau_2(t) & \text{if } \tau_1(t) = \perp \text{ and } \tau_2(t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \quad (4.54)$$

$$(\tau_1 \cap \tau_2)(t) = \begin{cases} \tau_1(t) \cap \tau_2(t) & \text{if } \tau_1(t) \neq \perp \text{ and } \tau_2(t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \quad (4.55)$$

$$(\tau_1 \setminus \tau_2)(t) = \begin{cases} \tau_1(t) \setminus \tau_2(t) & \text{if } \tau_1(t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \quad (4.56)$$

$$\overline{\tau_1(t)} = \begin{cases} \mathbf{A} \setminus \tau_1(t) & \text{if } \tau_1(t) \neq \perp \text{ and } \tau_2(t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \quad (4.57)$$

These definitions may seem quite intuitive, but they are not, e.g. does it make sense to take the union of all NATO countries and all EEC countries in 1954? According to our definition, it does since this set contains the countries that are either a member of NATO or of EEC. In 1954 this set will only contain the NATO members, since EEC did not form until 1958.

A similar question arises for intersection: does it makes sense to take the intersection of all NATO and EEC countries in 1954? According to our definition, it does not which means that the intersection is undefined, but it might also be a question whether the intersection for 1954 should be defined, but empty. And for set

difference, does it makes sense to compute difference between the NATO countries and the EEC countries (i.e. the set of the NATO countries who are not members of the EEC) in 1954? According to our definition, it does not since we assume that $\tau_1 \setminus \tau_2 = \tau_1 \cap \overline{\tau_2}$. However, if the definition of intersection is changed, so must the definition of set difference be changed accordingly

The Cardinality Function

The *cardinality* of a temporal set is also a function over time. Let τ be a temporal set, then the cardinality $|\tau|$ is a function in $\mathbb{N}^{\mathbb{T}}$ such that for all t in \mathbb{T} , the following holds:

$$|\tau|(t) = \begin{cases} |\tau(t)| & \text{if } \tau(t) \neq \perp \\ \perp & \text{otherwise} \end{cases} \quad (4.58)$$

The Support of a Temporal Set

The *support* of a temporal set τ denoted $supp(\tau)$ is a set that contains all elements who for at least one t in \mathbb{T} is a member of τ . In other words, for all t in \mathbb{T} ,

$$supp(\tau) = \bigcup_{t \in \mathbb{T}} \tau(t) \quad (4.59)$$

4.3.9 The Temporal Cartesian Product and Relations on Temporal Sets

A relation between two (non-temporal) sets A and B is a subset of the Cartesian product $A \times B$. Similarly, we want to define a *temporal relation* between two temporal sets as a subset of the product $\tau_1 \times \tau_2$. Since a pair in a temporal relation can be related to each other forever (even if each entry has a finite membership lifespans in its respective set), the lifespan of a product between two temporal sets has to be \mathbb{T} . Moreover, we can also have a temporal relation between two non-temporal sets.

Let τ_1 and τ_2 be temporal sets. The *temporal Cartesian product* of τ_1 and τ_2 denoted $\tau_1 \times \tau_2$, is a temporal set that contains all ordered pairs $\langle a_1, a_2 \rangle$ where $a_1 \in supp(\tau_1)$ and $a_2 \in supp(\tau_2)$, and such that the membership lifespan of each pair in $\tau_1 \times \tau_2$ equals \mathbb{T} . In other words:

$$(a_1 \in supp(\tau_1)) \wedge (a_2 \in supp(\tau_2)) \iff \langle a_1, a_2 \rangle \in (\tau_1 \times \tau_2) \quad (4.60)$$

And for each pair $\langle a_1, a_2 \rangle$ we have

$$LS(\langle a_1, a_2 \rangle \in (\tau_1 \times \tau_2)) = \mathbb{T}. \quad (4.61)$$

This, again, requires according to (4.46) that

$$LS(\tau_1 \times \tau_2) = \mathbb{T} \quad (4.62)$$

Hence, if a temporal set τ_1 is considered to be a subset of $supp(\tau_1) \times \mathbb{T}$, and in the same way, τ_2 is a subset of $supp(\tau_2) \times \mathbb{T}$, then $\tau_1 \times \tau_2$ correspond to the Cartesian product $supp(\tau_1) \times supp(\tau_2) \times \mathbb{T}$.

As indicated above, we also define a product as a temporal set from non-temporal sets. If τ_1 contains elements from the set \mathbf{A}_1 and τ_2 contains elements from the set \mathbf{A}_2 , then we introduce the *temporalized Cartesian product* of two non-temporal sets, denoted $\mathbf{A}_1 \times_T \mathbf{A}_2$, such that the result is a temporal set. In other words, the temporalized Cartesian product of two sets \mathbf{A}_1 and \mathbf{A}_2 correspond to the Cartesian product $\mathbf{A}_1 \times \mathbf{A}_2 \times \mathbb{T}$. In this case, note that if $supp(\tau_1) \subset \mathbf{A}_1$ or $supp(\tau_2) \subset \mathbf{A}_2$, then also $\tau_1 \times \tau_2 \subset \mathbf{A}_1 \times_T \mathbf{A}_2$.

Now, that we have established the the concept of the temporal Cartesian product, we can move on to define *temporal relations*. A temporal relation can be used to express that two objects in different sets (non-temporal or temporal sets) are related to each other only at some subset of \mathbb{T} . Let τ_1 and τ_2 be temporal sets with members from \mathbf{A}_1 and \mathbf{A}_2 respectively. A *binary temporal relation from τ_1 to τ_2* is a subset of $\tau_1 \times \tau_2$ (or, if you like, $\mathbf{A}_1 \times_T \mathbf{A}_2$). In other words, a binary temporal relation from τ_1 to τ_2 (or from \mathbf{A}_1 to \mathbf{A}_2) is a temporal set R_T of ordered pairs where the first element of each ordered pair comes from $supp(\tau_1)$ and the second element comes from $supp(\tau_2)$. We may use the notation $\langle a_1 R_T a_2 \rangle(t)$ to denote that $\langle a_1, a_2 \rangle \in R_T(t)$, and $\langle a_1 \not R_T a_2 \rangle(t)$ to denote that $\langle a_1, a_2 \rangle \notin R_T(t)$.

Based on these definitions, we can identify at least five properties of temporal relations between two temporal sets. Let R_T be a temporal binary relation between the two temporal sets τ_1 and τ_2 :

- A temporal relation is called a *strict temporal relation* if the following holds for all times:

$$LS(a_1 R_T a_2) \subseteq LS(a_1 \in \tau_1) \cap LS(a_2 \in \tau_2) \quad (4.63)$$

- A temporal relation is called a *loose temporal relation* if it is not strict and the following condition holds for all times:

$$LS(a_1 R_T a_2) \subseteq LS(a_1 \in \tau_1) \cup LS(a_2 \in \tau_2) \quad (4.64)$$

- A temporal relation that is not a strict temporal relation nor a loose temporal relation is called a *free temporal relation*.
- A strict temporal relation is called an *exactly strict temporal relation* or *co-existence relation* if the following holds for all times:

$$LS(a_1 R_T a_2) = LS(a_1 \in \tau_1) \cap LS(a_2 \in \tau_2) \quad (4.65)$$

- A loose temporal relation is called an *exactly loose temporal relation* or *conditional temporal relation* if the following condition holds for all times:

$$LS(a_1 R_T a_2) = LS(a_1 \in \tau_1) \cup LS(a_2 \in \tau_2) \quad (4.66)$$

To clarify the semantics of these types of relations we give three examples: Let τ_P be the temporal set of all (living) persons, and let τ_C be the temporal set of all (existing) companies. Then an employment relation from τ_P to τ_C is a strict temporal relation since no person can be employed at a company unless both exist at the same time. A person in τ_P may be inducted to the GIS hall of fame. This relationship is a loose temporal relation since a person may only be in a hall of fame if the hall of fame exist, although the person may still be in the hall of fame after his death. A grandparent relation from τ_P to τ_P is a free relation since a grandchild is related to its grandparent even if the grandparent died before the grandchild was born. The grandparent relation will start to exist from the moment the grandchild was born and will continue to exist forever, even if both grandchild and grandparent are dead ([MSW92]).

Furthermore, as we can identify binary relations on a set A to be reflexive, symmetric, and transitive, we may categorize their temporal counterparts in the same way. So, if we let x stand for reflexive, symmetric, transitive etc., we may say that a binary temporal relation R_T on a temporal set τ is x if for all t in $LS(R_T)$, $R_T(t)$ is x .

4.4 Extending to the Bitemporal Domain

So far, we have assumed that time is one-dimensional. In this section, we extend the concept to two-dimensional time. In principle, this can be achieved by replacing occurrences of the time parameter with a pair of time parameters. In this section, we give a brief overview of the extensions into the bitemporal set theory and the associated models.

4.4.1 The Bitemporal Set Theory

In the temporal set theory, we expressed attributes and sets as functions over one time dimension, preferably the valid time dimension. Extensions into the bitemporal realm is quite easy since we can replace all functions over one time dimension with functions over two time dimensions. Let \mathbb{T}_v and \mathbb{T}_t be the valid time and the transaction time dimension respectively, a bitemporal attribute f of some domain \mathbf{A} is then defined as a function

$$f : \mathbb{T}_t \times \mathbb{T}_v \rightarrow \mathbf{A}_\perp. \quad (4.67)$$

Then, $a = f(t_t, t_v)$ is the function that for all pairs of transaction time t_t and valid time t_v returns a unique element a in $\mathbf{A} \cup \{\perp\}$. Thus, any attribute domain \mathbf{A} has a corresponding *bitemporal domain*:

$$\mathbf{A}^{2T} = \{f \mid f : \mathbb{T}_t \times \mathbb{T}_v \rightarrow \mathbf{A}_\perp\} \quad (4.68)$$

More specifically, we define the valid time dimension to be the set of all times stretching from the beginning of time till the end of time (or infinity). The transaction time domain is more restrictive, since it spans from the the creation of the database and till the present time, usually denoted with the symbol *now*. As already indicated, we use the symbol $\mathbb{T}^2 = \mathbb{T}_t \times \mathbb{T}_v$ to denote the bitemporal domain.

To each function f from some temporal domain \mathbf{A}^{2T} we assign a *bitemporal lifespan* $LS_2(a_{2T})$, which can be any subset of \mathbb{T}^2 . Although [JCE⁺94] only provide a definition for bitemporal intervals, we use the same distinction as for the mono-temporal domain to distinguish between bitemporal intervals and lifespans. Thus, a *bitemporal interval* is a connected subset of \mathbb{T}^2 with sides parallel to the time axis, whereas a bitemporal lifespan is any subset of \mathbb{T}^2 . Both bitemporal intervals and lifespans can be finite or infinite sets of bitemporal instants.

The *bitemporal duration* of a bitemporal lifespan, denoted $|LS_2(a_{2T})|$, is the total area covered by the lifespan. However, many bitemporal lifespans are infinite, and the area of a bitemporal lifespan is maybe not of great importance. On the other hand, in many cases we are interested in the valid time projection of a bitemporal lifespan at some transaction time. We therefore define the function $LS_v : (\mathbf{A}^{2T} \times \mathbb{T}_t) \rightarrow P(\mathbb{T}_v)$ which for every transaction time in \mathbb{T}_t returns a lifespan in valid time $P(\mathbb{T}_v)$ of some function in \mathbf{A}^{2T} . Subsequently, $|LS_v(f, t_t)|$ is the function that returns the duration of the valid time lifespan of f at some transaction time t_t . A similar function $LS_t(f, t_v)$ can be identified for transaction time lifespans.

This leads us to the concept of valid and transaction *timeslices*. Given any set of functions \mathbf{A}^{2T} , the following two functions, called *timeslice projectors*, are defined:

$$\pi_v : \mathbf{A}^{2T} \times \mathbb{T}_t \rightarrow \mathbf{A}^{T_v} \quad (4.69)$$

$$\pi_t : \mathbf{A}^{2T} \times \mathbb{T}_v \rightarrow \mathbf{A}^{T_t} \quad (4.70)$$

These two functions return respectively a *valid timeslice* and a *transaction timeslice* of a bitemporal attribute respectively. In other words, given a fixed transaction time t'_t or a fixed valid time t'_v , we have

$$\pi_v(f, t'_t) = f(t'_t, t_v) \quad (4.71)$$

$$\pi_t(f, t'_v) = f(t_t, t'_v) \quad (4.72)$$

4.4.2 Functions over bitemporal attributes

For every function ω in Ω there is a corresponding function ω^{2T} in Ω^{2T} , usually with the same symbol, where Ω^{2T} is the set of all functions ω over bitemporal domains. So if,

$$\omega : \mathbf{A} \rightarrow \mathbf{B}, \quad (4.73)$$

then also

$$\omega^{2T} : \mathbf{A}^{2T} \rightarrow \mathbf{B}^{2T}. \quad (4.74)$$

As we did in Section 4.3.6, we can identify the following rules to construct products of a mixture of bitemporal and non-temporal domains. So, if $f : \mathbf{A} \times \mathbf{B} \rightarrow$

C, then

$$f : \mathbf{A}^{2T} \times \mathbf{B} \rightarrow \mathbf{C}^{2T} \quad (4.75)$$

$$f : \mathbf{A} \times \mathbf{B}^{2T} \rightarrow \mathbf{C}^{2T} \quad (4.76)$$

$$f : \mathbf{A}^{2T} \times \mathbf{B}^{2T} \rightarrow \mathbf{C}^{2T} \quad (4.77)$$

Then, what about $\mathbf{A}^{2T} \times \mathbf{B}^T$ or $\mathbf{A}^T \times \mathbf{B}^{2T}$? There are two solutions to this problem:

1. The bitemporal domain is projected onto a mono-temporal domain by making a timeslice with t_v (or t_t) set to *now*. Thus, the result of the function is a mono-temporal attribute. In other words,

$$f : \mathbf{A}^T \times \mathbf{B}^{2T} \rightarrow \mathbf{C}^T \quad (4.78)$$

$$f : \mathbf{A}^{2T} \times \mathbf{B}^T \rightarrow \mathbf{C}^T, \quad (4.79)$$

where \mathbf{C}^T can be either refer to the valid time or the transaction time dimension, depending on the application.

2. The mono-temporal domain may be expanded into a bitemporal domain by setting $t_t = t_v$ for all times in the mono-temporal domain.

$$f : \mathbf{A}^T \times \mathbf{B}^{2T} \rightarrow \mathbf{C}^{2T} \quad (4.80)$$

$$f : \mathbf{A}^{2T} \times \mathbf{B}^T \rightarrow \mathbf{C}^{2T} \quad (4.81)$$

However, the second alternative is not unproblematic if we try to duplicate a gradually changing attribute from valid time into transaction time. Notoriously, transactions are not gradual changes, and the transaction time domain should therefore be implemented with such a constraint. In addition, expanding temporal data into bitemporal data necessarily means that we add information which is not based on reality. It is therefore only the first alternative that provides a consistent solution to the problem.

4.4.3 Bitemporal Sets

As with temporal sets, there are at least three ways to express a bitemporal set using classic set theory. But formally, we define a bitemporal set as a *function over* \mathbb{T}^2 :

given an arbitrary domain \mathbf{A} and the time domains \mathbb{T}_t and \mathbb{T}_v , a bitemporal set β can be described as a function from \mathbb{T}^2 , into the power set $P(\mathbf{A})$:

$$\beta : \mathbb{T}^2 \rightarrow P(\mathbf{A}) \quad (4.82)$$

that for every pair of times in \mathbb{T}^2 returns a subset of \mathbf{A} containing the elements that are members of β at time $\langle t_t, t_v \rangle$.

In general, the operators defined over temporal sets may be extended to bitemporal sets, with the difference that all occurrences of the time is replaced with a pair of times.

4.5 Tropology

A mature time-based temporal data model must take all aspects of time into consideration. This also includes aspects of actions, processes and changes that occur in the modelled world. Since these are important matters, there has recently been an increasing interest on the *change* aspect of the real world. Another aspect of the temporal set theory, as presented so far, is that it is not equipped with a machinery to update the structure as time passes. Therefore, we need some kind of *evolution language* to describe changes in a temporal data set.

Because there is a need to study and understand the evolutionary aspects of the real world, we introduce the term *tropology* to denote this area of study. The word ‘tropology’ stems from the Greek word ‘tropos’ which means ‘to change’ or ‘to turn’. The tropology introduced in this section aims to fill the need to describe qualitative and quantitative aspects of changes, and to enable means for updating a temporal data set. In some ways, we view tropology to be akin to topology, but on the other hand, there is a subtle difference between them.

4.5.1 Defining Tropology

To give tropology a formal definition rooted in mathematics has been the most difficult task of the work presented in this article. One tempting way to define tropology is by defining some kind of vector space. In the Euclidean plane, a vector is the directed distance from one point to another. Similarly, we could define *change* as a vector from one state to another.

Formally, a *vector space* is a space \mathbf{V} which is equipped with the two functions *sum* and *scalar product*:

$$v_1 + v_2 : \mathbf{V} \times \mathbf{V} \rightarrow \mathbf{V} \quad (4.83)$$

$$c \cdot v : \mathbb{R} \times \mathbf{V} \rightarrow \mathbf{V} \quad (4.84)$$

such that a number of properties, such as associative and distributive properties, are satisfied. However, we get problems with such a definition of tropology since not all changes are ‘scalable’, and it might also be that two changes may not give the same result if they were applied in a different order.

Instead, we think that tropology should be defined in a similar way as topology is defined. Given any set \mathbf{A} the tropology must be defined for the space spanned by the Cartesian product $\mathbf{A} \times \mathbb{T}$.

Since we did not have the time to review the preciseness and conciseness of the definition, it should only be considered to be a preliminary definition: Let \mathbf{A} be a set containing versions of a group of objects. To this set, there is a function $LS : \mathbf{A} \rightarrow I(\mathbb{T})$, that to every object a in \mathbf{A} assigns a time interval which denotes the time during which a is current. The *tropology* on \mathbf{A} is then defined as the relation \mathcal{R} on \mathbf{A} such that:

1. for every pair $\langle a, b \rangle$ in \mathcal{R} , the following condition holds

$$\overline{LS(a)} \leq \underline{LS(b)} \quad (4.85)$$

2. for every two pairs $\langle a, b \rangle$ and $\langle b, c \rangle$ in \mathcal{R} , then $\langle a, c \rangle$ is not in \mathcal{R} .

It is thereby clear that the tropology \mathcal{R} is irreflexive, asymmetric and non-transitive, and can therefore be represented as a Hasse-diagram.

In general, each pair $\langle a, b \rangle$ represent a passage from one state to another state indicated by the objects versions a and b , respectively. This passage is generally referred to as a *change*. Each change have a lifespan which denotes the time during which the change occurred, and is defined by

$$LS(\langle a, b \rangle) = [\overline{LS(a)}, \underline{LS(b)}]. \quad (4.86)$$

This definition of tropology conforms to a great extent with Frank’s discussion of qualitative reasoning with partially ordered time models [Fra94]. It is a question, of course, whether the tropology should be defined to conform with Frank’s discussion of the *semi-ordered* time model, but a semi-ordered tropology is not discussed in this article.

4.5.2 Related Work

Researchers have studied changes in temporal data sets for some time, but the term ‘tropolgy’ has never been used. Some of the most interesting sources that can be related to tropology are found in the literature of linguistics and artificial intelligence. McCarthy’s *situation calculus* is one attractive model [MH69]. In this theory, a *situation* represents a complete state of the universe at an instant of time, and the set S is defined to contain all situations, including imaginary situations. Then all actions or changes are modelled as *fluents* which are functions from S into S , or into the binary set \mathbb{B} . We have also seen some extension to this calculus that handles continuous change (e.g. [Sha90]).

One such extension is Allen’s general theory of action of time [All84], which is partly based on [Mou78]. His theory describes the world by a set of temporally qualified assertions describing knowledge about past, present and future time. The static aspects of the world are captured by *properties* whereas the dynamic aspects of the world are captured by *occurrences*. Occurrences can again, according to [Mou78], be divided into *events* which focus on performances and achievements, and processes which focus on continuous processes. An event is an accomplishment that occurs over some time interval. If an event occurred over a time interval X , it could not occur during any subinterval of X because the accomplishment was not fulfilled over that time interval. A process, on the other hand, does not focus on the accomplishment. A process that is occurring during a time interval X is also defined to occur during any subinterval of X . Allen’s theory is supported by the tense operators described in Section 4.3.1 along with the set of relationships between time intervals [All83] already introduced in Figure 4.1 (Section 4.2.2).

What distinguishes the research described above from what we have seen particularly in the field of temporal databases, is the focus on the underlying processes and the actions that cause changes in the real world. It has already been recognized that temporal databases can be constructed by only representing changes or events. Such data models have therefore been named *event-oriented* data models. However, we assume a model where both the changes and states are explicitly represented and described, and that known relationships between them are given.

Interestingly, in recent research of temporal GIS we have seen an increasing interest in these matters. Yuan [Yua98] states that such an approach is necessary to enable temporal GIS to efficiently handle queries about static and dynamic aspects of the universe of discourse (UoD). One of the earlier models for spatiotemporal

models is the Triad model where spatiotemporal information is viewed from where (spatial), when (time), and what (thematic) perspectives [Peu94]. A fourth corner can be added to represent the objects which are characterized by all those three perspectives and then [CM98] adds the fifth corner, processes, which can be related to all the four other corners. If we are to represent changes explicitly in the database, we need a taxonomy of changes or processes. In general, we would like to have an explicit representation of all actions and processes of interest to our UoD. Not all of these result in changes to the UoD, but it is on such changes where the focus is in the current section.

In recent literature, several taxonomies of changes have been presented. Claramunt and Thériault [CT95] distinguishes between three types of changes or processes: The first type of changes includes those that concern the evolution of a single entity. These include basic changes as appearance, disappearance and transformations. The second type of changes includes those that involve functional relationships between several entities. These include replacement processes such as succession and permutation, and diffusion processes such as production, reproduction and transmission. The third type of changes includes those that are the evolution of spatial structures involving several entities. These include restructuring processes such as split, merge and reallocation; for example of polygons in a polygon mesh.

In [Ren96], a conceptual modelling language where states and transitions between states are linked together in a directed acyclic graph is proposed. In this language the duration of both states and transitions has also been taken into consideration. This way, seven basic types of transitions between objects have been identified. These are birth, death, alteration, reincarnation, split, merge and reallocation. A more comprehensive modelling formalism has been presented in [HE97], and the set of different change types has been further refined (although the changes are not considered as explicit entities). A similar taxonomy has been presented for Voronoi diagrams [MAGM98]. The interesting aspect of the research above is that it defines changes in a similar way to McCarthy and Hayes [MH69], as functions from one set of states into another. Each change takes a set of objects as input, and give another set of objects as output.

Changes in topological relationships among objects are also of interest to topology. Egenhofer and Al-Taha [EA92] used the 9-intersection method to study changes in topological relationships between gradually changing regions. The eight different topological relationships between pairs of regions are set up in a

closest-topological-relationship-graph and the different paths in this graph have been identified for different types of changes between the two regions. This theory has also been extend to include interrelationships among points, lines and regions [VR96].

Another aspect of topology is the study of quantitative aspects of changes. For continuous real functions over time, it is easy to obtain and exploit first and second derivates as well as integrals of such functions. It should also be possible to make representations of the derivate of a gradually changing region. However, such changes can be quite difficult to quantify. This problem have been studied in [Gal97] and three proposed metrics to measure the separation (or difference) between two regions, boundary separation, size separation and interior separation were studied. However, none of these proposals satisfy the conditions of a metric, and some of them will even fail to give a sensible result in certain situations, e.g. the boundary separation will not be able to measure the separation between the northern and southern hemisphere of the globe.

Another problem is the combination of continuous changes with instantaneous changes, e.g. if a peninsula is gradually eroded by the sea, it may eventually become an island. Although the erosion is a continuous process in every respect, the topological change that occurred when the sea broke through, is certainly a discontinuous one.

4.5.3 Valid Time Tropology

In order to describe changes we need some kind of *change description language*. Such a language can be used to describe previous changes in a data set as well as to update a data set as time evolves. According to [RS93b], we introduce three types of change operators or *tropological operators*: Let a and b be two objects with respect to some universe of discourse.

- The *constructor* of an object a is an unary operation that creates a new object, and is denoted \vdash . The expression $\vdash a$ is read ‘the creation of a ’.
- The *modifier* from one object a to an object b is a binary operator that establish a predecessor-successor relationship between two groups of objects, and is denoted \mapsto . The expression $a \mapsto b$ is read ‘ a becomes b ’.

- The *destructor* of an object a is a unary operator that destroys the object a , and is denoted \dashv . The expression $\dashv a$ is read ‘the destruction of a ’.

Additionally, for each object we may assign a value. We use the notation $a:x$ to indicate that a has the value x . In other words, if a is a function over time and has the value x at time t , then $a(t) = x$. However, we write $a:x$ because it is not so important at exactly which time a has the value x .

Related to these tropological operators and the lifespans of these which are temporal intervals, a certain number of assertions must hold. In order to express these axioms, we need the *meets* relation, indicated by the symbol ‘m’, between two temporal intervals described in Section 4.2.2 ([All83]). It is now clear that, according to [All84], the following four assertions must hold for the operators given above.

$$LS(a) \text{ m } LS(a \mapsto b) \quad (4.87)$$

$$LS(a \mapsto b) \text{ m } LS(b) \quad (4.88)$$

$$LS(\vdash a) \text{ m } LS(a) \quad (4.89)$$

$$LS(a) \text{ m } LS(\dashv a) \quad (4.90)$$

We can now use these operators to express different types of changes. In [HE97], a set of 20 different change types were identified, but it is obvious that more exist. Related to simple functions over time, we can identify at least six different types of changes:

- *Create* — An object has been created and assigned the value x .

$$\vdash a:x \quad (4.91)$$

- *Alter* — An object has changed from value x to value y .

$$a:x \mapsto a:y \text{ or } a:(x \mapsto y). \quad (4.92)$$

- *Destroy* — An object has been destroyed.

$$\dashv a \text{ or } a:(x \mapsto \perp). \quad (4.93)$$

- *Reincarnation* — An object that has been destroyed earlier, is recreated.

$$a:(\perp \mapsto x) \quad (4.94)$$

- *Metamorphose* — An object changes type and appears with new identity and (possibly) a new value.

$$\neg a:x \mapsto \vdash b:y \quad (4.95)$$

- *Duplicate* — An object has been duplicated.

$$a:x \mapsto \vdash b:x + a:x \quad (4.96)$$

Related to sets, we may have a lot of different change types, many types involve more than one set, e.g. the annex operation would dissolve one set and include the members into another set. Some of the most typical types of changes amongst sets are listed below:

- *Aggregate* — An arbitrary number of objects are gathered together to form a new set, e.g. Belgium, Netherlands and Luxembourg form Benelux.

$$a + b + c \mapsto \vdash \mathbf{A}:\{a, b, c\} \quad (4.97)$$

- *Dissolve* — A set is destroyed, but the members are not, e.g. the prime minister dissolves the parliament.

$$\neg \mathbf{A}:\{a, b, c\} \mapsto a + b + c \quad (4.98)$$

- *Include* — A new member is included into the set, e.g. Sweden is taken up as a new member of the European Union (EU).

$$a + \mathbf{A}:\{b, c\} \mapsto \mathbf{A}:\{a, b, c\} \quad (4.99)$$

- *Secede* — A member secedes from the set, e.g. a politician secedes from a political party.

$$\mathbf{A}:\{a, b, c\} \mapsto \mathbf{A}:\{a, b\} + c \quad (4.100)$$

- *Empty* — All the members in a set are seceded from the set. The set itself is not destroyed, e.g. a bottle of water is poured out.

$$\mathbf{A}:\{a, b, c\} \mapsto \emptyset \quad (4.101)$$

- *Split* — A set is destroyed, and the members of the old set form one or more new sets, e.g. Czechoslovakia splits to form Czech Republic and Slovakia.

$$\neg \mathbf{A}:\{a, b, c, d\} \mapsto \vdash \mathbf{B}:\{a, b\} + \vdash \mathbf{C}:\{c, d\} \quad (4.102)$$

- *Merge* — Two or more sets are destroyed, and the members of the old sets form a new set, e.g. East and West Germany merge to form Germany.

$$\neg \mathbf{A}:\{a, b\} + \neg \mathbf{B}:\{c, d\} \mapsto \vdash \mathbf{C}:\{a, b, c, d\} \quad (4.103)$$

- *Deduct* — A subset of a set, secedes from one set and form a new set, e.g. the Baltic Countries are deducted from the Soviet Union.

$$\mathbf{A}:\{a, b, c, d\} \mapsto \mathbf{A}:\{a, b\} + \vdash \mathbf{B}:\{c, d\} \quad (4.104)$$

- *Annex* — A set is destroyed, and the members are included into another set, e.g. Iraq annexes Kuwait.

$$\mathbf{A}:\{a, b\} + \neg \mathbf{B}:\{c, d\} \mapsto \mathbf{A}:\{a, b, c, d\} \quad (4.105)$$

- *Move*: $\mathbf{A}:\{a, b, c\} + \mathbf{B}:\{d, e\} \mapsto \mathbf{A}:\{a, b\} + \mathbf{B}:\{c, d, e\}$ — Some members of one set, secede from this set and become members of another set, e.g. Israel takes the western bank from Jordan.

- *Copy out* — Some members of one set, form a new set, but do not secede their membership from the old set, e.g. some members of the EU form the European Monetary Union (EMU).

$$\mathbf{A}:\{a, b, c\} \mapsto \vdash \mathbf{B}:\{a, b\} + \mathbf{A}:\{a, b, c\} \quad (4.106)$$

- *Copy over*: — Some members of one sets also become members of another set, e.g. some members of the EU joins the EMU at a later time.

$$\mathbf{A}:\{a, b, c\} + \mathbf{B}:\{d, e\} \mapsto \mathbf{A}:\{a, b, c\} + \mathbf{B}:\{b, c, d, e\} \quad (4.107)$$

As can be seen above, many of these changes involve multiple creations and destructions of sets (or even members). It is important to notice that all the operations of one individual change have the same lifespan as the total change, e.g. given an annex operation $\mathbf{A} + \neg\mathbf{B} \mapsto \mathbf{A}$ we must assert that

$$LS(\neg\mathbf{B}) = LS(\mathbf{A} + \neg\mathbf{B} \mapsto \mathbf{A}) \quad (4.108)$$

4.5.4 Causal Relationships

In many cases, we are interested in preserving causal relationships in a temporal dataset, provided that these are known. We therefore introduce the causal operator \rightsquigarrow as a new tropological operator. In fact, the causal operator takes, according to [All84], two forms:

- *Agent causality*, denoted $\overset{a}{\rightsquigarrow}$, means that some change was caused by some object or agent (using AI terminology), e.g. if the object a makes a modification to object b such that b , having the value x gets the new value of y , we may write

$$a \overset{a}{\rightsquigarrow} b:(x \mapsto y). \quad (4.109)$$

- *event cause*, denoted $\overset{e}{\rightsquigarrow}$, means that one change was caused by some other change or event, e.g. if the modification of an object a from value x to value y caused object b to change from value v to value w , we may write

$$a:(x \mapsto y) \overset{e}{\rightsquigarrow} b:(v \mapsto w). \quad (4.110)$$

According to [All84], it is clear that a certain set of assertions must hold for these operators. For agent causality, if $a \overset{a}{\rightsquigarrow} C$, then

$$\underline{LS(C)} \in LS(a), \quad (4.111)$$

which means that if the object a caused some change, the change must start during the lifespan of a . Note that, according to this axiom, the change may continue to occur even after the death of a . With event-causality on the other hand, we must require that the change cannot cause another change prior to its occurrence [All84]:

$$\underline{LS(C_1)} \leq \underline{LS(C_2)} \quad (4.112)$$

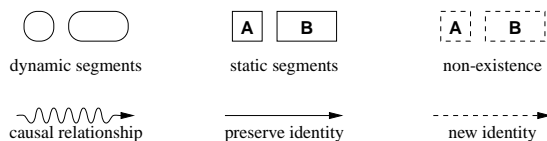


Figure 4.2: The extended history graph notation

A question is whether the change C_2 may have to begin before the change C_1 ended, but according to Allen [All84], it may not be so. In other words, we may allow a potential change to be ‘latent’ in the system before it actually occurs, e.g. the building of a new road may cause a new petrol station to pop up some time after the construction of the road finished.

Such latent changes cannot be caused by agents. Instead, this should be modelled as a caused some change C_1 during its lifespan, which again caused another change C_2 to occur some time after C_1 .

4.5.5 Conceptual Modelling of Tropology

In literature, a few conceptual models have been suggested to describe tropology, such as [Ren96] and [HE97] and also to some extent [CT95]. Whereas [HE97] and [CT95] focus on qualitative aspects of temporal relationships between objects or versions of objects, [Ren96] is also concerned about some durational aspects, and also models the change as a separate type of object. In the following, we suggest a new modelling language called the *extended history graph* model (EHG) which combines the notations of [HE97] and [Ren96].

Given a function over time, we can identify at least three types of ‘states’ of that function which may hold over intervals of time. If f is a function over time, f is *dead* over the interval \mathbf{X} if $f(t) = \perp$ for all t in \mathbf{X} , it is *static* over the interval \mathbf{X} if $f(t)$ has the same value for all t in \mathbf{X} and $|\mathbf{X}| > 0$, and it is *changing* otherwise. A function is *alive* if it is not dead (i.e. either changing or static). Subsequently, it is possible to implement a function over time by a sequence of alternating static and dynamic segments, in addition to null-segments that identifies time intervals during which the function is undefined.

In the EHG-model, which is illustrated in Figure 4.2, static segments are displayed as rectangular boxes, while dynamic segments are displayed as circles or round-ended boxes. A null segment (previously introduced as \perp) is displayed as a

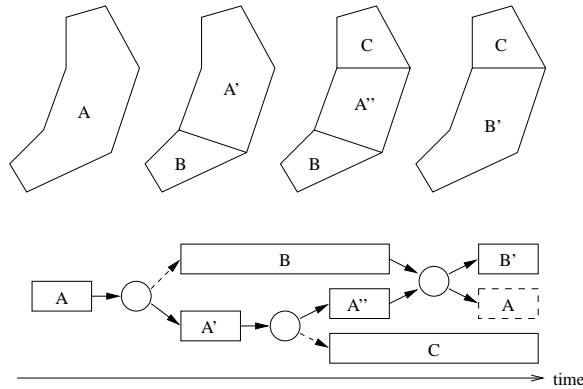


Figure 4.3: A sample Extended History Graph

rectangular box with a dashed outline. Letters are applied to the boxes to denote identity, and primes (or subscripts) are used to denote different versions within the same identity. The boxes are stretched along the horizontal time dimension in order to mimic the lifespan of the segments.

In some cases we may need to give an intermediate state of a continuously changing object. This can be done by a static segment whose lifespan has no duration, and is indicated by a very narrow rectangle. Instantaneous changes on the other hand, are modelled as dynamic segments whose lifespan has no duration. These types of changes are indicated by circles. In addition, arrows provide connections between dynamic and static segments. A solid arrow is used to denote persistence of identity, while a dashed arrow is used to denote the creation of a new identity. A wavy arrow is used to indicate causal relationships. A wavy arrow from a dynamic segment to another dynamic segment indicates event causality, whereas a causal relationship from a static segment to a dynamic segment indicates agent causality.

Figure 4.3 shows an sample EHG of a story where a set of spatial regions has been split and merged. The example shows a region A that *deducts* a subset of itself to form a new object B . The remainder of A is denoted A' . In the next change, A' deducts another part to the new object C and becomes A'' . In the last change, B annexes A'' to become B' whereby A ends its life.

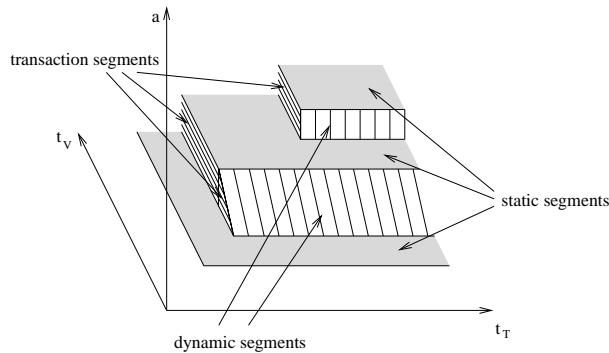


Figure 4.4: The mapping of an attribute into the bitemporal space

4.5.6 Bitemporal Tropology

The valid time tropology and transaction time tropology are in principle of a similar nature. The main difference is that an attribute can only change gradually over valid time, while transactions are regarded as instantaneous changes. Although, transactions may have duration and certain transactions may even last over quite some time, we adopt the view that any transaction can be associated with one single instant: the *commit-time*.

However, the bitemporal tropology is much more complex to deal with. If we assume an attribute as a function over bitemporal space as illustrated in Figure 4.4, we can identify at least three types of segments (in addition to the null-segment) of such a function. These are as follows:

- *static segments* — represent the values or states of a bitemporal attribute.
- *dynamic segments* — represent the valid time transitions from one value or state to another value or state of the attribute.
- *transaction elements* — represent the transactions associated with an attribute.

A static segment of some domain \mathbf{A} is an element of the set $\mathbf{A} \times P(\mathbb{T}^2)$, while dynamic segments and transaction segments are elements of the set $\mathbf{A}^2 \times P(\mathbb{T}^2)$.

The bitemporal lifespan of a segment e is denoted $LS_2(e)$, and is restricted to be a bitemporal interval. The bitemporal duration of the segment is denoted

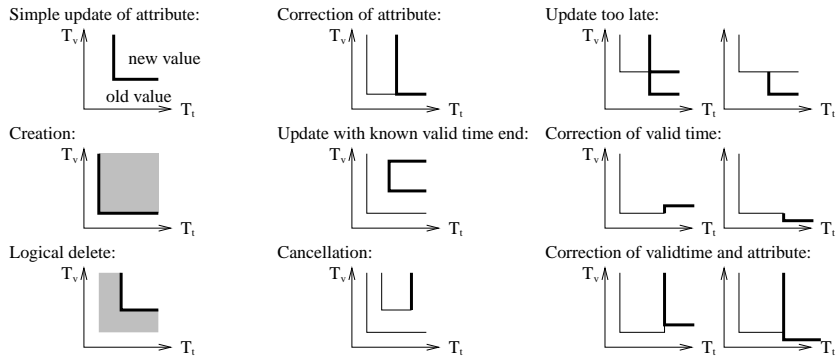


Figure 4.5: Transaction types

$|LS_2(e)|$ and represents the total area of the segment in the \mathbb{T}^2 space. For all segments we have $|LS_2(e)| \geq 0$, which allows us define segments that have no duration, e.g. transaction segments always have zero duration since they have no duration over transaction time.

4.5.7 Transaction Types

By studying various permutations of parameters needed in an update, a set of different transaction types can be identified. Generally we may have *updates*, *corrections* and *validations*. Corrections may apply to all these three types of transactions, that is correction of updates, correction of corrections and corrections of validations. A special type of corrections is *cancellation* which cancel previous updates. Validations are important types of transactions, e.g. if an object has not been updated for a long time, it might give the impression that the object has not been checked and verified since the time of change, even if newer sources have validated the old information [Lan93].

Using valid time-transaction time diagrams, a set of distinct transaction types has been identified. In such diagrams, the valid time dimension is the vertical axis, whereas the transaction time dimension is the horizontal axis. The lines separate different lifespans of the segments, and each diagram can only be associated with one attribute or object. The lines that are created by the different transactions are emphasized with a fat line.

Figure 4.4 shows a 3-D version of this diagram where the attribute dimension

is displayed as the third axis. If we name the attribute a and let its values range from the real domain \mathbb{R} such that the three values of the attribute displayed in the figure are 1.0, 2.0 and 3.0 respectively. Then the following three transactions on a are displayed:

- *Transaction 1 at tt_1* : The attribute was created at valid time vt_1 with a value of 1.0.
- *Transaction 2 at tt_2* : The attribute changed gradually from 1.0 to 2.0 during the valid time period from vt_2 to vt_3 .
- *Transaction 3 at tt_3* : The attribute changed instantaneously from 2.0 to 3.0 at valid time vt_4 .

Using such diagrams, we can identify and visualize a number of transaction types, as illustrated in Figure 4.5:

- *Update of attribute* — this is the simplest of all transactions which means to change one or more attributes of an object to reflect some real world change.
- *Update too late* — two updates on the same attribute (right) or different attributes of the same object (left) are made in wrong order.
- *Update with known valid time end* — an update where we know for how long the new data will be valid.
- *Creation* — add a new entity to the database.
- *Logical delete* — an entity has been destroyed.
- *Cancellation* — an earlier update was completely wrong, as e.g. updating the wrong entity. Cancellations are a special type of corrections.
- *Correction of attribute* — correct an earlier update where one or more attributes were incorrectly entered.
- *Correction of valid time* — correct an earlier update where the valid time was incorrectly entered.

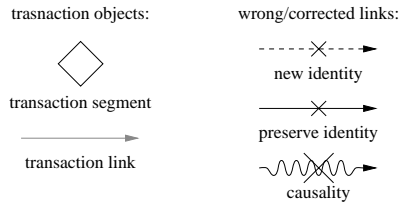


Figure 4.6: The bitemporal constructs of the Extended History Graph model

- *Correction of attribute and valid time* — correct an earlier update where both the valid time and at least one attribute were incorrectly entered.

Combined with the different types of changes described in Section 4.5.3, we get a complex set of different change-transaction types.

4.5.8 Conceptual Modelling of Bitemporal Tropology

For simple functions over bitemporal space, the bitemporal tropology can be displayed in form of transaction-time / valid-time diagrams as explained above. To capture the relationships between the different segments of a bitemporal attribute, in addition to more complex change types involving more objects, five new symbols are introduced to the Extended History Graph notation. These are, as shown in Figure 4.6, the *transaction segment* which is displayed as a diamond, the *transaction link* which links the inputs and the outputs of a transaction which is displayed as a grey arrow, and three types of cancelled links. The transaction link connects the dynamic and static segments involved in a transaction. The cancelled links are indicated with arrows that have been ‘crossed out’ by an ‘x’.

An example of the bitemporal constructs of the extended history graph language is provided in Figure 4.7. This figure shows three transactions on the attribute a and three changes:

- *Transaction 1* at tt_1 : a is created at valid time vt_1 with the value of A .
- *Transaction 2* at tt_2 : a has changed to the new value of B at valid time vt_3 .
- *Transaction 3* at tt_3 : new information is added that a actually had the value C from valid time vt_2 until it changed to the value B at valid time vt_3 .

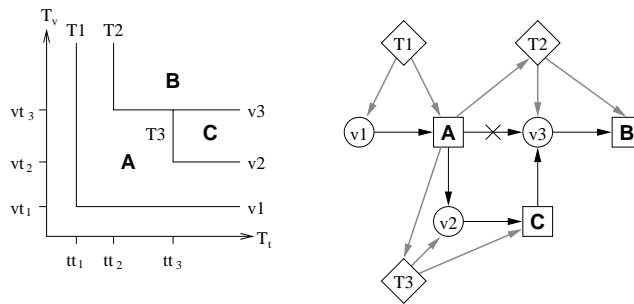


Figure 4.7: A Sample Extended History Graph schema

The link from the static segment *A* to the dynamic segment *v3* is therefore crossed out since that passage never really occurred, but was a result of an update in wrong order.

4.6 *Implementing the Approach Using Abstract Data Types*

4.6.1 *Background and Related Work*

The temporal relational model represents the simplest way of implementing a temporal database. In this model, each tuple is timestamped with the time interval during which the tuple was valid [JS96]. This tuple-level versioning method is of course sufficient in many applications, but the method also has some drawbacks. First, the history of one logical entity is spread over many tuples in the relation, and there are no links that connect successive versions of the same entity together. Second, the method has no inherent notion of change, unless the database designer creates specific relations for that purpose. And third, it is not well suited to handle data that is continuously changing.

In general, there have been few proposed models that express data in form of functions over time. The main problem is maybe that such an approach fits in badly with the relational database model. The object-oriented model on the other hand, is much more suited for such an approach, whichever [WD92] demonstrates.

In fact, data can be expressed as a function over any parameter, such as time, space, people and map scale [Gad93b] [Mis93] [BS93]. However, two reports are

of particular interest to the current research, viz. the work of Erwig et al. [EGSV97] [ESG97]. In their model, real valued functions over time, called *moving reals*, as well as *moving points* and *moving regions* are studied. They suggest that such functions could be implemented by linear approximations using data points. They prove that this model can help strengthen the expressive power of queries, e.g. suppose that we need to determine at which time two points were closest to each other. This query can be answered by first determining the distance between the two points as a function over time, and then find the minimum value of this function.

The drawback of this method is that if two adjacent regions share a common boundary, the common boundary is duplicated in each region. This is both redundant and awkward to handle if the common boundary is moved, and thereby changing both regions. One of the earliest spatiotemporal data models proposed is the *space-time composite* model by Langran and Chrisman [LC88]. In this simple model, the spatial data is kept in a traditional polygonal data structure, also known as a *polygon mesh*, and each polygon is associated with one attribute history. The drawback of this method is that the number of polygons increases exponentially with time (whichever Saafeld's work indicates [Saa91]) and that two adjacent polygons may have partially common histories because they either belong to the same logical entity at present, or at some past time.

A few other models have been described by Worboys [Wor92] [Wor94b] and [Wor94a]. In [Wor94a], spatiotemporal objects are modelled using so-called *ST-complexes* which are sets of *ST-simplexes*. Each ST-simplex is an ordered pair $\langle S, T \rangle$, where S is a simplex and T is a bitemporal lifespan. We have also seen some other models, such as the amendment vector model, the time-slice model [LC88] (the latter applicable to both raster and vector data) and the event-oriented raster model, which is a raster model where only the changes between each time-slice is represented [PD95].

In general, the problem is to link a base model to abstract objects where the location of the object can be returned as a function over time. In this section we propose a spatiotemporal model, which, in contrast to the other models, also can handle gradual change. In addition, the topological as well as the tropological relationships are explicitly embedded in the model.

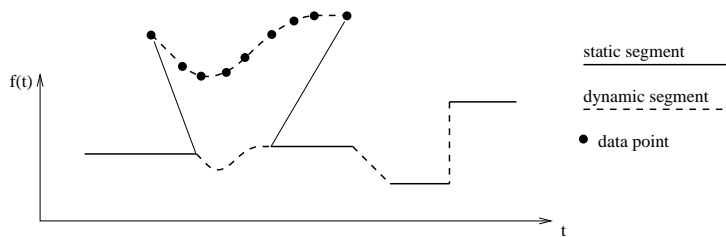


Figure 4.8: Dynamic and static segments of a function

4.6.2 Implementing Temporal Attributes

There are many ways of implementing a temporal attribute. Each method suit different types of attributes. We distinguish between four types of attributes, each of which would benefit from a different implementation:

- *continuously changing attributes*, which are always in a dynamic state, e.g. temperature.
- *discretely changing attributes*, which are always in a static state, and where changes are always instantaneous, e.g. bank accounts.
- attributes that are continuously changing in some periods, while static in some other periods, e.g. the position of a ship.
- *time series data* which is data that accumulates over some time interval, e.g. daily precipitation.

As indicated in Section 4.5.5, we can divide a function over time into segments such that three different segment types can be identified. These are static segments, dynamic segments and null-segments.

Figure 4.8 illustrates this. For the times when the function is not changing, it is implemented by one static segment. For times when the function is in continuous change, it may be implemented by a sequence of static segments with no duration called *data points* which are linked together with dynamic segments. Dynamic segments are also used to represent instantaneous changes. This way of implementing functions is slightly different from [EGSV97], who implement functions only using data points together with a flag that determines whether the point represent a gradual or instantaneous change since the previous data point.

The advantage of this model is that the dynamic segments provide the tropological relationships between states of objects, and that, since these are explicitly given in the data structure, it is possible to assign attributes to the changes that are not inherently a part of the object that they associate; e.g. if a building is built, it is possible to give information about who was the entrepreneur, and how much it cost to build the house.

Static segments are identified by an element of some domain A and a time interval in $I(\mathbb{T})$. Dynamic segments on the other hand are identified by two elements in A , one for the initial state and one for the end state, together with two corresponding times in \mathbb{T} . Both dynamic and static segments may have a lifespan with zero duration.

This way of implementing a temporal attribute can, in principle, be used to implement all the first three types of attribute behaviour. However, in many cases we are not interested in the dynamic segments, e.g. in temperature measurements. It therefore makes sense to implement such attributes using a simple sequence of data points, combined with linear or spline interpolation or approximation between data points.

4.6.3 *Implementing Temporal Sets and Sequences*

In the temporal relational model, the membership of a record in a relation equals the lifespan of that record. If the membership of some entity in another set is to be modelled, this has to be done in an additional relation that contains only the key attribute and the lifespan of that record; e.g. Steiner [SN97b] proposes an object-oriented model where each object is timestamped, and the properties of an object is kept in a so-called *time-collection* of other objects. The membership in the time collection and the lifespan of each member are different, which allows an object to have several roles, i.e. be member of several time collections at the same time. Thus, each collection contains entries of the type $\langle oid, ls \rangle$, where *oid* is the object identifier and *ls* is the membership lifespan of the object.

Often, the order of the members in a set is of vital importance. Totally ordered sets are in computer programming often implemented using array structures. We call them *sequences* and regard them as sets to which we have assigned a total order. Implementing a *temporal sequence* is not a trivial task, e.g. if two entries in a sequence are swapped, then how is this to be implemented if the order-history is to be preserved. A similar problem occurs if a new entry is inserted and the entries

following the new entry have to be shifted up one index.

However, we believe that the simplest method is also the best. For each version of the temporal sequence, a complete sequence of object-identifiers is stored. Thus, for each change to the temporal sequence, a complete new sequence of *oid*'s has to be made. The entries themselves are not stored in this sequence, only the object identifiers. This way, if the entries happen to be temporal objects, then the history of these objects are independent from their *index-history* in the sequence.

4.6.4 Implementing Spatiotemporal Attributes

Implementing functions over time using sequences of static and dynamic segments is particularly well suited for spatiotemporal objects. In this article, we restrict ourselves to a two-dimensional Euclidean plane, \mathbb{E}^2 . We define three types of *spatial atoms*:

- *point* — represents a 0-dimensional element. We define $\mathbb{S}_p = \mathbb{E}^2$ to be the set of all points.
- *curve* — represents a 1-dimensional element c which is a continuous mapping $c : \mathbb{R} \rightarrow \mathbb{E}^2$. We define \mathbb{S}_c to be the set of all curves in \mathbb{E}^2 .
- *region* — represents a 2-dimensional element r which is subset of \mathbb{E}^2 , having at least one interior point. We may identify two types of regions, simple regions which are connected subsets of \mathbb{E}^2 , and complex regions which are not connected. We define \mathbb{S}_r to be the set of all regions.

Then, the set of all spatial atoms \mathbb{S} is defined as the superset

$$\mathbb{S} = \mathbb{S}_p \cup \mathbb{S}_c \cup \mathbb{S}_r, \quad (4.113)$$

and the corresponding set of *spatiotemporal atoms* becomes \mathbb{S}^T .

Worboys [Wor92] describes a system of so-called *ST-atoms* where each ST-atom consists of one spatial extent associated with a time interval. Since this types of ST-atoms cannot represent spatial objects that have a gradually changing extent, they correspond to the static segments of a spatiotemporal attribute. Thus we may call them *static ST-segments*. Then, we also need to define a set of dynamic segments of spatiotemporal attributes, called *dynamic ST-segments*. This gives us a system of six types of spatiotemporal segments as illustrated in Figure 4.9.

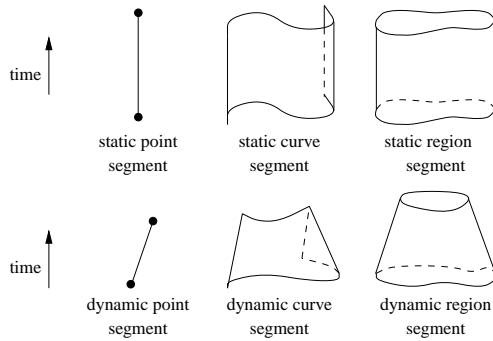


Figure 4.9: Spatiotemporal segments

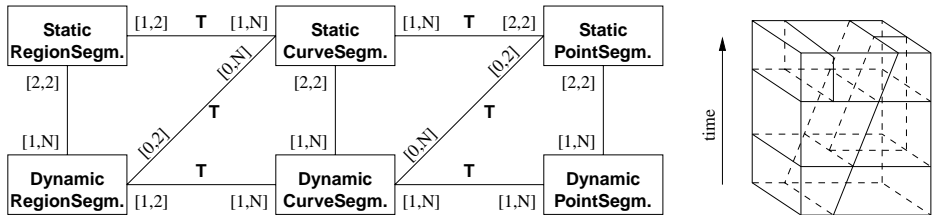


Figure 4.10: An ER-model of a polygon mesh

The task is now reduced to develop a temporal polygon mesh that incorporates the six types of spatiotemporal segments in Figure 4.9. Figure 4.10 shows an entity-relationship model of these segments and their relationships. Here, the entities (i.e. the segment types) are shown as boxes and the relationships are shown as lines between the boxes. The vertical relationships in the model are predecessor/successor relationships between the segments. The horizontal and diagonal relationships are topological. These relationships are also *temporal relationships* and are therefore marked with a ‘T’. That means that the relation between the two entities must be a strict temporal relation, since the relation may exist only during a subset of the time at which both entities co-exist.

All segments are associated with a bounding box in the space-time cube that can be used to build indexes, e.g. using R-trees. The time extent of this bounding box also represent the lifespan of each segment.

4.7 *Concluding Remarks*

We have introduced and discussed the temporal set-theory, and we have demonstrated its application in building temporal abstract data types. We have showed how to construct non-spatial attributes, spatial attributes and collections or sets as functions over time. We have also looked into the bitemporal domain, and we have looked into the new research area that now seem to emerge from research in temporal GIS, which we have called *tropology*.

We believe that, due to the wide range of applications of temporal GIS, we must strive for a common and comprehensive foundation and a general model for all these applications. The temporal set-theory provides a suitable framework for this. Hence, we can achieve many of the goals within GIS research today.

ON THE DESIGN OF AN OBJECT-ORIENTED TEMPORAL GIS

5.1 Introduction

Before computers, information about the geographical world was stored in the form of paper maps. However, current geographical information systems (GIS) are still representing spatial information utilizing this traditional cartographic view. Thus, real world objects are represented through cartographic primitives such as points, lines and areas, or gridded areal units. These primitives are then organized within a set of layers, where each layer represent one specific theme, such as roads and vegetation. Although the topological relationships between objects in this model were explicitly embedded in the data structures, the information that these data models can convey is still limited to the features that can be depicted in graphical form.

After the boost of the object-oriented paradigm in programming languages (such as C++) and the following success of implementing these concepts in computer aided design (CAD), graphical user interfaces and office information systems, the object-oriented paradigm stood out as a promising approach to the implementation of the next generation GIS. But, the object-oriented paradigm merely became a wrapper around the cartographic model.

In general, cartographic data structures have been implemented with little consideration to how the information is perceived at the conceptual level. Moreover, terms like *object-based* GIS and *feature-based* GIS are frequently used in the literature, all with similar definitions.

In this work on object-oriented modelling for *temporal* GIS, we adopt the view

that the real world consists of objects and that these objects can be structured according to the *basic object model* described in Section 5.2. The reason for our enthusiasm for the object-oriented model is that it allows us to encapsulate an entity and its entire history into one single object [KRS90]. Moreover, it allows us to model geographical and non-geographical objects together with temporal and non-temporal objects in a uniform way. The purpose of this article is to show how to utilize the object-oriented model to model geographical objects over time, and to show that the object-oriented paradigm provides an expressive and flexible framework in that respect.

An important part of any information system is the database. Temporal databases have been an active research field in computer science since the early 1980s. Because of the dominance of the relational model, it was natural to choose this model as a basis for a temporal database model, viz. the *temporal relational model*. Today, this model and its variants have been described vigorously in the literature [Tan87] [CC87] [T⁺93] [JS95] [JS96] [Sno92]. Recently, object-oriented temporal databases have also received increasing interest. A few models have been presented, see [KRS90] [FSG92] [WD92] [SSP95] [BFG96] [SN97b]. Additionally, a significant body of research has been dedicated to the problem of object and schema evolution in object-oriented databases [Bra93] [MS93] [Lau97] [LSW97]. This research is important to temporal databases and can, with only minor adaptations, be directly incorporated into an object-oriented temporal database management system. Therefore, we have also seen a growing interest in schema evolution for temporal databases [KCLS95] [DGS95] [DGS97].

Also within GIS, a substantial body of research concerning temporal representations has emerged, see [Lan92] [AB90] [Peu94] [Fra94], [Wor94a]. A bibliography of this work can be found in [ASS94]. However, this work to a large extent still utilizes the cartographic view described above. The objective of this work is therefore to provide an object-oriented model that describes the real world rather than a cartographic map thereof.

The remainder of this article is organized as follows: In the next section we describe the basic object model, and in Section 5.3 we review relevant work. In Section 5.4 we present the temporal object model, and in Section 5.5 we illustrate the power of the model by presenting an example. In Section 5.6 we illustrate the utility of queries on the model, and finally, in Section 5.7, we sum up and indicate directions for future research.

5.2 The Basic Object Model

In the literature, several different variants of the object-oriented paradigm exist. In this section we provide a description of the *basic object model* as it will be used here.

An *object* is considered to be an abstract representation of some real world entity. An object *encapsulates* a set of *attributes* describing the internal state of the object. An object also has a set of *operations* or *methods*. The operations together with the attributes identify the *properties* of an object. An operation can be an *observer* which reports on the state of the object, or a *mutator* which is changing the state of the object. The attributes are usually hidden to the outside world, and clients can only access the objects through the operations, which is called the *interface* of the object. Thus, encapsulation is also called *information hiding*.

Together with this construction, the object-oriented paradigm defines a set of four hierarchical abstraction mechanisms:

- *Classification* — objects with a common behaviour and the same set of attributes are grouped together in *classes*. A class consists of an *intent* and an *extent* sharing the same name. The intent defines the common set of operators and sets up a state space for the objects. The extent is the set of objects belonging to the class. An object who is a member of an extent of a class, is said to be an *instance-of* that class, e.g. Norway is an instance of a monarchy.
- *Generalization* — classes with similar behaviour are grouped into higher order classes using a mechanism called *inheritance*. The higher order class is called a *superclass* while the lower level classes are called *subclasses*. An object-oriented system may allow *multiple inheritance* which means that a class can have several superclasses, or it may allow only *single inheritance* which means that a class can only have one superclass. Subclasses are related to their superclass by an *is-a* relationship, and by relating every classes to each other, a *class hierarchy* is formed. An instance of a class is also an instance of all superclasses of that class, e.g. a monarchy is a country, and hence, Norway is also an instance of a country.
- *Association* — several objects may be related to each other in an *is-associated-to* or *is-member-of* relationship. Association is, roughly speak-

ing, a similar concept as the relationship concept in the ER-model, e.g. a country may be a member of the European Union.

- *Aggregation* — complex objects may be built up from other part objects which have their own set of attributes, and may again be built up from other part objects. Thus, aggregation is a stronger relationship between objects than that of an association. Part objects are related to their owner objects through a *has-a* or *part-of* relationship. This relationship form a directed graph called *class-composition hierarchy* which is an orthogonal concept to that of a class hierarchy, e.g. a monarchy may have a prime minister and a monarch, both of which are instances of the class person.

In object-oriented systems, any real-world entity is uniformly modelled as an object. In addition, all objects are associated with a unique *object identifier* which is used by the system to identify and retrieve the object.

5.3 Temporal Object-Oriented Databases and Temporal GIS

Object-oriented databases are based on the basic object model, or some variant of this, and provide persistent storage for objects and their schema [Kim90]. In such databases, objects can be stored or referenced in *collections* or *sets* coupling the object-oriented databases tightly to the set theory. However, the coupling between the set-theory and the object model is not as strong as it is between the set-theory and the relational model. Nevertheless, sets can be used to build complex objects, to store associations between objects, or to store collections of objects.

Over the last years, a substantial body of work has been dedicated to the design of object-oriented temporal databases. All of the important issues have been dealt with and are well understood, but no proposed model integrate all issues into one model. In the following, these concepts will be described along with references to related work.

There are at least two ways to implement temporal objects; either by using *object-level versioning*, where each object keep its history in a list of past (or possibly future) object versions, see [FSG92] [SN97a], or by using *attribute-level versioning* where each object keeps a separate history of each of its attribute, see [SSP95], [BFG96] [EGSV97]. Because both methods provide the same semantics, and since the internal representations of the objects are hidden to the outside world,

both methods may co-exist in the same database [WD92]. However, in many situations it makes sense to group and version attributes in clusters such that attributes that are updated together (e.g. the attributes of an address), are also versioned together (i.e. in a synchronized manner) [JS96].

However, the attributes are not the only part of objects that may change over time. Association and aggregation between objects may also change over time. We may have several types of temporal relationships between entities, depending on their semantics and their relative duration over time [MSW92]. Some associations between objects may only exist if both objects co-exist, but in other cases we may have associations between objects at times when either or both of the objects in the association does not exist. One-to-many, and many-to-many relationships may be implemented in form of collections and sets, and we must carefully distinguish between the life cycles of the member objects, and their membership lifespans (i.e. visibility) in sets [SN97b].

Also the fact that objects may change in type has been addressed in several papers, see [Zdo90][LSW97]. Richardson and Swartz [RS91] even recognizes that an object may possess several classes at one time, i.e. have different roles to different clients. Zdonik [Zdo90] distinguishes between two types of classes: *essential classes* which are classes that an object can never lose once it has been acquired (e.g. a person is always a person and can never become an elephant), and *exclusionary classes* which can only be acquired upon creation time and which can never be reacquired once they are lost (e.g. a person may start out as a child, but can never be a child again once it has lost that class).

Databases are often exposed to evolution on the schema level. This may be the result of bad schema design, the domain being modelled is evolving, the organization using the database is changing, laws and legislations have been changed, or independent databases need to be integrated [Bra93]. Schema evolution is a very important concept of temporal databases since historical data always should be studied in context with the rules and the schema current at the time in question, see [DGS95] [DGS97] [MNP⁺91]. Changes at the schema level should be done such that (1) objects that already exist in the database are available through the new classes, and new objects should be available through existing classes, and (2) the other objects (i.e. clients) existing prior to the modification should be compatible with the new classes. Several methods exist [Bra93], with one attractive method being to specify new classes simply by subtyping from existing classes.

Although, some methods outline that modification is to be kept at the class

level, Lauteman [Lau97] proposes a method where different versions of the whole schema is kept in a *schema derivation DAG*¹. This method has many similarities with model versioning mechanisms found in the CAD systems today and the method also represent a branching view of time.

Some work has also been done for schema evolution in temporal (relational) databases. DeCastro et al. [DGS97] have identified two basic solutions for schema versioning in temporal relational databases: The *single-pool solution* where each version of the same class shares the same extent, and the *multi-pool solution* where each version of the same class has its own extent. Related to temporal object databases, Kim et al. [KCLS95] study possible forms of evolution to the schema in a temporal database, indicating a number of change operations to the database schema.

The change and behaviour of objects in temporal systems is also an important perspective of temporal databases. Chen et al. [CLL96] argue that the behaviour history should be stored together with the state history of objects, and explicit links between predecessors and successors should be provided. Mathiassen et al. [MMNS94] also state that the events that are associated to an object should be identified for each object class. Sometimes, new abstract classes representing events have to be constructed in order to incorporate all applicable events.

However, the behavioural aspect of objects are usually expressed in forms of *state-transition diagrams*. Rumbaugh et al. [RBP⁺91] and Martin [Mar93] provide simple models, whereas Coleman et al. [CHB92] presents a system based on Statecharts [Har87], where the expressive power is increased by providing AND and XOR decomposition mechanisms. An example of a XOR decomposition is provided in Figure 5.1. A more complex approach to analyze object behaviour is also presented in [LKH94], but their notation is not as intuitive as the other behavioural models. Also related to the behaviour of objects, is a set of constraints that an object must comply to. Alagic [Ala97], has introduced such a constraint language based on Horn-clauses.

The concept of a query language was earlier an alien concept in object-oriented databases [Kim90]. Instead, the necessary functionality would be embedded in the methods of objects. But now, several query languages for object-oriented databases exist; they are either extensions of SQL-like languages or they are based on the set-theory. A query language also depend upon an algebra in order to express certain

¹DAG: Directed Acyclic Graph

conditions. Several algebras for the querying and retrieval of objects in object-oriented temporal databases have been proposed, see [RS93b] [RS93a] [SN97b]. These algebras are also related to the set theory, and include operators such as temporal union and temporal intersection.

Also within the field of geographical information systems (GIS), a great deal of effort has been spent to extend GIS with temporal modelling capabilities. Several models have been proposed for the spatiotemporal component such as the space-time composite model [LC88], the event-oriented spatiotemporal data-model [PD95] and the Triad model [PQ96]. Worboys [Wor94a] suggests a ‘spatio-bitemporal’ model using so-called T-complexes, which are collections of simplexes that have been assigned a bitemporal lifespan.

Also some object-oriented data models, i.e. class hierarchies, have been suggested for temporal GIS, see [Ham94], [VBH96] [BVH96]. Ramachandran et al. [RMD94] introduce a model-driven approach using object versioning and temporal relationship objects. Another approach that defines a set of temporal and spatiotemporal attributes for the construction of abstract data types as functions over time [ESG97], is of great interest to the current research. However, common to these proposals is that they focus on the realization of the spatiotemporal component of data, and to varying degree, leave the behavioural, semantical and topological aspects of the data undecided.

Within the GIS community, we have also seen an increasing work on the issue of studying the behavioural and the change aspect of geographical information. Egenhofer and Al-Taha [EA92] give a thorough investigation of changes in topological relationships between continuously changing regions, and Frank [Fra94] investigates temporal relationships in terms of partial orders and semi-orders. Recently, we have also seen a number of taxonomies of change types, see [Ren96] [CT96] [CT95] [HE97]. Claramunt and Thériault [CT96] and Hornsby and Egenhofer [HE97] emphasize the qualitative aspect of changes, neglecting the durational aspect (or they assume that changes are instantaneous). In some models, such as [CT96] and [Ren96], the change object has been identified as a distinct object from that of a state object (object version).

Related to changes is also the concept of an *update language*. Lagorce et al. [LSW97] propose an update mechanism where both operators for updating specific instances and for updating the database schema have been included. Similar constructs have been proposed by Rose and Segev [RS93b] and by Zdonik [Zdo90]. In general, we have seven types of update operators for object instances: *add-*

object (constructor), *delete-object* (destructor), *modify-object*, *associate-object*, *cut-objects* (delete association), *add-class* (add class instanceship), and *delete-class* (delete class instanceship).

In the following section, we propose an object-model for temporal and spatiotemporal applications where we try to integrate as many of these aspects as possible in one unified model. First, we outline the basic concepts of the temporal model, and then we define a basic class hierarchy for according to these concepts.

5.4 The Temporal Object-Oriented Model

In this section we synthesize the work described in the previous section and provide a general taxonomy of the concepts of an object-oriented temporal database. Generally, a system of objects is characterized by many properties, all of which may change over time.

5.4.1 Object and Attribute Behaviour

Figure 5.1 shows a state-transition diagram of a temporal object or attribute (in a general sense, an attribute is also an object) using Coleman's *ObjectChart* formalism [CHB92]. According to this model, a temporal object is either *alive* or it is *dead*. If it is alive, it is either in a continuously *changing* state, or it is in a *static* non-changing state. The transitions between each state are identified by *events*, and an event is defined to have no duration. The first event of every object is its *birth*. It then enters a state of being alive, either changing or static. If an object is in a static state, it may start to change and enter the changing state. When an object is changing it may stop to change, i.e. *stabilize* into a static state. When an object is in a static state, it may also be subject to an instantaneous change from one static state to another.

An object ends its life by its *death* entering the state of being *dead*. An object can either be in a static state or in a changing state when the death occurs. When an object is dead, it may become alive again by a *reincarnation* event.

The model provides a general framework for modelling three kinds of object or attribute behaviour. These are objects that are (1) always in a static state and are only exposed to sudden changes (e.g. the borders of parcels in a cadastral database), (2) objects that are always in a dynamic or changing states (e.g. the temperature),

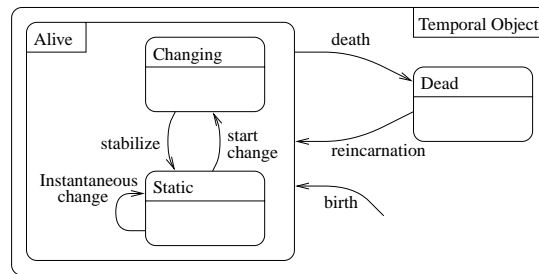


Figure 5.1: An ObjectChart model of a Temporal object

and (3) objects that are sometimes in a static state and other times in a dynamic state (e.g. the position of a ship).

5.4.2 Observing Temporal Objects

An observer of a non-temporal object can only report on the current state of the object. For temporal objects, we may identify several types of mutators: The most obvious is to report on the state of the object (or an attribute) at a specific time, but we may also observe properties that are not associated with a single point in time. We have identified at least five types of observers of temporal objects:

- *The atemporal observer* — which report to the client about properties of temporal objects that is not inherently temporal, e.g. to report the state of a non-temporal attribute.
- *The snapshot observer* — which reports on the state of an object or attribute at one specific time. This operator must have one parameter specifying the time of interest.
- *The behavioural observer* — which report on inherently temporal information without having to specify any time parameter, e.g. to find the age of an object, or the frequency if the object has periodic fluctuations.
- *The time-interval observer* — which reports on the change or difference in state of an object during a time interval, e.g. how much or how quickly did this object change during a time interval.

- *The history observer* — which returns the entirety or a part of the history of one or more attributes, possibly as a function over time.
- Other observers — we do not attempt to make any further classifications, but accept the existence of observers that are more complex particularly involving comparison with other objects.

5.4.3 Object Evolution and Mutators

In general, mutators correspond to the transitions or events of Figure 5.1. We may have three basic types of mutators. Two types of mutators are already standardized within most object-oriented programming languages, viz. the *constructor* and the *destructor* of the object. In addition, we need external operations, such as the C++ operators *new* and *delete*, to make the actual creation of the object, and to establish the successor predecessor relationship between objects.

In case of the third type of mutator, the *modifier*, we distinguish between four types of modifiers:

- those who change the state or the rate of change of the object,
- those who change the type of the object,
- those who add, remove or replace some part of the object, and
- those who change the associations between objects.

All these mutators need time parameters identifying the real-world time of the changes in question. The transaction time for these mutators is supported by the system itself, while the valid time must be supplied as a parameter to the mutator. Subsequently, no time argument needs to be specified for systems supporting only transaction time, but both valid-time systems and bitemporal systems need one valid-time argument to be passed to the mutators.

Change in State

The attributes of an object may be changed and observed independently. A modifier that reports on a state change to the object may do this to all its attributes or just to some, usually according to how updates are synchronized [JS96]. Basically, for

each attribute, there should be one mutator for each transition in Figure 5.1. A temporal object should therefore have the following mutators which influences the state of the object.

Constructor(time, attributes . . .) which is called automatically upon creation of the new object, e.g. by a *new* operator. The parameters to this operation must be the valid time of the creation, and the values of the initial state of the object. This operation should also take care of the reincarnation of the object.

Destructor(time) which is called automatically upon destruction of the object, e.g. by a *delete* operator. A valid time parameter must be provided as a parameter to the operator.

Attributes are also objects, and those types of attributes that may change gradually should also have the following operations:

StartChange(time, attributeName) Specify that an attribute (or group of attributes) should change from a static state to a dynamic state.

StopChange(time, attributeName, newValue) Specify that an attribute (or group of attributes) should change from a dynamic state to a static state. A new value of the attribute should be specified for the new static state of the attribute.

Update(time, attributeName, newValue) Modify or update an attribute (or a group of attributes) with a new value. If the attribute is in a static state, it is considered as an instantaneous change at valid time **time**. Otherwise, it is considered as a smooth change from the last update.

Furthermore, a mutator that operates on a continuous changing attribute may also provide updates about the rate of change (velocity) and even rate of rate of change (acceleration) for the purpose of more accurate interpolation and extrapolation. E.g we may implement a continuous changing attribute by cubic Hermite interpolation, which demands that a derivate (i.e. a rate of change) for each data point must be provided.

Change in Object Type

When an object changes in type, i.e. from being an instance of one class to become an instance of another class, we may adopt the view that an object can possess (i.e. be an instance of) either several classes at the same time (multi-class model), or only one class at a time (single-class model), e.g. a river may also be the border between two countries, which may be implemented by one cartographic object possessing two classes, border and river.

We suggest three mutators for changing the class as suggested by [LSW97] and [Zdo90]:

AddClass(time, className) In the multi-class model, **AddClass** adds a new class to the object. In the single-class model, **AddClass** moves down in the class hierarchy. A call to this operation may imply that one or more new attributes needs to be added to the object.

DeleteClass(time, className) In the multi-class model, **DeleteClass** deletes a class (or role) from an object. In the single-class model, it moves up one step in the class hierarchy, in which no class name need to be specified. A call to this operation may imply that one or more attributes of the object ‘die’.

ReplaceClass(time, oldClass, newClass) Migrate from one class to another class. This may imply that one or more attributes of the object ‘die’ or are added to the object. In the single-class model, it not necessary to specify the **oldClass** parameter.

In a multi-class model, the history of all the classes that an object possessed during its life time can be expressed in terms of a temporal set. An important constraint to these mutators is that we may not allow an object to possess less than one class.

It is also natural to restrict changes of object types to be of the instantaneous type. Although, gradual changes of type may be common in nature (e.g. an old dirt road may gradually over-grow to become a path), we do not have any support for this in our model.

Change in Aggregation and Association

We assume that a relationship between two objects is a crisp relationship. In other words, we cannot have gradual changes in aggregations and associations amongst objects. Both aggregation and association may be implemented as temporal sets, and the operations to include a new member or to secede an existing member are essential operations pertinent to these matters [SN97b]; e.g. a country may be a member of the European Union, and the members of European Union may be implemented as a temporal set. By using fuzzy-set theory it is also possible to model gradual changes in relationships.

5.4.4 *Validations and Corrections*

If an object has not been changed for a long time, it may give the impression that it has not been updated in a long time, even if newer information validates its current state [Lan93]. We therefore need an operation to validate the current state of an attribute (or group of attributes).

On some occasions, we may also need to make *corrections* to earlier updates if they were wrong. A special type of corrections is the *cancellation* which completely cancels an earlier update. A correction may apply to an earlier update, an earlier correction or cancellation or a validation. Cancellations and corrections can be queryable if the system has a bitemporal database.

5.4.5 *Class Evolution*

If we need to modify a class intent, we cannot allow a destructive approach where the old version of the intent, and hence also the database schema, is lost. We should always be able to study historical data in relation to the schema and the associated constraints existing at the time in question, e.g. if the European Union introduces new requirements for aspiring members of the Union, it must be reflected in the schema history. Otherwise, it would lead to inconsistency if existing members do not meet the new requirement.

As the rules and constraints may be programmed into the operations of the classes, we must have some mechanisms to record the history of the intent of each class. In general, we may outline four possible solutions:

- *Property-level versioning* — For each modification to a property of a class,

a new version of each modified operation and attribute definition is added to the class history. Thus, each class records a history of each operation and attribute type; a method that is akin to the attribute-level versioning method.

- *Intent-level versioning* — For each modification to a class, a complete new version of the class intent is recorded in the class history. Hence, the model is akin to the object-level versioning method. All instances of the class are always treated in relation to the intent current at the time for which the object was inspected.
- *Schema-level versioning* — For each modification to a class, an entire new version of the database schema is created. This model is therefore akin to the database-level versioning method.
- *Universal classes* — When a modification to a class needs to be made, a new class is made by making a subclass (or a superclass) of the existing class. There is no versioning of classes or schema. When a new class has been added, e.g. by subtyping from the ‘old’ class, members of the old class are changed to become members of the new class. This way, we can avoid inconsistency between the class extent and the class intent, if the intent is changed without due consideration to the extent. This is the simplest and most consistent of the schema versioning mechanisms. With this method, it is always clear which class (or classes) an object should be associated with at each time.

It seems obvious that with temporal databases, since the history of each object is captured (including the type history), many of the problems of schema evolution in object-oriented databases can be solved. A class that an object possessed at a particular time can always be accessed through that class for that specific time. In non-temporal databases, one single instance of a class might have to be compatible with many versions of the same class. But as long as the class membership history is stored for each object, we do not have to worry about compatibility with existing objects when designing a new class. We therefore advocate the universal class approach, since this is the simplest of the methods. First, we do not have to create a class-versioning or schema-versioning mechanism. Second, we do not have to associate each class or class version with a lifespan. Third, we avoid much of the complexity related to the fact that old objects are not directly comparable to

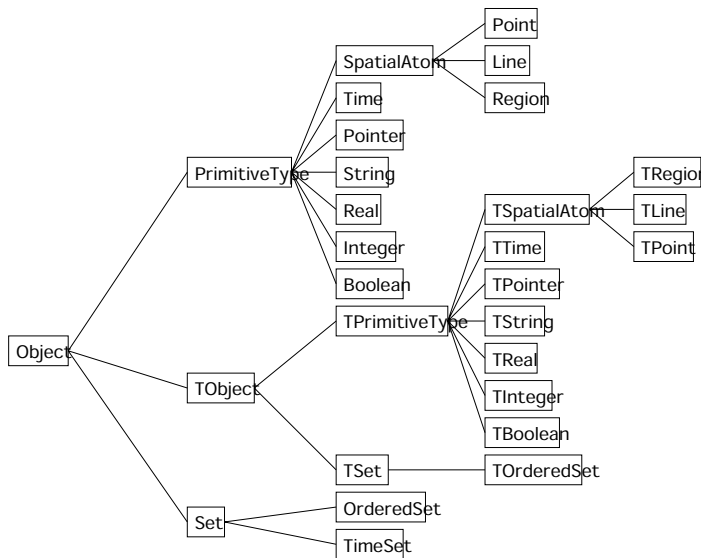


Figure 5.2: The Basic Class Hierarchy

current objects because they relate to different version of the schema. And fourth, two objects may possess different versions of the same class at the same time.

Schema evolution may also be recorded over the bitemporal domain, e.g. if we decide to add a new attribute to a class, say the gross domestic product for each country for each year. Although the new class may be introduced at some time, objects may be changed to the new class valid prior to the introduction of the class (transaction time), if information about gross domestic products for earlier years is available (valid time).

When a new class is created, all the instances of the old class may automatically become instances of the new class upon creation of the new class. This corresponds to an *eager* consistency maintenance. On the other hand, if e.g. an attribute has been added, the instances of the old class become instances of the new class from the time when values for the new attribute is provided. This corresponds to a *lazy* consistency maintenance.

5.4.6 *The Basic Class Hierarchy*

Figure 5.2 shows the basic class hierarchy of our model. The generic class **Object** has three subclasses, **PrimitiveType** which is the superclass of booleans, integers, reals, strings and so on, **Set** which provides a mechanism to store objects in sets or collections, and **TObject** which provides a generic class for all temporal objects (We use the prefix “T” to denote that a class is a class for temporal objects). The **TObject** class have two subtypes, **TPrimitiveType** which is the superclass of primitives that are expressed as functions over time, and **TSet** which support sets as functions over time.

In the following, we will pay closer attention to a few of these classes. Note that we have only included the most basic operations for each class:

SpatialAtom Which is regarded as a primitive type that provides a generic class for the locational and geometrical property of spatial objects. In the case of a two-dimensional model, this property can be either a point, line or a region object. We may also implement other types of spatial objects, such as fields (e.g. elevation models) and fuzzy regions.

Pointer A Pointer is used to reference another object and therefore contains the object-identifier of this object. We assume that the database system has some means of referencing, pinpointing and retrieving a specific object according to some identification (object identifier). The Pointer should provide enough information for the database system to identify and retrieve the object, whether it exists on disk or in random access memory.

Set This class implements a set, and the members of the set are instances of the class **Object**. Hence, a set can have non-temporal as well as temporal objects as members. Often, a **Set** contains pointers to other objects.

TimeSet This class implements a set of times and should be utilized to represent lifespans of objects. It may be implemented differently than an ordinary set, but it provides the same interface since lifespans are, by definition, sets of times.

TObject This is a generic class for temporal objects. It provides the following interface.


```

class TObject is a Object
{
    mutator Constructor(validTime:Time)
    mutator Destructor(validTime:Time)
    mutator AddClass(validTime:Time, className:String)
    mutator DeleteClass(validTime:Time, className:String)
    mutator ReplaceClass(validTime:Time, oldClass:String, newClass:String)
    observer OID() → Pointer
    observer LifeSpan() → TimeSet
    observer Class() → TString
    observer StateOf(validTime:Time) → Object
}

```

TPrimitiveType This is a generic class for all temporal attributes. It should provide the following interface.

```

class TPrimitiveType is a TObject
{
    mutator Update(validTime:Time, newValue:PrimitiveObject)
    mutator StartChange(validTime:Time)
    mutator StopChange(validTime:Time)
    mutator Correct(validTime:Time, newValue:PrimitiveObject)
    mutator CancelUpdate(validTime:Time)
    mutator Validate(validTime:Time)
    observer StateOf(validTime:Time) → PrimitiveObject
}

```

TPointer This class can be utilized to implement one-to-one temporal relationships between objects. The pointer may point to different objects at different times, and the history of which objects it pointed to at which time is recorded within the pointer itself, e.g. a monarchy may have a **TPointer** that points to the succession of monarchs of the country.

TSet This class implements a temporal set. It can be utilized to store both temporal and non-temporal objects. A temporal set of pointers may be utilized to implement a one-to-many or many-to-many relationship between objects. **TSet** have the following interface:

```

class TSet is a TObject
{
    mutator IncludeObject(validTime:Time, newMember:Object)
    mutator SecedeObject(validTime:Time, member:Object)
    mutator DoForEach( Procedure(member:Object))
    observer StateOf(validTime:Time) → Set
    observer MembershipLifespan(member:Object) → TimeSet
    observer FindObjects( Condition(object:Object) → Boolean) → TSet
    observer NumItems() → TInteger
    observer IsEmpty() → TBoolean
}

```

The **FindObjects** operators takes a boolean function **Condition** as an argument and returns all the objects in the set, where **Condition**(object) = *true*.

5.4.7 *Spatial and Spatiotemporal Objects*

Spatial objects (often referred to as spatially referenced objects) are types of objects that have a geometric location as one of its properties. A Spatiotemporal object (**STObject**) is an object whose location may change over time. An object may be spatial and temporal but not spatiotemporal if the object has properties that varies over time, but these are not spatial properties. As illustrated in Figure 5.3 we therefore distinguish between **TSpatialObject** whose location does not change over time and **STObject** whose location does change over time. In order to further emphasize the difference, we assign the name **TLocation** to the operator that returns a **TSpatialAtom** and the name **Location** to the operator that returns a **SpatialAtom**.

```

class SpatialObject is a Object
{
    observer Location() → SpatialAtom
}

class STObject is a TObject
{
    observer TLocation() → TSpatialAtom
}

```

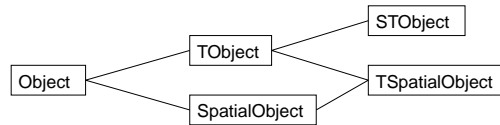


Figure 5.3: Spatial objects, Temporal spatial objects and spatiotemporal objects

```
class TSpacialObject is a TObject, SpatialObject
{
    /* No need to add mode properties */
}
```

5.5 *A Temporal Map over Europe: an Exercise in Spatiotemporal Modelling*

Suppose that we want to make a temporal map over Europe (or even the world), showing the history of countries, their borders, population and so on. This is a very useful exercise, as it covers many important aspects of spatiotemporal modelling, and the temporal object-oriented model.

5.5.1 *Assumptions*

In this exercise, we are going to model all countries in Europe. Each country has a location, population, a capital, and a number of other cities. Some countries are monarchies and thus have a monarch and a prime minister, while other countries are republics and have a president.

Before we start modelling, we must make certain assumptions on which we are going to base our model. These may be as follows:

- Coastlines are spatial objects that delimit countries and are regarded as static objects. Coastlines also bound islands.
- Borders can only change instantaneously. Although borders may be more fuzzy during wars, we assume that borders are crisply defined both in space and time.

- The population of countries and cities changes continuously, although jumps must be allowed when sudden changes in country borders occur.
- A country can only have one capital at a time.
- Countries and cities may change their name.
- A country may change from being a monarchy to being a republic (e.g. the French revolution), or it may change from a republic to a monarchy (e.g. Spain after Franco).
- The position of cities cannot vary over time, and the location of cities are only recorded as points. Because countries change borders, a city may be in different countries at different times, but we assume that a city can never be in more than one country at a time. (However, it is principally nothing wrong in assuming that some cities may lie on the border between two countries, and thus can belong to more than one country)
- The time resolution of our database is one day, allowing us to capture the dates of specific events.
- For the sake of modelling temporal sets, we also want to model multinational organization such as the EU and EFTA. Both which may change name and members over time.

5.5.2 *Modelling the Temporal Database Conceptually*

Figure 5.4 shows an initial model of Europe using the Object Modelling Language (OML) of Rumbaugh et al. [RBP⁺91]. A country is associated with its border and coast line. Some countries, like Iceland does not have any borders, while other countries, like Switzerland does not have any coast line. Many countries have many borders, one border for each neighbouring country, and a border separates two countries. However, since we must allow some borders as be the border of Europe, some borders may only bound one country in the database. A coastline may also bound an island as well as the mainland. A country may therefore have many coast lines, and a coastline may bound several countries.

In Figure 5.5, we have added temporal and spatial markers on the model in a similar way as introduced in the ERT language ([MSW92], [TWL92]). An object

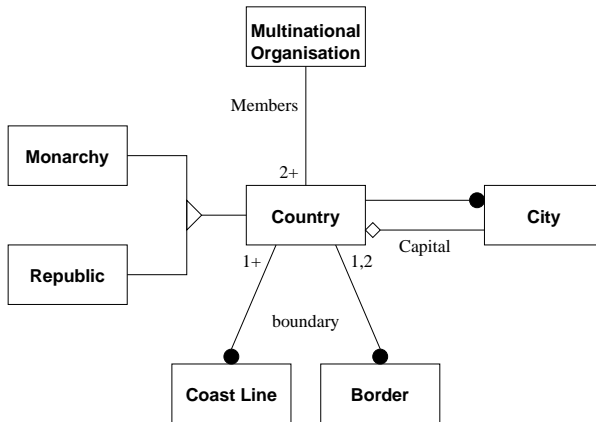


Figure 5.4: A non-temporal conceptual model of the European temporal database

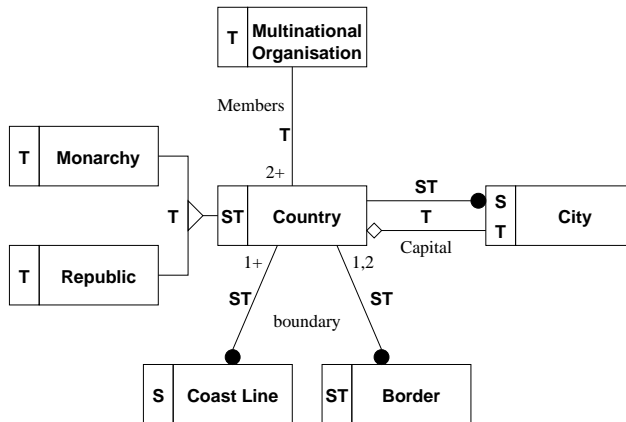


Figure 5.5: A temporal conceptual model of the European temporal database

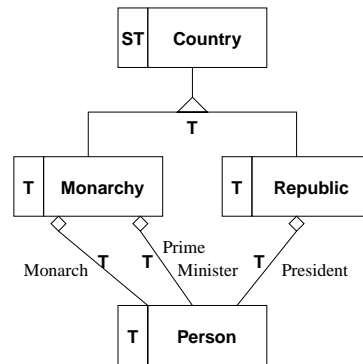


Figure 5.6: Monarchs, Prime Ministers and Presidents are just human after all

class that is T-marked has properties that varies over time. An object class that is S-marked is a spatial object and thus has a location. If an object class is ST-marked, it means that the location of the object may also vary over time. An object class may be both T-marked and S-marked, but not ST-marked, if it has a static location, but there are other properties that vary over time. Subclasses inherit all markers from their superclasses, and are only marked if additional properties are added that are either temporal, spatial or spatiotemporal.

A generalization link is T-marked if the instances of the class may change in type over time, e.g. a monarchy may change to become a republic.

An aggregation or association link is T-marked if the link may exist only during a part of the time for which both involved objects co-exist. If, on the other hand, a link between two temporal object classes is not T-marked, it means that the link exists as long as both objects co-exist. If an aggregation or association link is S-marked, it means that the link is spatially dependent, e.g. a city can only be associated to the country where it is spatially inside.

For pedagogical reasons, we will model the persons who are presidents, prime ministers or monarchs in one separate class. This way, as illustrated in Figure 5.6, we are able to determine when these persons lived, and when they were president, prime minister or monarchs.

5.5.3 *Designing the Interface for each Class*

Our next step is to design an interface for each class. At this stage, we are not interested in the internal representation of each class, but the operations and the expected behaviour that we need for each class.

Countries

Countries are true spatiotemporal objects as both population and location change over time. The **Country** class inherits the location operation from the **STObject** class, but specifies the locational attribute to be of type **TRegion**. A **TRegion** may sometimes be a single connected region if the country only comprises a single area of land (such as Switzerland). A region may also at other times be a disconnected complex region with many connected sub-areas such as islands or enclaves. In either case, the region is modelled as a single object, containing the entire history of a country's location.

The population of a country is usually modelled as a continuously changing attribute. If the population of a country has changed gradually since last update, the mutator **UpdatePopulation** is used. However, if a new country is deducted from another country, we may see a jump in the population. In this case, the **ChangePopulation** mutator is used. The **ChangePopulation** mutator takes, in addition to the valid time parameter, two population values as argument: **population1** contains the population right before the jump, whereas **population2** contains the population right after the jump.

```
class Country is a STObject
{
    mutator UpdatePopulation( newPopulation: Integer, validTime:Time)
    mutator ChangePopulation( population1, population2: Integer, validTime:Time)
    mutator ChangeName( newName:String, validTime:Time)
    mutator ChangeCapital( newCapital:City, validTime:Time)
    observer Name() → TString
    observer Population() → TInteger
    observer TLocation() → TRegion
    observer Area() → TReal
    observer Capital() → TPointer to City
    observer Neighbours() → TSet of Pointer to Country
```

```

observer Cities() → TSet of Pointer to City
observer GetMemberships() → TSet of Pointer to InternationalOrganization
}

```

We also want to find the neighbours with which the country shares a border. This observer returns a temporal set of country pointers, referencing the countries that were neighbours at different times.

In the database, we will have no direct instances of the class **Country**. Instead, they will be either of the type **Republic** or of the type **Monarchy**.

```

class Monarchy is a Country
{
  mutator ChangeMonarch(newMonarch:Person, validTime:Time)
  mutator ChangePrimeMinister(newPrimeMinster:Person, validTime:Time)
  mutator BecomeRepublic(validTime:Time)
  observer Monarch() → TPointer to Person
  observer PrimeMinister() → TPointer to Person
}

```

```

class Republic is a Country
{
  mutator BecomeMonarchy( validTime:Time)
  mutator ChangePresident( newPresident:Person, validTime:Time)
  observer President() → TPointer to Person
}

```

Cities

At the abstraction level that we want to work with, it is sufficient to locate cities to a single point. A city (according to our assumptions) can change name and the population, but not its location, thus a city is a subclass of **TSpatialObject**.

We also want to find the history of which countries each city is in. This may not be explicitly stored with each object, as it is possible to derive it from the location

of the city and the location of the countries. However, it is not the scope of this article to suggest how to implement this function.

Finally, we also want an operation to find out when (or if) the city has been a capital. Again, this may not be explicitly stored together with each city object, as it is possible to derive the information otherwise.

```
class City is a TSpatialObject
{
  mutator ChangeName(newName:String, validTime:Time)
  mutator UpdatePopulation( newPopulation:Integer, validTime:Time)
  observer Name() → TString
  observer Population() → TInteger
  observer Location() → Point
  observer Country() → TPointer to Country
  observer IsCapital() → TBoolean
}
```

Borders and Coast lines

All countries are locationally bounded by borders and coast lines. Whenever we create a new border, either of the following two situations occur: A new country is deducted from another country (e.g. the Baltic countries separates from the Soviet Union), or a country is split to form two new countries (e.g. Czechoslovakia splits to form Slovakia and Czech Republic). Similarly, if a border is deleted, either two countries are merged to form a new country (e.g. East Germany and West Germany form Germany), or one country is annexed into another country (e.g. Sweden gets Norway in 1814).

```
class Border is a STObject
{
  mutator Adjust(newLine, validTime)
  observer Countries() → TSet of Pointer to Country
  observer Length() → TReal
  observer TLocation() → TLine
}
```

```
}

```

```
class CoastLine is a SpatialObject
{
  observer Countries() → TSet of Country
  observer Location() → Line
}
```

Monarchs and National Leaders

We choose to model prime ministers, presidents and monarchs as a type **Person**. The persons are stored in a **Set**, and each person has a lifespan which is equivalent with the persons real lifespan. The sequences of presidents, prime ministers and monarchs are then referenced via a **TPointer**. Each **TPointer**, say to the Monarch, only points to the person during which the person was a monarch.

```
class Person is a TObject
{
  mutator ChangeName( newName:String, validTime:Time)
  observer Gender() → {Male, Female}
  observer Name() → TString
}
```

International Organizations

International organizations are organizations such as the European Union, EFTA, NATO and the Nordic Council. Each of these organizations may have both a name that may change over time, and a set of member countries that may change over time. The membership in these organizations may be both full or associate. Thus, we propose the following interface for the **InternationalOrganization** class:

```
type MembershipType = {Full,Associate}
```

```
class InternationalOrganization is a TObject
{
    mutator AddMember(newCountry:Country,
                     type:MembershipType, validTime:Time)
    mutator ChangeMembershipType(member:Object,
                                 type:MembershipType, validTime:Time)
    mutator SecedeMember(member:Object, validTime:Time)
    mutator ChangeName( newName:String, validTime:Time)
    observer Name() → TString
    observer GetFullMembers() → TSet of Country
    observer GetAssociateMembers() → TSet of Country
    observer GetAllMembers() → TSet of Country
}
```

5.6 Querying the Database

Since we only can retrieve information about objects only via the operations, we must be able to use these operations in queries in order to retrieve data from the database. A problem with this approach is that the internal implementation of the objects is not known outside the object. This makes indexing and query optimization a hard task. The advantage of the approach is flexibility. In a relational database system, the queries are limited by the attributes and the expressiveness of the query language. But with an object-oriented database, it is possible to add new operations (i.e. observers) to the classes during implementation in order to support specific queries needed for a particular problem. This is valid whether some traditional query language (e.g. SQL-based), Tomlin's map algebra [Tom90] or Qian's visual query language [QP98] is used.

For example, using the temporal object-oriented query language (TOOSQL) [RS93a], we may try to list all countries in Europe whose borders have not changed since 1900. One naive way to do this is to find all countries whose location in 1900 is identical to the location *now*.

```
SELECT A.Name(now)
FROM A:Country
```

WHERE A.Location(1900) = A.Location(now)

However, this approach is not bullet-proof as some countries may have changed at some time after 1900, but changed back at a later time. Alternatively, we could use the \forall or \exists qualifiers to check whether that the location is identical at all times since 1900

```
SELECT A.Name(now)
FROM A:Country
WHERE  $\forall t \in [1900, now) : A.Location(t) = A.Location(now)$ 
```

This approach is of course expensive if we have to check all of the approximately 36000 days since 1900. Instead, temporal objects can be equipped with a **LastChanged** observer, which returns the last time when the object changed. An object that is in continuous change would return the current time. The final query then becomes:

```
SELECT A.Name(now)
FROM A:Country
WHERE A.Location.LastChanged() < 1900.
```

Another example is to find all cities whose population at some point decreased after 1900. One way to do this is to take out an interval of the population function by the **WHEN** clause, then find the derivate of this interval with respect to time, and finally find out whether this was negative at some point. Thus, we provide a **Derivate** and **IsNonNegative** function to the **TInteger** class. The query then becomes as follows:

```
SELECT A.Name(now)
FROM A:City
WHEN [1900.01.01, now)
WHERE A.Population.Derivate().IsNonNegative() = false.
```

The object model that we have presented so far, is by far complete. In other words, we have supplied each object class with only the most basic operators. But, in order to support some particular query at hand, it is easy for the developer to add new operators for a specific class by subtyping from existing classes.

5.7 *Concluding Remarks*

In this article we have discussed the fundamentals of a temporal object-oriented model and provided the necessary functionality for such a model in geographical information systems. Our focus has been on an overall level, rather than focusing on implementational details. We have identified important object classes such as temporal and spatiotemporal objects, as well as temporal and non-temporal sets and temporal pointers. We have also given an initial set of operations and have shown the applicability of the model by an example, describing a temporal political map over Europe.

We have shown that the object-oriented model provides a flexible and expressive framework for modelling real world objects. However, based on the framework described in this article, we suggest that future research should be carried out along two directions. One direction is the implementational level, e.g. how to implement the temporal attributes and temporal sets and, not to forget, how to implement the spatial data repository and the associated indexing methods. The other direction is related to data analysis and queries, which we have only briefly discussed in this article. These topics include issues of presentation and visualization of the query results and problems of user interface design.

ON GENERALIZATION AND DATA REDUCTION IN SPATIOTEMPORAL DATA SETS

6.1 Introduction

Applications of digital cartography and geographical information systems often involve huge volumes of spatial data. In order to optimize performance and readability, it is important to optimize data volumes and information density in relation to requirements of accuracy, performance and legibility of presented and analyzed information. Generalization and data reduction methods are therefore critical components of geographical databases.

Over the last years, a new generation of geographical information systems that is capable of representing spatial data over time has been investigated. Since these systems also handle historic information in addition to the current information, they involve even larger volumes of data. Hence, generalization and data reduction become an even more critical part of temporal databases than traditional spatial databases.

Generally, generalization in cartography is a process that map makers successfully have done since the very first maps were published. When maps migrated into computers, it was natural that the generalization process would follow. However, although the generalization process has been a popular research topic over the past years, it has been difficult to find good methods that makes the process more automatic. In the case of spatiotemporal information, there are not many contributions to this topic. Langran gives a brief discussion in [Lan93] while Monmonier gives a fairly comprehensive discussion related to map animations [Mon96]. The pur-

pose of this paper is to present some initial ideas related to generalization and data reduction in temporal data sets in general and spatiotemporal data sets in particular.

In the remainder of this section, we will first clarify the distinction between generalization and data reduction and then make a brief review of operators of cartographic generalization. Section 6.2 presents a number of applications of spatiotemporal information systems where generalization and data reduction play an important role. Section 6.3 presents the history graph notation which we are going to use to illustrate some of the ideas. And finally Section 6.4 presents some ideas related to cartographic generalization, temporal generalization and map animations.

6.1.1 Data Reduction vs. Generalization

Normally, a distinction is made between *model-oriented generalization* which is pertinent to the computer representation, and *cartographic generalization* which is pertinent to the visualization process [MLW95]. Data reduction is then considered as a sub-set of model-oriented generalization.

However, this paper will use the following distinction of what is data and what is information: *data* is pertinent to computers, while *information* involves some kind of human interpretation. Thus, data convey information when presented to a user in a sensible format. In a similar way, we also distinguish between *data reduction* and *generalization*:

- *Generalization* is the process of reducing the amount of *information* conveyed in a data set such that the *information density* (amount of information per area unit) is kept at a reasonable level in relation to the scale, resolution and purpose of the map.
- *Data reduction* is the process of reducing the *volume* of a data set without (significantly) reducing the amount of information conveyed by the data or without loss of accuracy.

Generalization and data reduction must therefore be considered to be conceptually different processes since their objectives are different. However, many generalization algorithms will also reduce the data volume, but not necessarily, e.g. a line generalized using a low-pass Fourier filter [Bou89] will contain the same number of points as the original line. Nevertheless, it is obvious that generalized

information can be represented by less data than detailed information. It is therefore a good idea to do data reduction after generalization in order to optimize the data volume as much as possible.

On the other hand, generalization may also be used to reduce the data volume, e.g. if we have a data set that is too large for our needs, even after data reduction, we may choose to trade accuracy and information content for reduced data volume. Hence, a generalized data set is used instead of a detailed data set in order to get an acceptably small data volume.

6.1.2 Generalization Operators

The process of generalization is often subdivided into a number of sub-tasks or operators. McMaster and Shea [MS92] divide these operators into two main groups: *spatial transformations* (10 operators) and *attribute transformations* (two operators):

Spatial Transformations:

- *Simplification* — in principle, equivalent to data reduction.
- *Smoothing* — to smooth out rough edges and remove spikes. This operator is, in a general sense, equivalent to a low pass filter since it filters out low frequency variations.
- *Aggregation* — to join point objects into a ‘point-group’ object or into area objects, e.g. a group of buildings is grouped into a settlement object.
- *Amalgamation* — to group several area objects into a larger area object, e.g. if several smaller settlement objects are situated close to each other, they may be amalgamated into one larger settlement object.
- *Merging* — two or more parallel line objects are merged into one line object, e.g. if a double railway may be merged into one single railway symbol.

Aggregation, amalgamation and merging are operators that join several objects into one higher-order object.

- *Refinement* — to remove small and unimportant objects.

- *Exaggeration* — to increase the size of important features to make sure they become visible in the presented map.
- *Enhancement* — to increase the size of a map symbol to make small but important objects become visible.
- *Displacement* — to move objects apart in order to avoid conflicts or overlapping map symbols.

Attribute Transformations:

- *Classification* — group objects of similar types into a higher order class, e.g. to group grain fields and corn fields into cultivated land. If all classes are recursively grouped together, a *class hierarchy* is formed.
- *Symbolization* — to assign symbols to objects such that they become visible and give an idea of the semantics of the object.

Some of these operators make sense only if the data is to be visualized. These operators include exaggeration, enhancement, displacement and symbolization. Therefore, these operators will be referred to as *visualization operators*. It is also these operators that distinguish model-oriented generalization from cartographic generalization, since they do not apply to model-oriented generalization.

6.2 The Utilization of Generalization and Data Reduction in Spatiotemporal Data Sets

6.2.1 Generalization as a Tool for Data Retirement

Without control, a temporal database will grow monotonically in size because older data are superseded, but generally not deleted. Available disk space, and performance requirements dictates that sooner or later, data must be removed from the database. The most intuitive way to do this, is to remove the oldest data. Langran suggests a two-step strategy involving two levels of past: near and distant [Lan92]. The near past can for example be compactly stored on primary disk space with no extra indexing, while distant past data can be stored on backup media on the shelf

(such as CD-ROM disks). Retired data is then still available to the user, but to an extra time cost.

The drawback with this method is that queries like ‘What is the full history of this object’ cannot be answered unless retired data has been made available. A better strategy is therefore to retire data by generalization. Based on the assumption that requirements for accuracy and precision is lower for older data than for recent data, we may apply a generalization strategy where the level of generalization increases with the age of the data. This way, the database can respond adequately to queries involving full histories of objects, although details of the older history may be lost.

However, if we want to sustain accessibility to such retired data, there is a problem associated with how to build in ‘awareness’ of data that is removed by generalization. Obviously, awareness occupies space which again reduces the effect of the data reduction. Moreover, there is a question of the need for a one-step or two-step retirement strategy.

Webster’s chronology of major dates in history¹ provides an excellent example of such a generalized data set. The first events described in this chronology are vaguely described and dated such as ‘736-716 BC: The first Messenian war; Sparta conquers Messina’. Later in the chronology, dates are becoming more and more accurate, and more and more events are described per year, e.g. ‘1990, Dec. 9: Lech Walesa, leader of the Solidarity union movement, is elected president of Poland’.

6.2.2 *Multiple Scale Temporal Databases*

National mapping agencies often maintain map data at different scales and generalization levels; often one data set for each scale interval. These data sets are in many cases updated independently, or at least manually for each map [MLW95]. Hence, there is a gain in providing update propagation mechanisms which automatically generalizes new updates to smaller scale data sets [Tii95].

Modelling and construction of such multi-scale databases are now subject to an increasing research interest (such as [DTR96]). A potential problem with such databases is that some updates should not propagate to smaller scale levels unless a certain amount of changes has been accumulated. But, how is it possible to

¹In Webster’s Encyclopedic Unabridged Dictionary of the English Language

determine when some update needs to be propagated when the database do not remember the data that the generalized version is based on? Having the capabilities of a temporal system in such databases, this problem can be solved. Since such databases also keep track of *when* updates are made, they can also determine when there is time to propagate an accumulation of changes to a smaller scale data set.

6.2.3 *Map Animations*

When making maps legible and readable, the cartographer has to make many adjustments in symbolization, position and sizing of map objects. Introducing the time as a cartographic variable, the cartographer may be faced with a whole range of new opportunities. Viewing a spatiotemporal data set is not only carried out by means of viewing snapshots, but also by means of animation sequences.

The animator in this case, carefully have to exploit the visual [Ber81] and the dynamic variables [Mac94] to produce effective animations. They also need to perform generalization not only in the spatial dimensions, but also along the time dimension to clarify the course and development of the changes in the data set, and to avoid flickering and potential misinterpretations.

These problems has recently been addressed by Monmonier [Mon96], who gives a fairly comprehensive list of methods. For the comprehensiveness of this paper, we will comment on his work in section 6.4.4, supplied with some of our own results.

6.3 *A Notation for Temporal Data*

Following Mourelatos' work in linguistics [Mou78], we propose a system with two distinct types of entities: those objects who describe the static properties amongst objects, and those who describe movements and changes of objects. In the sequel we call these entities *states* of an object and *changes*. These entities can be linked together such that a change that was applied to an object is linked with the previous state and the next state that represent the result of that change.

In this paper, we use a Petri-net like notation called *history graphs* that was originally introduced in [Ren96]. Here, a rectangle represents the state of an object, whereas a round-ended rectangle represents the change entity. The rectangles are displayed parallel to the horizontal time axis and the length of the rectangles

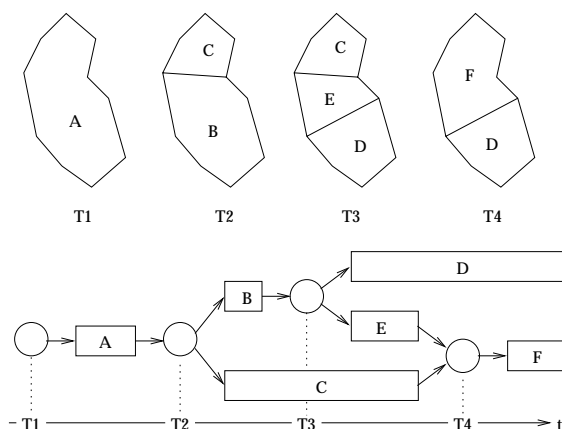


Figure 6.1: The history of objects within a region

mimics the lifespan of the states or the changes. An instantaneous change is therefore indicated with a circle, whereas a snapshot of a continuous changing object (i.e. an instantaneous state) is represented by a very short rectangle. The history of an object is always represented by a sequence of alternating changes and states. Every change have one or more *input states* and one or more *output states*. A change may have several output states and one input state if the change indicate that an object was split, or it may have several input states if the change indicate that several objects were merged.

Figure 6.1 shows a small data set of a region that is split and merged. The reason for choosing this type of notation is that we more easily can visualize gradual changes of some types of objects which are not possible to describe as functions over a metric space, e.g. if a house is built, the change from ‘no house’ to a finished house cannot be interpolated or be described by a simple function.

6.4 Methods of Generalization and Data Reduction

As indicated above, McMaster and Shea [MS92] distinguish between two types of generalization transformations: spatial transformations and attribute transformations; in the sequel referred to as spatial and attribute generalization, respectively. For spatiotemporal data sets, we also add temporal generalization which operates

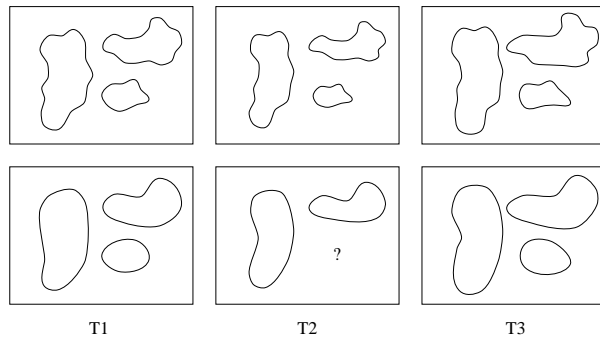


Figure 6.2: Glaciers contract and expand due to climatic changes. Why did the small glacier disappear at time $T2$?

in the time dimension. Because visualizing spatiotemporal information naturally would involve some kind of animation, they should be treated slightly differently than the spatial operators. Therefore, we also take out the visualization operators and present them in a separate subsection.

6.4.1 Spatial Generalization

The application of a generalization operator depends on several factors, such as the spatial and non-spatial characteristics of the object, as well as the surrounding objects and context in general. The problem of spatial generalization in a temporal data set is that these characteristics change over time. Changes between two snapshots that result from inconsistent results of spatial generalization may erroneously be perceived as physical changes, while they in fact are two different views of the same original object. Therefore, two successive snapshots (or frames) should not be generalized independently.

In the sequel, we will discuss and give examples of the generalization transformations described in section 6.1.2, applied to a spatiotemporal data set.

Refinement and Smoothing

Generally, refinement and smoothing mean to remove information, either whole objects or parts of objects (e.g. to remove a small narrow fjord along a coast line). This introduces two problems: First to identify chunks of information and second,

to determine whether these are sufficiently insignificant to be removed. A problem is that a chunk of information can be significant at one time, while not at another time; either because the object is changing itself or because the context changes.

Consider a glaciologist who are monitoring individual glaciers and their annual fluctuation patterns. In a lower scale database, the glaciologist may want to monitor long term fluctuations. If the inclusion of a glacier on the generalized map is determined by the size only, as shown in Figure 6.2, it is likely that some small glaciers will be included at some times and not at other times. This could be an undesirable situation. It is therefore important that one consider the importance of information chunks at a larger temporal scale such that the information is not eliminated or simplified for shorter time periods in the data set.

Aggregation, Amalgamation and Merging

In general, aggregation, amalgamation and merging mean to *group* several objects into one *group object*. This can introduce two problems in a temporal data set. The first problem is how the resulting group of objects should reflect the history of the involved objects, and the second problem is how to avoid the situation where the same object is grouped into different groups at different times.

In the first case, the problem is basically how to deal with the problem that the spatial extent of the group object change because elements ‘come and go’ in the group, or that the elements themselves undergo spatial changes.

We may illustrate this problem with an example. Consider a municipality that is mapping their city in a detailed database with houses and parcels given as individual objects. The national mapping agencies uses this databases to create their small scale regional maps.

Figure 6.3 illustrates this situation. In the generalized data set, this area is given by a group object which roughly outlines the perimeter of the settlement. The question now is: When did this area come into existence? The individual houses were built over a time period of about one year each (shown as changes), whereas it could typically take three years to build the whole area. Since the last three houses were built considerably later than the other ones, we have chosen to represent the settlement with one intermediate state (P). The changes that are associated with the generalized area match the times when the first house was being built and when the last house included in the area was finished.

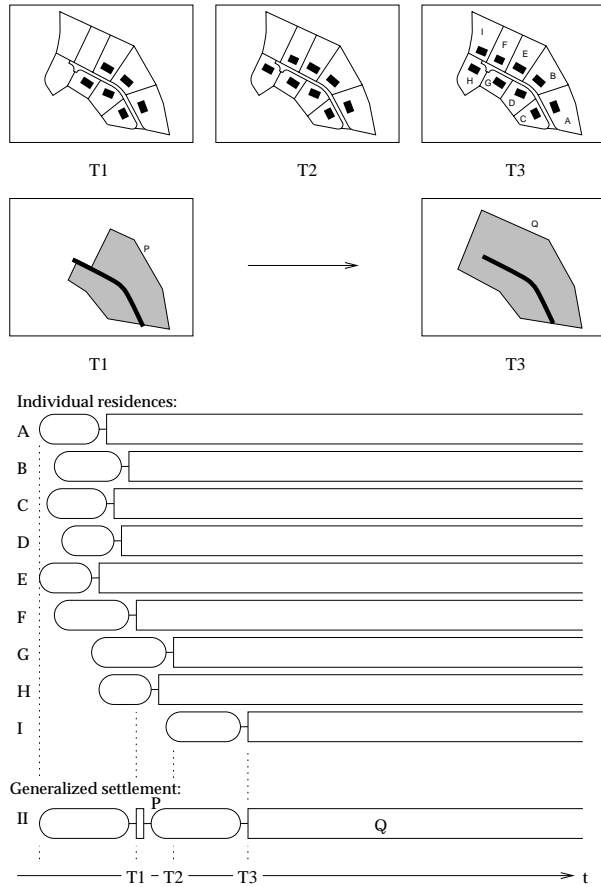


Figure 6.3: Grouping temporal objects together

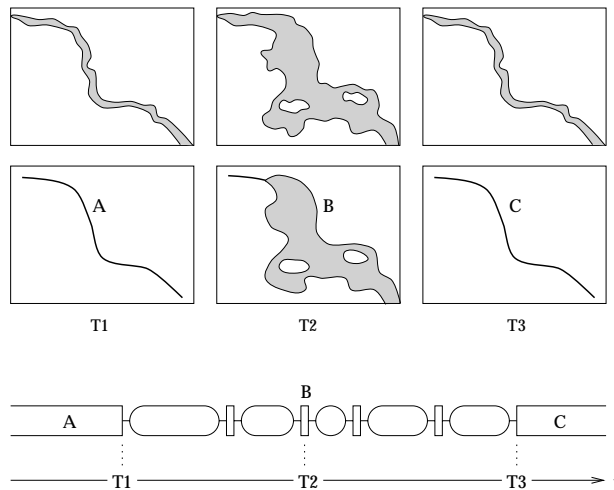


Figure 6.4: During flood, a single-line river can be represented as a two-line river showing the different states of the flood

Collapse

When an object collapses in an animation, it is a sudden event although the change that resulted in the collapse may be smooth, e.g. a city may grow in population, but when the population has grown above a certain threshold it changes from being a point object to become an area object.

Figure 6.4 shows another example where a power plant is monitoring their river in order to give estimates of power production during the seasons. The national water resource and energy administration on the other hand, operates a generalized database where the river in question is derived from the database of the power plant. Normally, this river is represented as a line object, but as Figure 6.4 illustrates, during flood, they are interested in the expansion and the damage and uses a two-line representation of the river. The figure shows how such a flood may be represented spatially in the generalized database; with several states of the flood to more accurately represent the course and development of the flood over time.

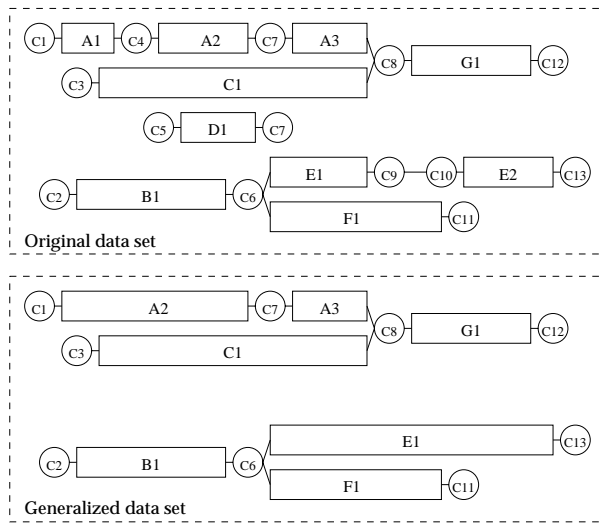


Figure 6.5: Temporal data reduction

6.4.2 Attribute Generalization

Attribute transformations include classification and symbolization. Because symbolization is pertinent to the visualization process, we will describe symbolization in a subsequent subsection.

We view classification in temporal data sets to be unproblematic in relation to spatiotemporal data sets. As far as we can see, there are no problems or side-effects in the temporal dimensions, as long as the classifications is made consistently for all time periods, and according to a predefined class hierarchy.

6.4.3 Temporal Generalization and Data Reduction

In this section we are going to consider aspects of data reduction that can be visualized using history graphs. Let us consider the small data set shown in Figure 6.5 with the original data set above and the generalized set below. Studying the graphs, we see that the change $C4$ has been removed, either because it was a small change in a less significant attribute, or the duration of one of the states was short enough to justify replacing a state with a previous or successive state. In this case, we have chosen the state $A2$ to be the most representative for the period in question since

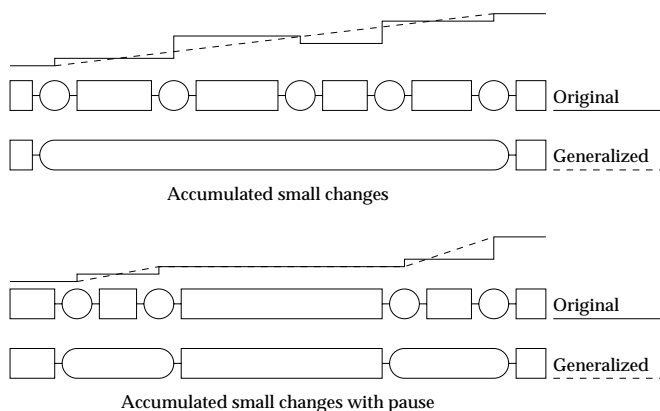


Figure 6.6: Accumulation of small changes over time and how to represent this in a generalized data set

it has a longer duration than $A1$. Furthermore, object D existed for such a short time, it was eliminated, and similarly object E was absent for a short time it could be ignored.

Accumulated Changes

Bringing these concepts one step further, we may also find a solution to the accumulation problem mentioned in the beginning of this paper. In general, minor changes may not propagate to the generalized version since these changes will not contribute significantly to the shape of the generalized version.

But, what if a series of small changes is accumulated over a long period causing the main characteristics to change over that time? To illustrate this, we may consider the same example illustrated in figure 6.3. The national mapping agency, receives regular updates about expanding city perimeters. But normally, none of the updates are large enough to justify propagation to a lower scale level. But over time, the old generalized city does not represent the city at the required precision anymore.

Figure 6.6 outlines a solution to this problem. Instead of representing each change individually, one may consider the city as being in a continuous and gradual expansion. This would be represented with one gradual change having a longer

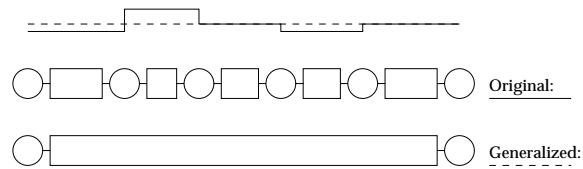


Figure 6.7: Objects in an unstable state can be generalized to be in a stable static state if the changes never changed the object significantly

duration, instead of many states punctuated by events. Intermediate states can also be given if the changes last over a longer period to more accurately represent the course of change. If one needs the state of a city between two states, this can be done by linear interpolation.

Unstable States

A related problem to the accumulation problem is the situation where the changes have *not* made any significant difference to an object over some time, e.g. a census bureau is monitoring the population in their municipality and the various regions therein. People are moving in and out of the municipality so the population changes all the time. The county on the other hand want to monitor the population trends in the municipalities over a longer period. A municipality which has a fairly stable population over a period, may be represented by one single state for that period, saving a lot of space.

Figure 6.7 illustrates this situation. The upper graph may show the states of the population that the census bureau have in their database, while the graph below shows the generalized data set that the county has made from the census bureau database.

Continuous changes and intermediate snapshots

Many objects that change continuously, exhibit a smooth and often predictable behaviour. As illustrated in Figure 6.8, in the case where the gradual change of an object is given by frequent snapshots, one may consider removing some snapshots if they can be interpolated from other snapshots with a sufficient accuracy. The well-known Douglas-Peucker algorithm is typically a good candidate for such a

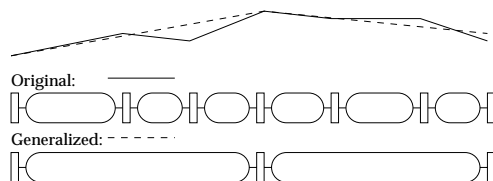


Figure 6.8: Objects that change smoothly and continuously, may be given with fewer intermediate snapshots if these can be interpolated from other snapshots with sufficient accuracy

data reduction.

6.4.4 Cartographic Generalization and Animated Maps

Section 6.1.2 described the four visualization operators smoothing, displacement, symbolization and exaggeration. As with the information reduction operators we must take care when applying these in animated maps. An animated map sequence may appear visually jarring and flickering if each map frame is generated independently.

At least eight different approaches to conquer these problems have been identified (six of them are described in [Mon96]):

Blurred Transitioning

If an animation is displayed with a large time-scale, i.e. that there is a considerable time gap between each animation frame, it might be helpful too fade symbols between two frames to diminish the distracting effect of jumping and flashing symbols. This operator is equivalent to smoothing in the spatial dimension.

Linked Design

If two successive frames in an animation are generalized independently, the result may be different, and the animation may show changes that are not really changes. It is therefore important that the animation is built on a spatiotemporal dataset, and that the generalization operators take the time dimension into consideration. In fact, it is this problem that we have been discussing in the previous two subsections.

Temporal Suppression and Smoothing

If an otherwise smooth movement or change has a spike, it may be advisable to remove the distracting spike in the animation sequence. Also if a feature have a quite irregular behaviour, it may be worth smoothing it out in order to emphasize the general trend.

Chorodots

If an object is located near the border between the two pixels in the animation, it might occur that the object is allocated to different pixels at different times. Such jarring movement is distracting and can be avoided by using the chorodot technique proposed by McEachren and DiBiase [MD91], where the positions of large square dots are fixed within a grid.

Temporal Exaggeration

Exaggeration in the temporal dimension means to lengthen the lifespan of an object or object state if they are too short to be noticed in an animation. Similarly, duration of absence and changes can be lengthened to obtain a similar effects. Exaggeration is also used to increase the duration of a spike in order to make sure that the viewer discovers it.

Temporal Displacement

Two events that occur in close succession may be separated in time in order to emphasize that the two events did not occur simultaneously, if they happend to fall within the same animation frame. The construction of a new international airport in Norway may be an example of this. Inevitably, such an airport will attract companies to establish in the area. The cartographer may then postpone the construction of related industrial areas to emphasize the fact that the building of such areas is a consequence of building the airport (and not the other way around).

Temporal Symbolization

In 1981, Bertin published the theory about the seven visual variables [Ber81]. Introducing time as a cartographic dimension, we may add a number of *dynamic*

variables [Mac94]. The cartographer must carefully exploit these variables when making an animation, e.g. if an object appears, changes or disappears during animation, the cartographer may want to attract the viewers attention towards the change, by showing the event in a flashing style. On the other hand, the cartographer must also avoid attracting the attention away from other (more important) changes; e.g. if a dam is built over a waterfall, which is an event with great consequences, it is important to make this change appear dominating compared to the construction of the associated road.

6.5 *Concluding Remarks*

This paper has enlightened some aspects in generalization of spatiotemporal data sets. With the help of the history graph notation, we have demonstrated some new aspects of the generalization and data reduction problem in spatiotemporal data sets.

In particular, we feel that it is important to meet the challenge with increasing data volumes in spatiotemporal data sets. We suggest that data reduction and generalization methodologies should play a role in the conquest against the monotonically growing size of temporal databases.

Moreover, we have also discussed some issues dealing with map animations. In general, Monmonier has given a fairly comprehensive discussion of these matters [Mon96], but we have supplemented his work with a couple of more operators.

We believe that generalization is inherent to cartography, and generalization necessarily have to follow when we are migrating cartography into the temporal domain. It is therefore important to discuss these matters, also in the context of spatiotemporal data and information.

INDEXING AND REPRESENTING BITEMPORAL LIFESPANS USING BINARY TREES

7.1 Introduction

We assume that we have a temporal database that represent some part of the real world. Thus, when changes occur in the real world, the database needs to be updated. Hence, we distinguish between two time dimensions in temporal databases. These are *valid time* (VT) which is pertinent to when facts are true in the real world, and *transaction time* (TT) which is pertinent to when facts are current in the database. These two times are considered to be orthogonal and time can therefore be said to be two-dimensional. A database system that supports both time dimensions is called a *bitemporal database* [Sno92].

Introducing the bitemporal database, we may now be able to distinguish between proactive ($VT < TT$), retroactive ($VT > TT$) and real-time ($VT = TT$) updates. We also become able to distinguish between updates and corrections and to explain why certain decisions became flaws if they were based on incorrect or non-current data.

Although, research in temporal databases started to emerge in the 1970s [Sno90], bitemporal databases have not really received much attention until the beginning of 1990s [Kli93]. Ben-Zvi [BZ82] suggested already in 1982 a scheme using five time stamps to capture the multidimensionality of time. However, his pioneering work remained less known to the database community at large until recently [Gad93a]. Some of the recent research has been dedicated to the design of temporal and bitemporal support in query languages such as TQuel [Sno87]

or SQL/Temporal [SBJS96], but only a few models have been suggested for the representation of bitemporal lifespans and the indexing thereof. Nevertheless, a very important contribution to the bitemporal database research is the bitemporal conceptual data model presented by Jensen and Snodgrass [JS96].

This article deals with the problem of representing *bitemporal lifespans* of different versions of the same entity in a bitemporal database. A bitemporal lifespan can in most cases be given as a region in $TT \times VT$ space with sides parallel to the axis. The most common and simplest method found in the database literature is simply to store bitemporal lifespans in terms of rectangles. Each tuple in a temporal relation is associated with one rectangle, bounded by VT_{start} and TT_{start} as the lower left corner and VT_{end} and TT_{end} as the upper right corner. In this way, one rectangle describes the valid time and transaction time intervals for which a certain fact was true. Rectangles that are unbounded in valid time are marked with $VT_{end} = \infty$ or 'forever', while rectangles that are unbounded in transaction time are marked with $TT_{end} = \text{'now'}$ or 'UTC' (until changed). The following table illustrates this with some examples.

lifesp.	VT_{start}	VT_{end}	TT_{start}	TT_{end}
L1	1985	∞	1989	1991
L2	1985	1988	1991	now
L3	1988	∞	1991	1993
\vdots	\vdots	\vdots	\vdots	\vdots

However, this approach is not unproblematic as each tuple is associated with two transaction time stamps, imposing at most two transactions for each tuple [WE96]. Moreover, one logical lifespan often needs to be described by more than one rectangle. Thus, if each tuple is given together with one rectangle, one logical version of an entity may be represented by several tuples. On the other hand, the advantage of the rectangle approach is that it is simple to implement and that an indexing method for rectangles, the R-tree, already exists [KTF97].

Other methods such as Capelli et al. [CDS95], use a three-level temporal indexing schema. In this method, which is illustrated in Figure 7.1, a complete valid time-slice is stored for each transaction time. An improvement upon this method is implemented by Nascimento et al. [NDE96]. Instead of using simple tables, the first level is implemented using a B-tree, then each n th valid time slice is represented by another B-tree, while the remaining time slices can be accessed through

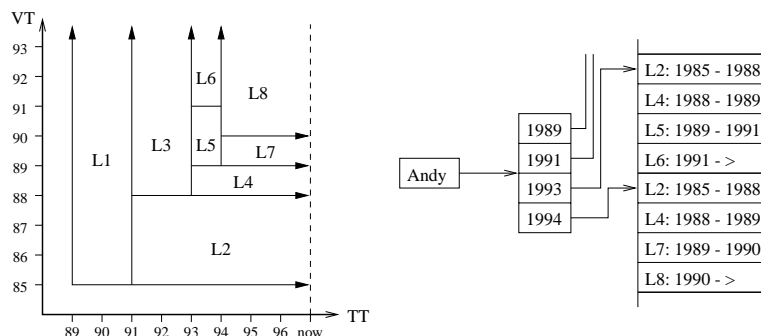


Figure 7.1: The Three-Level Temporal Index

so-called *change patches*. The drawback with this method is that, usually, the whole valid-time history of the entity must be repeated for each transaction.

Aleksic [Ale96] has presented two methods, the *implicit invalidation method* and the *explicit invalidation method* where the corner points of the bitemporal lifespans are stored in a backlog. In the implicit invalidation method, each entry in the backlog contains the surrogate key for the entity and the entity version, one transaction time stamp and two valid time stamps (VT_{start} and VT_{end}). In the explicit invalidation method, each entry contains the surrogate keys along with one transaction time stamp and one valid time stamp in addition to a transaction type (INS or DEL) and a valid change type (BEGIN or END). The drawback with this method is that it requires a linear search through the backlog for each entity version, but the advantage is that it is compact and neat.

Kumar et al. [KTF97] implemented and compared several methods including the *bitemporal interval tree* (BIT) and the *bitemporal R-tree* (BRT). The latter method is inherently two-dimensional, but is based on bitemporal rectangles. However, the problem of two transaction time stamps for each rectangle is solved using a *double tree* (2-R) methodology where the rectangles are stored in two trees; one tree for the rectangles where both transaction time endpoints are known, and one tree for the rectangles where only the left transaction time endpoint is known.

In this article, we introduce a method for representing and indexing bitemporal lifespans called the *BL-tree*. The BL-tree is an acronym for Bitemporal Lifespan

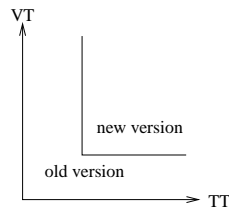


Figure 7.2: An update in bitemporal space

tree or Binary ‘L’-tree (or even BackLog tree). The BL-tree has emerged from the idea that bitemporal lifespans could be described by spatio-hierarchical data structures such as quad-trees or KD-trees. Instead, we will present another way of dividing two-dimensional space: by the shape of the letter ‘L’. Consider that an update of an entity is made at a transaction time TT_u concerning a valid time VT_u . As illustrated in Figure 7.2, in the bitemporal space the new updated data will pertain to the upper right quadrant from (TT_u, VT_u) , i.e. ‘inside’ the L, while the old data pertain to the remaining area ‘outside’ the L.

The main advantage of the BL-tree is that we have a neat representation that more conveniently and effectively represent the shape of bitemporal lifespans. In addition, we get the benefits of hierarchical data structures and the related recursive searching algorithms. The disadvantage of the BL-tree is that one BL-tree can only index all the versions of one logical entity. In other words, we must create one tree for each entity.

The remainder of this article is organized as follows: Section 7.2 presents a simple database model and discuss the nature of bitemporal lifespans, and Section 7.3 presents the data structure for the BL-trees together with the most fundamental algorithms such as inserting into and retrieving data from the tree. The final section concludes with some thoughts concerning possible future developments.

7.2 Updates and Bitemporal Lifespans

Before we move on to discuss the semantics of updates and bitemporal lifespans, we present a simple model in the following section.

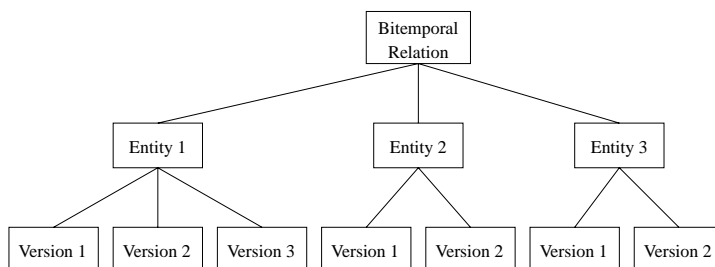


Figure 7.3: A simple bitemporal relation schema

7.2.1 A Conceptual Bitemporal Relation Schema

Conceptually, a relation according to our schema consists of a number of *entities*. Each entity is given by a number of *versions*, where each version is associated with a bitemporal lifespan. Thus, our schema outlines a tuple-level versioning model as illustrated in Figure 7.3. In theory, we could keep all non-temporal attributes at the entity-level and the temporal attributes at the version level. However, non-temporal attributes are just as prone to human errors as temporal attributes and should therefore also be kept at the version-level. Thus, making human errors ‘queriable’ [GN93].

In this article, we assume a discrete linear model of time where a *chronon* denotes the smallest (atomic) interval of time. The time domain T may then be defined as the set of all chronons. A (mono-temporal) *lifespan* is defined as any subset of T , whereas a *temporal interval* is defined as a connected subset of T [JCE⁺94]. A similar distinction can be made for bitemporal lifespans and intervals. In Jensen et al. [JCE⁺94] a bitemporal lifespan is not defined, whereas a *bitemporal interval* is defined as a connected subset (region) of the bitemporal domain T^2 , with sides parallel to the time axis. Nevertheless, in this article we will restrict all *bitemporal lifespans* of an entity version to be a bitemporal interval according to the definition of [JCE⁺94].

Each version of an entity can be associated with a bitemporal lifespan that is the time in which the version is said to exist. In principle, no version of an entity can co-exist with another version of the same entity, i.e. bitemporal lifespans of versions of the same entity cannot overlap. This model is unfortunately somewhat restrictive, since it denies the implementation of a branching model of time (e.g.

one may imagine that an entity have several possible, or alternative, versions in a future time). However, as we will show in Section 7.3.6 this problem can be solved. To illustrate the idea of entities, their versions and their lifespans, we will provide three examples in the next section that should explain some typical features of bitemporal lifespans implemented in the schema suggested above.

7.2.2 Examples

Let us consider a relation in the Widget inc. employee database which has the attributes: name, salary and department. In the sequel, we will describe in detail three of the employees: Caroline, Mark and Yvette.

Caroline

Caroline is our simplest and most ideal example. She worked in the company for two and a half year, from February 1995 till June 1997. The following transactions are associated with her entry:

1. *1995/02/07*: Caroline is added to the database for the first time. She is hired from 1995/02/01 at the administration department with a salary of 35000 per year.
2. *1995/11/28*: Caroline's salary is increased to 37000, effective from 1996/01/01.
3. *1996/04/08*: Caroline will be moved to the shipping department from 1996/05/01.
4. *1997/04/01*: Caroline will be leaving the company at 1997/06/30.

This is a simple example because all updates are made in chronological order and no error is done when the updates are made. This gives the following set of versions for Caroline with the lifespans, which are identified by the surrogate key, shown in Figure 7.4.

lifesp.	name	department	salary
1	Caroline	Administration	35000
2	Caroline	Administration	37000
3	Caroline	Shipping	37000

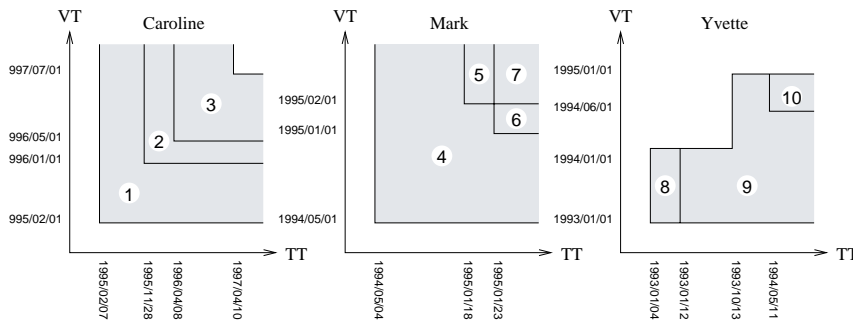


Figure 7.4: Bitemporal lifespans

Mark

Mark is a slightly more complex example as the database administrator failed to register that he was transferred from the Loading department to the Shipping department before his salary was increased from January 1995. So the following transactions are associated with Mark:

1. 1994/05/04: Mark is hired at the loading department with a salary of 30000 from 1994/05/01.
2. 1995/01/18: Mark's salary is increased to 32000 from 1995/02/01.
3. 1995/01/23: Mark is moved to the shipping department from 1995/01/01.

This gives the following set of versions for Mark with the surrogates identifying the version's lifespan in Figure 7.4:

lifesp.	name	department	salary
4	Mark	Loading	30000
5	Mark	Loading	32000
6	Mark	Shipping	30000
7	Mark	Shipping	32000

Note that, despite that we only made three transactions, there are four versions. This is because two versions had to be added for the last transaction. Also, note that version associated with lifespan 5 is a *false* version, because its lifespan has no

open end to the right. This means that (as far as we know now) the version never had root in reality: Mark never worked at the Loading department with a salary of 32000.

Yvette

Yvette is the most complex example, not only because she was supposed to work at the company for only one year, but also because the database administrator misspelled her name when she was first entered in the database. In addition, her period was extended for another year because she was doing such a great job. Hence, we have four transactions associated with Yvette:

1. *1993/01/04*: Yvette is hired at the Marketing department with a salary of 38000 for one year: 1993/01/01 - 1993/12/31. However, her name was misspelled 'Evette'.
2. *1993/01/12*: Yvette's name is corrected from Evette to Yvette.
3. *1993/10/13*: Yvette is hired for another year, till 1994/31/12.
4. *1994/05/11*: Yvette's salary is increased to 40000 for the last six months, from 1994/06/01.

This gives the following set of versions for Yvette with the surrogates identifying the tuple's lifespan as shown in Figure 7.4:

lifesp.	name	department	salary
8	Evette	Marketing	38000
9	Yvette	Marketing	38000
10	Yvette	Loading	40000

Again, we note that version 8 has no open end to the right, and therefore is a false version.

7.2.3 Transaction Types

In the examples above, we have seen a number of different transactions types such as creation, logical delete, updating and correction. Formally, we may have *updates*, *corrections* and *validations*. Corrections may apply to all three types of

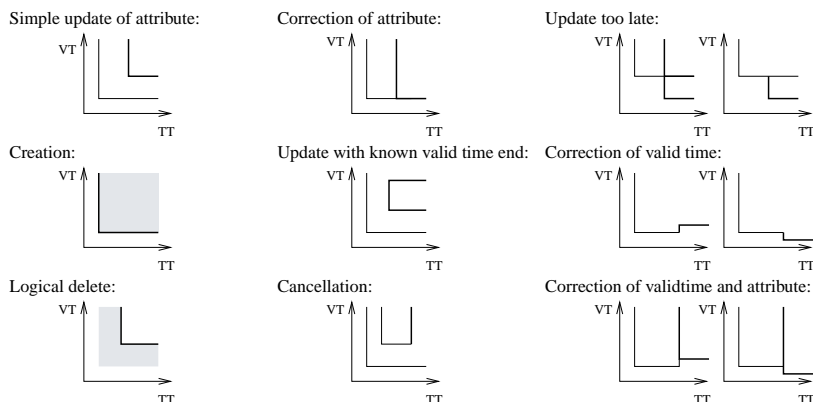


Figure 7.5: Transaction types

transactions; that is, correction of updates, correction of corrections and corrections of validations. A special type of corrections are *cancellations* which cancel previous updates. Validations are important types of transactions, e.g. if an object has not been changed for a long time, it may give the impression that the object had not been updated since the time of change, even when newer sources validate the old information [Lan93].

In the following, we present a taxonomy of some common transaction types as illustrated in Figure 7.5.

- *Update of attribute* — This is the simplest of all transactions which mean to change one or more attribute of an entity to reflect some real world change.
- *Update too late* — Two updates, either on the same attribute or on different attributes, are made in wrong order.
- *Update with known valid time end* — An update where we know for how long the new data will be valid.
- *Creation* — Add a new entity to the database.
- *Logical delete* — An entity has ceased to exist.
- *Cancellation* — An earlier update was completely wrong, as e.g. updating the wrong entity. Cancellations are a special type of corrections.

- *Correction of attribute* — Correct an earlier update where one or more attributes were incorrectly entered.
- *Correction of valid time* — Correct an earlier update where the valid time was incorrectly entered.
- *Correction of attribute and valid time* — Correct an earlier update where both the valid time and at least one attribute were incorrectly entered.

7.3 The BL-tree

In this section we explain the principle of BL-trees and describe the most fundamental algorithms: how to search for a version, how to build a BL-tree and how to query a bitemporal region.

7.3.1 The Principle of the BL-tree

The BL-tree is an extended binary search tree where every node has two children or is a leaf node. One BL-tree represents the bitemporal lifespans of all versions of one single entity. Each change to that entity is represented by a non-leaf node or *tree node* identified by a (TT, VT) -pair. All lifespans are represented by *leaf nodes* and each leaf node is associated with the version pertaining to that lifespan. A tree node divides the bitemporal space in two by the shape of an 'L' with the corner of the L located at (TT, VT) . Every lifespan that exists 'inside' the L (i.e. the upper right quadrant) is in the right subtree, while those who are 'outside' are in the left subtree.

Figure 7.6 shows the BL-trees of Caroline, Mark and Yvette respectively. Tree nodes are shown as square boxes and labeled with capital letters A, B and so forth. The leaf nodes are shown with circles and are numbered according to the examples in the previous section. As indicated in the figure, null-pointers are pointers to bitemporal space for which the entity did not exist, and are shown in the figure with an upside-down 'T', e.g. the first tree node (the one that appears at the lower left of the tree) of each entity will always have a null left pointer because the entity did not exist prior to being added in the database for the first time.

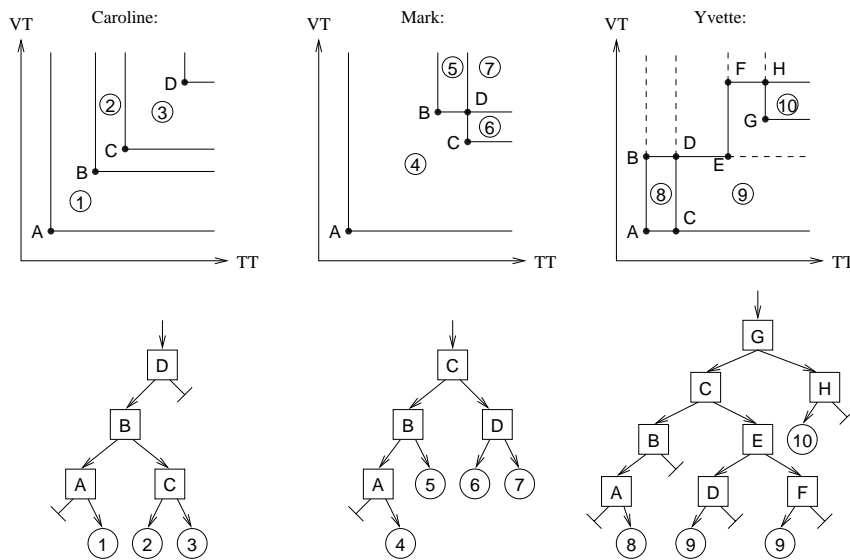


Figure 7.6: A sample of BL-trees

7.3.2 Searching a BL-tree

Once we have built a BL-tree, searching for a specific version is easy, e.g. we may ask, what was the state of Caroline at $VT=1997/01/01$ as far as we knew at $TT=1996/01/01$, e.g. by using TQuel [Sno87] the query would become as follows:

```

retrieve *
where Employee.name = "Caroline"
when 1997/01/01
as of 1996/01/01

```

The result, 2:[Caroline,Administration,37000], is of course wrong in the sense that Caroline worked at the Shipping department at that (valid) time, but at 1996/01/01 we did not know (yet) that she would be transferred from the administration department. Nonetheless, the searching algorithm is a simple binary search algorithm as shown below:

```

function Search( root :pointer; TT, VT :time) → ID

```

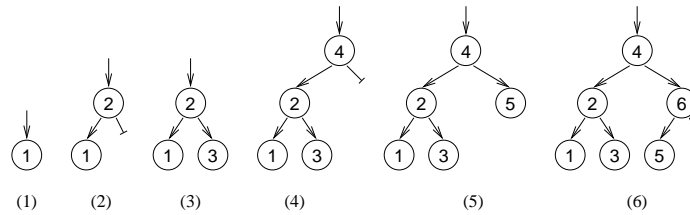


Figure 7.7: The principle of inserting nodes in increasing order into a tree

```

{
  if (root ≠ null)
  {
    if (root is leaf node)
    {
      return root→ID;
    }
    else if ((TT ≥ root→TT) and (VT ≥ root→VT))
    {
      return Search( root→right, TT, VT);
    }
    else
    {
      return Search( root→left, TT, VT);
    }
  }
  return null;
}

```

7.3.3 Building BL-trees

The way we want to build the tree is to always have a tree that is reasonably balanced. A standard insertion algorithm is unfortunately not going to work because new nodes will, with a few exceptions, always appear inside existing nodes, and thus produce a degenerated tree. Instead we could use AVL-trees. But, since we cannot rotate any arbitrary pair of tree nodes in BL-trees, and since the nodes are inserted in an near-sorted order, we decided do develop our own insertion algorithm.

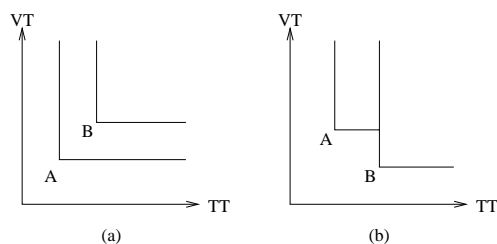


Figure 7.8: Two situations when inserting B in to a BL-tree whose root is A

The way we have implemented the insertion algorithm is by building the tree in a manner illustrated in Figure 7.7. The key issue of the insertion algorithm is the concept of *full trees*. A full binary tree is a tree that is either empty with height $h = 0$ or is not empty with height $h > 0$, and where both subtrees are full with height $h - 1$. Our approach to this algorithm is to provide each tree node with two counters that give the total number of nodes in the left and right subtree respectively.

With this algorithm it is easy to see that, for all nodes, the number of nodes in the left subtree will always be larger or equal to the the number of nodes in the right subtree. Thus, when the number of nodes in the left and right subtrees become equal, the tree is full. We may therefore store the number of nodes in the left and right subtrees as separate variables for each node. However, as Figure 7.8 illustrates, the incremental insertion algorithm works for BL-trees only when new nodes that are inserted are inside existing nodes (a). Since new tree nodes are always going to be inserted with increasing transaction times, the only situation when new nodes are not entirely inside the root node, is when the new node's valid time is smaller than the root's valid time (b). In this case, the root node A is outside the new node B , but B is not inside A . Hence, A can go into B 's left subtree, but not the other way around. Therefore, regardless whether tree A is full or not, B becomes the new root with A as the left subtree.

This violates the concept of counting nodes in the subtrees to determine whether a tree is full or not. Since the number of nodes we may have in a left subtree may be less than its capacity, a tree may prematurely be reported as full when the right subtree is filled with the same number of nodes as the left subtree, but has space for more. Fortunately, there is a way to circumvent this problem by setting *leftCount* to the maximum capacity of the left subtree rather than the actual

number of nodes itself.

Thus, the insertion algorithm becomes as follows:

```

procedure InsertNode( var root :pointer; newNode :pointer)
{
  if (root = null) or (root is leaf node)
  {
    newNode→left := root;
    root := newNode;
  }
  else if (root→leftCount = root→rightCount) or (newNode→VT < root→VT)
  {
    newNode→left := root;
    newNode→leftCount := 2 * root→leftCount + 1;
    root := newNode;
  }
  else
  {
    InsertNode( root→right, newNode);
    root→rightCount++;
  }
}

```

Another feature that we could easily implement is to enforce a maximum size of right subtrees:

```

...
else if (root→leftCount = root→rightCount)
  or (newNode→VT < root→VT)
  or (root→rightCount ≥ maxRightTreeSize)
...

```

This way, the tree will become lopsided which gives shorter access to the most recent data than the older data [Kol93].

The last issue of the insertion algorithm is the leaf nodes. For every insertion of a tree node, a leaf node must be inserted as well. Fortunately, we can insert the new leaf node into the new tree node's right subtree prior to inserting the tree node itself.

7.3.4 Using the Insertion Algorithm

The insertion algorithm outlined above, only inserts one single 'L' into the tree. However, many transactions need multiple nodes to be inserted, such as the update of Mark where he was moved from the Loading department to the Shipping department, and updates of Yvette where the VT_{end} was already known. At least, there are three situations where we need to insert multiple tree nodes: When the update or correction is made too late, when the VT_{end} is known, or sometimes when a valid time is corrected.

The general algorithm for updating an attribute of an entity would therefore be to first find all current versions that are valid during the interval from the valid time concerned by the update and the present time. Then for each of these versions, the attribute is updated and a new version with the updated information is added to the three. In other words:

```

procedure UpdateAttribute( whichAttribute, newValue,  $VT_u$ )
{
  now := CurrentTime();
  foreach version existing during  $[VT_u - \infty)$  as of now
  {
    version  $\rightarrow$  ChangeAttribute( whichAttribute, newValue);
    BLTree  $\rightarrow$  Insert( version, now,  $VT_u$ );
  }
}

```

Similar algorithms can be developed for all types updates, corrections and cancellations.

7.3.5 Query Algorithms

Figure 7.9 shows five different types of bitemporal regions that we may query, e.g. if we would like to know the history of an entity between two valid time points as of a particular transaction time, we may perform a *valid timeslice query* (Q2). A similar query can be performed for a transaction time period at a fixed valid time, i.e. a *transaction timeslice query* (Q3). Ultimately, we may choose to implement only the Q1 query since it may utilize the other four queries by allowing start and end times to be equal and to allow $-\infty$ and ∞ as valid search times.

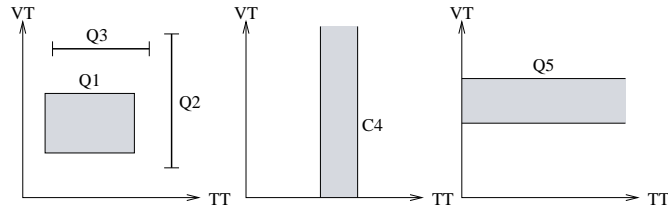


Figure 7.9: Different types of range queries

The bitemporal region query (Q1) looks for all versions whose lifespans intersect a rectangular region bounded by TT_{start} , VT_{start} , TT_{end} and VT_{end} . A general approach for this algorithm is to do an in-fix right-to-left search covering the most recent versions first and the oldest versions at last. It first checks whether the upper right corner (TT_{end}, VT_{end}) of the region is inside the tree node. If so, we recursively search into the right subtree. Then if the lower left corner (TT_{start}, VT_{start}) is outside the tree node, we recursively search into the left subtree. If we are in a leaf node, the version ID of the tree node is returned:

```

procedure SearchRegion( root :pointer;  $TT_{start}, VT_{start}, TT_{end}, VT_{end}$  :time)
{
  if (root  $\neq$  null)
  {
    if (root is leaf node)
    {
      output root $\rightarrow$ ID;
    }
    else
    {
      if ( $TT_{end} \geq$  root $\rightarrow$ TT) and ( $VT_{end} \geq$  root $\rightarrow$ VT)
      {
        SearchRegion( root $\rightarrow$ right,  $TT_{start}, VT_{start}, TT_{end}, VT_{end}$ );
      }
      if ( $TT_{start} <$  root $\rightarrow$ TT) or ( $VT_{start} <$  root $\rightarrow$ VT)
      {
        SearchRegion( root $\rightarrow$ left,  $TT_{start}, VT_{start}, TT_{end}, VT_{end}$ );
      }
    }
  }
}

```

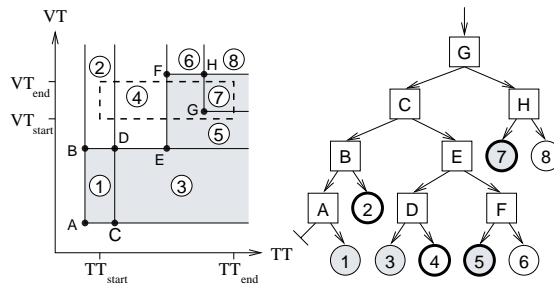



Figure 7.10: A bitemporal region query

}

However, this algorithm needs some adjustments. Consider the situation in Figure 7.10 where we have searched down to tree node D (in the figure, those leaf-nodes that intersect the query region are emphasized with a thick outline). How do we prevent the algorithm from searching into and return leaf node 3? The lower left corner of the search region is outside D, but lifespan 3 is not inside the region. The natural approach would be to shrink the query region to the intersection of the tree node's inside region and the query region. Therefore, when we search into the right subtree from tree node C the part of the query region in lifespan 2 is not passed further down. Hence, the recursive call for right subtree should be as follows:

```

...
if ( $TT_{end} \geq \text{root} \rightarrow TT$ ) and ( $VT_{end} \geq \text{root} \rightarrow VT$ )
{
    SearchRegion( root  $\rightarrow$  right,  $\text{Max}(TT_{start}, \text{root} \rightarrow TT)$ ,
                  $\text{Max}(VT_{start}, \text{root} \rightarrow VT)$ ,  $TT_{end}$ ,  $VT_{end}$ );
}
...

```

The reverse situation is illustrated in Figure 7.11, where we need to prevent the algorithm to search into leaf node 2 from tree node D. Shrinking the query region in a similar manner as for the inside region is not as trivial for the outside region.

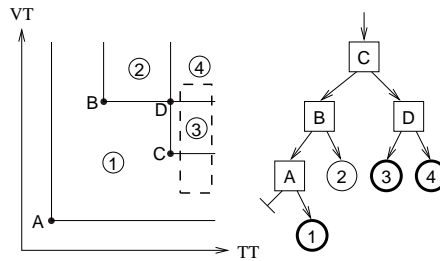


Figure 7.11: A bitemporal region query

Fortunately it turns out that there are only two situations where we need to shrink the query region before searching into the left subtree: the situations where the entire query region is above or to the right of the tree node itself. Hence, before searching into the left subtree we need only do the following adjustment:

```

...
if ( $TT_{start} \leq \text{root} \rightarrow TT$ ) or ( $VT_{start} \leq \text{root} \rightarrow VT$ )
{
  if ( $TT_{start} > \text{root} \rightarrow TT$ )
  {
     $VT_{end} := \text{Min}(\text{root} \rightarrow VT, VT_{end}) - 1;$ 
  }
  if ( $VT_{start} > \text{root} \rightarrow VT$ )
  {
     $TT_{end} := \text{Min}(\text{root} \rightarrow TT, TT_{end}) - 1;$ 
  }
  SearchRegion( root  $\rightarrow$  left,  $TT_{start}$ ,  $VT_{start}$ ,  $TT_{end}$ ,  $VT_{end}$ );
}
...

```

In traditional snapshot databases the only sensible thing to query is for the states of entities. In temporal databases, querying for changes also makes sense. For example, “find all transactions on this entity made during this period of time” or “how often has this entity changed during this period of time”. The BL-tree may support such queries by searching for tree nodes instead of the leaf nodes. With BL-trees, it is a simple task to implement algorithms similar to the one above that

finds all the tree nodes that are inside or intersects a given region. A more complex algorithm though, is to find whether to bitemporal lifespans overlap.

7.3.6 Representing One Single Bitemporal Lifespan

The BL-tree, as we have shown in the previous sections, is based on the entity-version model where one BL-tree describes and indexes the bitemporal lifespans of all the versions of one single entity or object. However, in some applications it is not always possible or practical to identify entities and the versions there of. Moreover, most temporal database models today, like the one presented by Jensen and Snodgrass [JS96] do not represent entities explicitly. Instead each tuple are marked with a surrogate key identifying the entity the tuple is a version of. Thus, two tuples with the same surrogate key are two versions of the same entity.

Furthermore, we may want to implement a branching model of time where one entity may have several possible future (or past) versions. In this case, two versions of the same entity may have overlapping bitemporal lifespans.

Fortunately, with a minor adaptation, the BL-trees can be utilized to suit the requirements of both the above-mentioned models. This is done by letting one single BL-tree represent one single lifespan. This adaptation is illustrated in Figure 7.12 where three examples of bitemporal lifespans and their associated BL-trees are given.

7.4 Concluding Remarks

This article has presented a method of representing bitemporal lifespans of versions of an entity in a bitemporal database called the *BL-tree*. The BL-tree is based on extended binary search trees and combines the excellence of binary search trees together with a new way of dividing two-dimensional space. The method is neat and effective, both in terms of building the tree and to search in the tree, as the tree will always be reasonably balanced.

We have also shown some query algorithms to select versions of an entity given various constraints in bitemporal space. The algorithms have been tested on a variety of examples using an array-based implementation. In this way, it is possible to store and retrieve whole BL-trees in one disk access. Because the BL-tree is based on a hierarchical data structure, we believe that it will perform reasonably

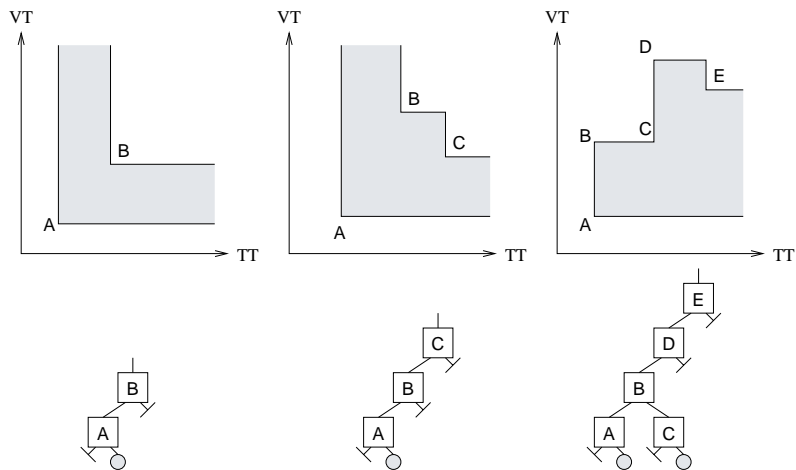


Figure 7.12: Single bitemporal lifespans and their BL-trees

well, but it is maybe a bit awkward to implement in certain contexts, e.g. if the size of the BL-trees exceed one disk block.

Our intension has been to present the idea of BL-trees, and to demonstrate its applicability in bitemporal databases. However, we encourage future research in BL-trees to accommodate performance testing, implementation of very large BL-trees (larger than one disk block), and to develop indexing methods that operates on BL-trees (such as the R-trees [KTF97]). Moreover, we also believe that the principle of BL-trees can be implemented using the principle of AVL-trees, 2-3 trees or B-trees.

However, because of the huge amounts of information associated with many applications, we advocate attribute-level versioning rather than tuple-level versioning. Moreover, in order to implement a branching model of time, we also foresee an approach using BL-trees to represent single lifespans as they were briefly introduced in Section 7.3.6.

TEMPORAL FUZZY REGIONS: CONCEPTS AND MEASUREMENTS

8.1 Introduction

The most common data model used in GIS today is the crisp model, i.e. the features in the natural world are modelled as objects with sharp boundaries and well defined interiors. It is well known that most features in the natural world have fuzzy boundaries or varying degrees of concentration of attributes inside. This means that crisp models often represent a simplification of the underlying geographical phenomena. What kind of boundary things in the natural world may have, depend on a host of material such as topological, functional, temporal and other empirical considerations pertaining to the bounded entities themselves. The complexity of the boundary concept is explained by many authors, for example [Cou96] (p. 55).

The time dimension makes models of changing features a daunting task for designers of geographical information systems (GIS). Since features of the natural world change over time, we can discuss *temporal fuzzy regions*. The present paper defines the concept of temporal fuzzy regions and proposes some measurements of change based on their membership functions.

8.2 Change and Time

Time is a difficult concept because it has no physical characteristics. We cannot grasp it; we only know that it exists. Even this knowledge is based on metaphors or analogies and not on an objective substance. Change and movement, however, are related to time and can be seen as an interpretation of time. Vasiliev [Vas96]

distinguishes between five categories of temporal information: (1) moments, i.e. the dating of an event in space; (2) duration, i.e. the continuance of an occurrence in space; (3) structured time, i.e. the organization or standardization of space by time; (4) time as distance, i.e. the use of time as a measurement of distance; and (5) space as clock, i.e. spatial relations as measures of time. We can identify different types of changes applicable to objects of the real world. We use the term *object* to represent a real world ‘thing’ that might exist as a physical entity, such as an island, a building or a mountain, or an abstract entity, such as the State of Maine and the North Sea [HE97]. Change can result in: (1) change in object identity, (2) change in properties of objects or (3) change in relations among objects.

Hornsby and Egenhofer [HE97] identify a number of operations that either preserve or change the object identity. The operations identified are: (1) on single objects, such as create, destruct or continue; (2) on the transition from an existing object to a new object, such as spawn and metamorphose; (3) on combining single objects, such as merge, generate and mix; (4) on combining composite objects, such as aggregate and amalgamate; (5) on splitting single objects, such as splinter and divide; and (6) on splitting composite objects, such as secede and dissolve.

The properties of objects, which also may change over time, describe the objects and are values that may be obtained through observation or measurement. They may be grouped into value sets and are typically classified as spatial, such as shape, size, dimension, orientation and location, and non-spatial, such as colour and name of object [HE97].

Relations, which represent some association or condition among objects, may also change over time. Such associations may include topological relations, such as meet, inside and overlap, or metrical relations like distance and direction.

8.3 Fuzzy Regions

The membership function μ_A of a fuzzy set \mathcal{A} [Zad65] has the form

$$\mu_A : X \rightarrow [0, 1], \quad (8.1)$$

where $[0, 1]$ denotes the closed interval of real numbers from 0 to 1. A notation that is often used in the literature for defining fuzzy sets with a finite support is

$$\mathcal{A} = \sum_{i=1}^n \mu_i / x_i. \quad (8.2)$$

Similarly, when X is an interval of real numbers, a fuzzy set is often written in the form

$$\mathcal{A} = \int_X \mu_{\mathcal{A}}(x) / x. \quad (8.3)$$

Altman [Alt94] defines a *fuzzy region* as a binary relation on the domain \mathbb{N}^2 . However, in our application of fuzzy regions, we will not restrict fuzzy regions to the domain of natural numbers, but we will also define fuzzy regions on \mathbb{R}^2 .

Definition 8.1. According to [Bjø98], we define a fuzzy region \mathcal{A} as

$$\mathcal{A} = \int \mu_{\mathcal{A}}(x, y) / (x, y), \quad (8.4)$$

for all $(x, y) \in \mathbb{R}^2$, where $\mu_{\mathcal{A}}(x, y) \in [0, 1]$ is the degree of membership at point (x, y) . Note that in Equation 8.4 the \int sign has not the meaning of integral, but is used to represent a fuzzy set with a continuous support.

As pointed out in [Alt94], there are two possible interpretations of each point within a fuzzy region. It may be interpreted as the degree to which a point is inside or a part of some feature; or as the concentration of some attribute belonging to the feature at the particular point. Based on [Bjø98], we give the following definitions.

Definition 8.2. The support of a fuzzy region \mathcal{A} , denoted $\text{supp}\mathcal{A}$, is defined as

$$\text{supp}\mathcal{A} = \{(x, y) \in \mathbb{R}^2 \mid \mu_{\mathcal{A}}(x, y) > 0\}. \quad (8.5)$$

Definition 8.3. The α -cut of a fuzzy region \mathcal{A} , denoted \mathcal{A}_{α} , is defined as

$$\mathcal{A}_{\alpha} = \{(x, y) \in \mathbb{R}^2 \mid \mu_{\mathcal{A}}(x, y) \geq \alpha\} \quad (8.6)$$

for all $\alpha \in [0, 1]$.

Definition 8.4. According to [BjØ98], the boundary of a α -cut of a fuzzy region A , denoted $\partial(A_\alpha)$, is defined according to the usual boundary definition of point set topology.

We also define a measurement concept for fuzzy regions. [Alt94] defines the distance between to fuzzy regions as a fuzzy set. We apply a slightly different view basing our measurement concept on the α -cut of the fuzzy region.

Definition 8.5. Assume a fuzzy region A and a function $\phi_A(\alpha)$ that computes a metrical measure of A based on the α -cut A_α of A . The α -measure of A is a fuzzy set \mathcal{M} defined as

$$\mathcal{M} = \int \mu_{\mathcal{M}}(\phi_A(\alpha)) / \phi_A(\alpha) \quad (8.7)$$

for all $\alpha \in (0, 1]$, where $\mu_{\mathcal{M}}(\phi_A(\alpha)) = \alpha$. Note that $\alpha \in (0, 1]$ means $0 < \alpha \leq 1$.

8.4 Temporal Fuzzy Regions

8.4.1 Basic Concepts

Definition 8.6. We define a period of time as a subset of \mathbb{R} , i.e. $T \subset \mathbb{R}$.

Definition 8.7. A temporal fuzzy region A is defined as

$$A = \mu_A(x, y, t) / x, y, t \quad (8.8)$$

for all $t \in T$ and for all $(x, y) \in \mathbb{R}^2$.

Definition 8.8. The membership function of a temporal fuzzy region A will be called the lifespan function of A .

Definition 8.9. Assume the temporal fuzzy region A and its lifespan function $\mu_A(x, y, t)$ defined for a period of time T . The snapshot of A at time $t = c \in T$ is a fuzzy region defined by

$$A_{t=c} = \int \mu_{A_{t=c}}(x, y) / x, y \quad (8.9)$$

for all $(x, y) \in \mathbb{R}^2$, where $\mu_{A_{t=c}}(x, y) = \mu_A(x, y, t)$ for $t = c$.

By the previous definitions we have introduced the concept of temporal fuzzy regions. This concept will be demonstrated by some examples at the end of this paper. First, we introduce the concept of *change* of fuzzy regions.

8.4.2 Some Measurements of Change of Fuzzy Regions

As a measure of the change in the membership function of a fuzzy region, we define three binary difference operators based on snapshots of temporal fuzzy regions.

Definition 8.10. Assume a temporal fuzzy region \mathcal{A} and its two snapshots $\mathcal{A}_{t=1}$ and $\mathcal{A}_{t=2}$. The growth of \mathcal{A} from $t = 1$ to $t = 2$ is a fuzzy region defined as

$$\Delta = \mathcal{A}_{t=1} \oplus \mathcal{A}_{t=2} = \int \mu_{\Delta}(x, y) / x, y \quad (8.10)$$

for all $(x, y) \in \mathbb{R}^2$, where

$$\mu_{\Delta}(x, y) = \begin{cases} |\mu_{\mathcal{A}_{t=2}}(x, y) - \mu_{\mathcal{A}_{t=1}}(x, y)| & \text{if } \mu_{\mathcal{A}_{t=2}}(x, y) > \mu_{\mathcal{A}_{t=1}}(x, y), \\ 0 & \text{otherwise.} \end{cases} \quad (8.11)$$

Definition 8.11. Assume a temporal fuzzy region \mathcal{A} and its two snapshots $\mathcal{A}_{t=1}$ and $\mathcal{A}_{t=2}$. The shrink of \mathcal{A} from $t = 1$ to $t = 2$ is a fuzzy region defined as

$$\Delta = \mathcal{A}_{t=1} \ominus \mathcal{A}_{t=2} = \int \mu_{\Delta}(x, y) / x, y \quad (8.12)$$

for all $(x, y) \in \mathbb{R}^2$, where

$$\mu_{\Delta}(x, y) = \begin{cases} |\mu_{\mathcal{A}_{t=2}}(x, y) - \mu_{\mathcal{A}_{t=1}}(x, y)| & \text{if } \mu_{\mathcal{A}_{t=2}}(x, y) < \mu_{\mathcal{A}_{t=1}}(x, y), \\ 0 & \text{otherwise.} \end{cases} \quad (8.13)$$

Definition 8.12. Assume a temporal fuzzy region \mathcal{A} and its two snapshots $\mathcal{A}_{t=1}$ and $\mathcal{A}_{t=2}$. The no-change of \mathcal{A} from $t = 1$ to $t = 2$ is a fuzzy region defined as

$$\Delta = \mathcal{A}_{t=1} \odot \mathcal{A}_{t=2} = \int \mu_{\Delta}(x, y) / x, y \quad (8.14)$$

for all $(x, y) \in \mathbb{R}^2$, where

$$\mu_{\Delta}(x, y) = 1 - |\mu_{\mathcal{A}_{t=2}}(x, y) - \mu_{\mathcal{A}_{t=1}}(x, y)|. \quad (8.15)$$

We also define the change of the α -measurement of temporal fuzzy regions.

Definition 8.13. *Assume a temporal fuzzy region \mathcal{A} and its two snapshots $\mathcal{A}_{t=1}$ and $\mathcal{A}_{t=2}$. The change of the α -measure of \mathcal{A} from $t = 1$ to $t = 2$ is a fuzzy set defined as*

$$\begin{aligned}\Delta &= \mathcal{M}_{\mathcal{A}_{t=1}} \circ \mathcal{M}_{\mathcal{A}_{t=2}} \\ &= \int \mu_{\Delta}(\alpha) / [\phi_{\mathcal{A}_{t=2}}(\alpha) - \phi_{\mathcal{A}_{t=1}}(\alpha)]\end{aligned}\quad (8.16)$$

for all $\alpha \in (0, 1]$, where $\mu_{\Delta}(\alpha) = \alpha$.

8.5 Examples

8.5.1 Change of the snapshot of \mathcal{A}

We assume the temporal fuzzy region ‘Forest’ and its two snapshots 1985 and 1995 shown in Figures 8.1 and 8.2. The growth, shrink and no-change of ‘Forest’ from 1985 to 1995 is computed from Equations 8.10, 8.12 and 8.14 respectively and shown in Figure 8.3.

8.5.2 Change of the Distance from p to \mathcal{A}

We assume the temporal fuzzy region ‘Forest’, denoted by \mathcal{A} , and its two snapshots 1985 and 1995 shown in Figures 8.1 and 8.2. We define the α -measure \mathcal{M} on the basis of a function $\phi_{\mathcal{A}}(\alpha)$ that computes the shortest distance from a point p to the boundary $\partial(\mathcal{A}_{\alpha})$ of the α -cut of \mathcal{A} . From Equation 8.7 we compute a distance measure for ‘Forest 1985’ as

$$\mathcal{M}_1 = \sum 0.2/2.6 + 0.4/2.9 + 0.6/3.6 + 0.8/3.6 + 1.0/5.7 \quad (8.17)$$

and for ‘Forest 1995’

$$\mathcal{M}_2 = \sum 0.2/3.1 + 0.4/5.8 + 0.6/4.5 + 0.8/3.6 + 1.0/4.5. \quad (8.18)$$

The change of the distance measure from 1985 to 1995 is computed from Equation 8.16

$$\begin{aligned}\Delta &= \mathcal{M}_{\mathcal{A}_{t=1}} \circ \mathcal{M}_{\mathcal{A}_{t=2}} \\ &= \sum 0.2/0.5 + 0.4/2.9 + 0.6/0.9 + 0.8/0.0 + 1.0/(-1.2). \quad (8.19)\end{aligned}$$

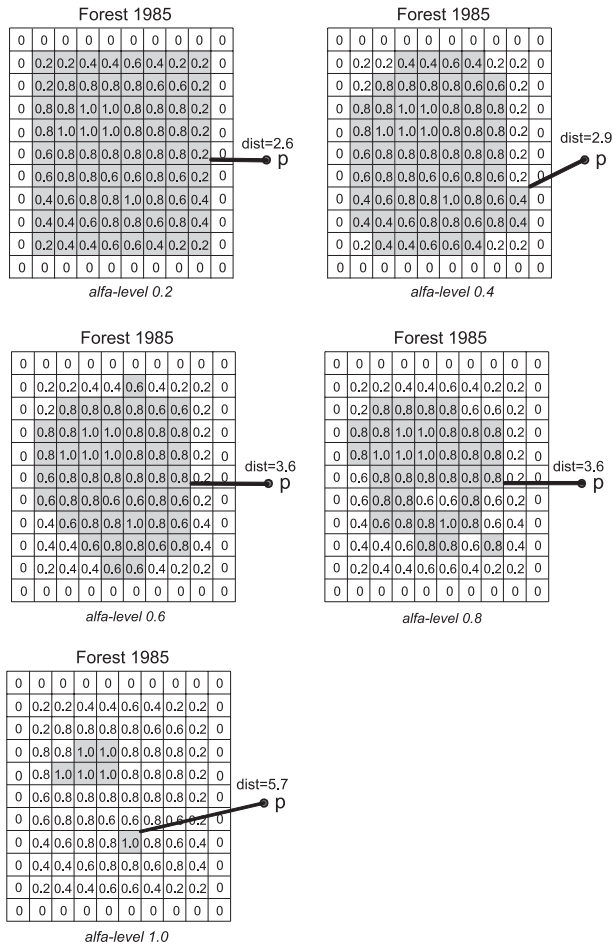


Figure 8.1: Different α -levels of fuzzy region 'Forest 1985'. The figure also shows the distance from point p to the different α -levels of 'Forest 1985'.

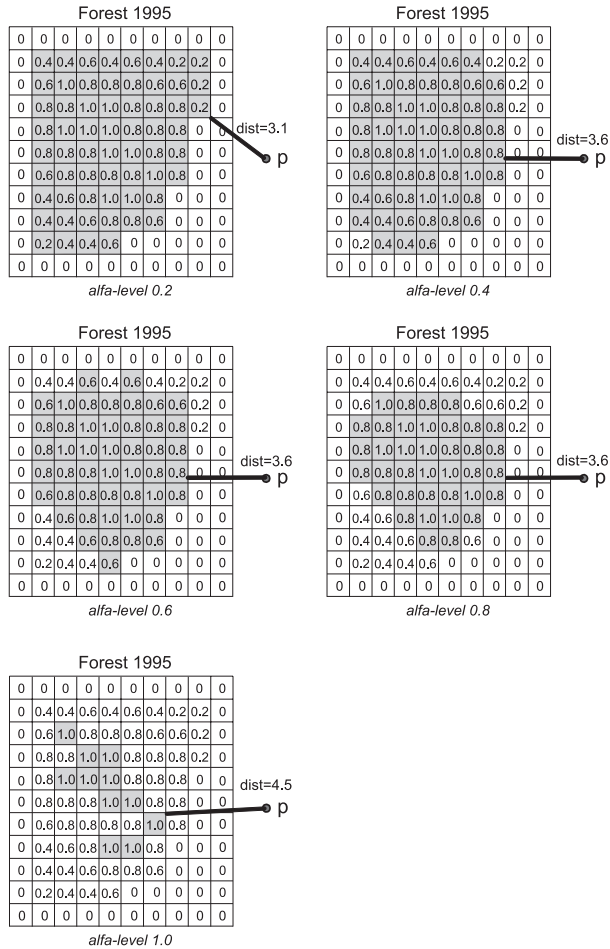


Figure 8.2: Different α -levels of fuzzy region 'Forest 1995'. The figure also shows the distance from point p to the different α -levels of 'Forest 1995'.

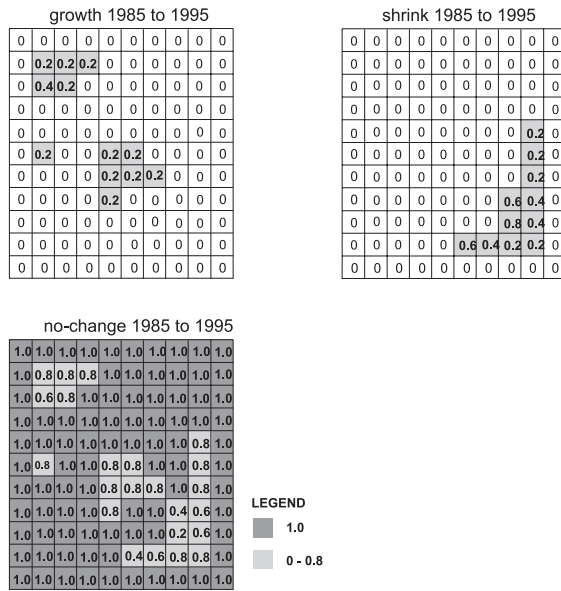


Figure 8.3: Change of the temporal fuzzy region 'Forest' from 1985 to 1995

8.5.3 Change of the Area of \mathcal{A}

We assume the temporal fuzzy region ‘Forest’ and its two snapshots 1985 and 1995 shown in Figures 8.1 and 8.2. Now we define the α -measure \mathcal{M} on the basis of a function $\phi_{\mathcal{A}}(\alpha)$ that computes the area of the α -cut \mathcal{A}_{α} . From Equation 8.7 we compute the area measure for ‘Forest 1985’ as

$$\mathcal{M}_1 = \sum 0.2/72 + 0.4/59 + 0.6/48 + 0.8/35 + 1.0/6 \quad (8.20)$$

and for ‘Forest 1995’

$$\mathcal{M}_2 = \sum 0.2/61 + 0.4/56 + 0.6/47 + 0.8/37 + 1.0/11. \quad (8.21)$$

The change of the measure from 1985 to 1995 is computed from Equation 8.16 as

$$\begin{aligned} \Delta &= \mathcal{M}_{\mathcal{A}_{t=1}} \circ \mathcal{M}_{\mathcal{A}_{t=2}} \\ &= \sum 0.2/(-11) + 0.4/(-3) + 0.6/(-1) + 0.8/2 + 1.0/5. \end{aligned} \quad (8.22)$$

8.6 Conclusions and Future Work

This paper has introduced a concept termed temporal fuzzy region. Based on snapshots of temporal fuzzy regions we have proposed three difference operators that model the linguistic expressions: (1) growth, (2) shrink, and (3) no-change of a fuzzy region \mathcal{A} from time t_1 to time t_2 . We have also introduced an α -measure that is applicable to compute the change of metrical properties of fuzzy regions, such as the area of a region and the distance from a point to a region. Some examples demonstrate the proposed operators and measurements.

Since most of the features in the real world have fuzzy characteristics, future research in geographical information science should pay more attention to the development of a theoretical basis for modelling change in fuzzy objects as well as change in topological and geometrical relations among them over time.

CONCLUSION

This work has been initiated by the increasing need to handle historical information in GIS. However, the models used in GIS and database systems today are not well suited to handle information that is varying over both space and time. The aim of this thesis has therefore been to develop a model or framework, such that the characteristics of spatiotemporal information are preserved in the model.

In this thesis, four different frameworks have been investigated. By using *conceptual modelling languages* developers can describe spatiotemporal information and systems dealing with such information at the conceptual level. However, these languages poorly capture the internal structure of such information. A more mathematical approach is therefore useful.

One approach is to use directed acyclic graphs to capture successor relationships between objects or states (versions) of objects. Using *history graphs*, the temporal relationships between both states and the changes between them are preserved. Using set-theory, an even more comprehensive and formal framework can be defined. The *temporal set-theory*, in turn, is associated with constructions that describe the *tropology* of information.

Finally, the concepts from the temporal set-theory are used to propose a temporal object-oriented model for geographic information. With these models, we consider ourselves a significant step closer to the implementation of a temporal GIS.

The accomplishments of this thesis can be summarized as follows:

- A large body of research from a wide range of research fields has been studied and integrated.
- A set of methodologies to study and model spatiotemporal information has

been described.

- The temporal structure of information has been investigated and the need to explicitly describe *changes* has been pointed out.
- A formal and fundamental framework based on the set-theory, aimed at describing temporal information and the changes therein, has been proposed.
- Concepts and properties of bitemporal information have been studied and uncovered.

From this standpoint, there are various aspects that should undergo further investigation:

- The temporal and topological relationships between states and changes together with the topological relationships that exist perpendicular to the time line need to be integrated, probably by using graph theory. In this thesis, we have been studying the topological relationships with only limited concern to the topological relationships.
- The properties of the temporal set-theory and the associated topology need to be investigated from a mathematical point of view. In particular, extensions akin to general and algebraic topology should be of interest. One idea that we have been playing around with, is to use a vector-analogy in order to describe the change from one state to another (as a vector may describe the change in position from one point to another).

Many organizations today need to handle temporal information, and temporal information is necessary in order to obtain knowledge that current systems are unable to deduce. Research on temporal GIS and spatiotemporal modelling is therefore important. Hopefully, this thesis is a valuable contribution to this research.

SET-THEORY AND RELATED TOPICS

A.1 The Naive Set-Theory

The set-theory is one of the most fundamental theories of mathematics, and has provided a basis for other mathematical theory as well as for many areas of computer science. It is expected that the reader is familiar with this theory, but it is included here for the sake of comprehensiveness and for the sake of clarifying the terminology and notation used throughout this thesis.

A problem with the set-theory was discovered by Russel. Let S be the set of all sets which are not members of themselves, in other words $S = \{x \mid x \notin x\}$. Then S is neither a member of itself nor not a member of itself. In other words $S \in S \iff S \notin S$. This paradox can be avoided by setting up a number of axioms. However, in this section we will briefly review the original, but naive set-theory, that is not based on such an axiomatic schema.

A.1.1 Basic Notation

A *set* is a collection of objects called *elements* or *members*, and a set is said to *contain* its members. To indicate that a is a member of the set A we write

$$a \in A, \tag{A.1}$$

and to indicate that a is not a member of A we write

$$a \notin A. \tag{A.2}$$

A set has no inherent ordering of its members, and a set cannot contain the same member twice. If a set \mathbf{A} has members a , b and c we write

$$\mathbf{A} = \{a, b, c\} \quad (\text{A.3})$$

We may also use the set builder notation,

$$\mathbf{A} = \{x \mid x \text{ has some property } P\} \quad (\text{A.4})$$

The number of members of a set \mathbf{A} denoted $|\mathbf{A}|$ is called the *cardinality* of \mathbf{A} . If $|\mathbf{A}|$ is a finite number, then \mathbf{A} is called a *finite set*, otherwise it is called an *infinite set*. A set that has no members is called the *empty set* and is denoted by the symbol \emptyset .

If all members of a set \mathbf{A} , are also a members of a set \mathbf{B} , then \mathbf{A} is said to be a *subset* of \mathbf{A} and is written

$$\mathbf{A} \subseteq \mathbf{B} \quad (\text{A.5})$$

Two sets \mathbf{A} and \mathbf{B} are *equal* if both $\mathbf{A} \subseteq \mathbf{B}$ and $\mathbf{B} \subseteq \mathbf{A}$.

If \mathbf{A} is a set, then the *power set* $P(\mathbf{A})$ is the set of all subsets of \mathbf{A} . In other words,

$$P(\mathbf{A}) = \{x \mid x \subseteq \mathbf{A}\} \quad (\text{A.6})$$

Both the empty set \emptyset and the set \mathbf{A} itself are members of $P(\mathbf{A})$. A set whose members are themselves sets are often referred to as a *family* of sets.

A.1.2 Operations on Sets

There are four main operators on sets. These are *union*, *intersection*, *difference* and *complement*. They are defined as follows: let \mathbf{A} and \mathbf{B} be sets, then the *union* of \mathbf{A} and \mathbf{B} , denoted $\mathbf{A} \cup \mathbf{B}$, is defined as

$$\mathbf{A} \cup \mathbf{B} = \{x \mid x \in \mathbf{A} \vee x \in \mathbf{B}\}, \quad (\text{A.7})$$

the *intersection* of \mathbf{A} and \mathbf{B} , denoted $\mathbf{A} \cap \mathbf{B}$, is defined as

$$\mathbf{A} \cap \mathbf{B} = \{x \mid x \in \mathbf{A} \wedge x \in \mathbf{B}\}, \quad (\text{A.8})$$

the *difference* of \mathbf{A} and \mathbf{B} , denoted $\mathbf{A} \setminus \mathbf{B}$ is defined as

$$\mathbf{A} \setminus \mathbf{B} = \{x \mid x \in \mathbf{A} \wedge x \notin \mathbf{B}\} \quad (\text{A.9})$$

and the complement is defined as

$$\overline{\mathbf{A}} = \{x \mid x \notin \mathbf{A}\}. \quad (\text{A.10})$$

A.1.3 Functions

Functions and relations are fundamental concepts of the set-theory. Databases and programming languages are heavily based on these concepts.

Let \mathbf{A} and \mathbf{B} be sets. A *function* f from \mathbf{A} to \mathbf{B} , is an assignment of exactly one element $b = f(a)$ of \mathbf{B} , to each element a of \mathbf{A} . A function from \mathbf{A} to \mathbf{B} is written

$$f : \mathbf{A} \rightarrow \mathbf{B}. \quad (\text{A.11})$$

The set \mathbf{A} is called the *domain* of f while the set \mathbf{B} is called the *co-domain* of f .

For functions, a number of properties are defined. A function f is said to be *one-to-one* or *injective* iff (if and only if), $f(x) = f(y)$ implies that $x = y$. It is called *onto* or *surjective* iff for every element b of \mathbf{B} there is an element a of \mathbf{A} with $f(a) = b$. If a function f is both one-to-one and onto, it is called a *one-to-one correspondence* or a *bijection*. In that case, there exist an *inverse function* $f^{-1} : \mathbf{B} \rightarrow \mathbf{A}$ such that if $b = f(a)$ then $a = f^{-1}(b)$.

A special set that is commonly defined is the *universal set* \mathbf{U} , which contain all the elements of concern in a particular context. Then, for each subset \mathbf{A} in \mathbf{U} we can define a *characteristic function*

$$\mu_{\mathbf{A}} : \mathbf{U} \rightarrow \{0, 1\} \quad (\text{A.12})$$

such that if $\mu_{\mathbf{A}}(x) = 1$ then $x \in \mathbf{A}$, and if $\mu_{\mathbf{A}}(x) = 0$ then $x \notin \mathbf{A}$.

An important application of functions is that the output of one function can be used as input to another function. This, way we may form *compositions* of two or more functions. Let $f : \mathbf{A} \rightarrow \mathbf{B}$ and $g : \mathbf{B} \rightarrow \mathbf{C}$ be two functions such that the codomain of f is the domain of g , then the composition $f \circ g$ is defined by

$$(f \circ g)(a) = f(g(a)) \quad (\text{A.13})$$

Often, we need to define sets of functions into some particular domain \mathbf{B} from a particular codomain \mathbf{A} . Such a set is indicated by the symbol $\mathbf{B}^{\mathbf{A}}$ and is defined by

$$\mathbf{B}^{\mathbf{A}} = \{f \mid f : \mathbf{A} \rightarrow \mathbf{B}\} \quad (\text{A.14})$$

A.1.4 Relations and Cartesian Product

We now move on to another important aspect of the set-theory which is called *relations*. Before defining a relation, we define the *Cartesian product* of two or more sets. If $\mathbf{A}_1, \dots, \mathbf{A}_n$ are n sets, then the Cartesian product $\mathbf{A}_1 \times \dots \times \mathbf{A}_n$ is the set of all *n-tuples* $\langle a_1, \dots, a_n \rangle$ such that $a_i \in \mathbf{A}_i$ for $i = 1, \dots, n$. If $n = 2$ the members of the Cartesian product are called *ordered pairs*.

Note that a Cartesian product contains all possible permutations of elements from the sets, thus the number of elements in the Cartesian product equals the product of the number of elements in all its sets.

A *relation* is a set that contain relationships between members of different sets. Let $\mathbf{A}_1, \dots, \mathbf{A}_n$ be n sets, then an *n-ary relation* on $\mathbf{A}_1, \dots, \mathbf{A}_n$ is a subset of the Cartesian product $\mathbf{A}_1 \times \dots \times \mathbf{A}_n$. The sets $\mathbf{A}_1, \dots, \mathbf{A}_n$ are called the *domains* of the relation and n is called the *degree*.

If $n = 2$, the relation is called a *binary relation* from \mathbf{A}_1 to \mathbf{A}_2 , and if $\mathbf{A}_1 = \mathbf{A}_2 = \mathbf{A}$ the relation is called a *relation on the set A*. If \mathbf{R} is a binary relation from \mathbf{A} to \mathbf{B} , it is common to write aRb to denote that $\langle a, b \rangle$ is a member of \mathbf{R} . Note that relations are also sets, but their members are pairs or n -tuples.

There are many interesting properties of relations on a single set. In the following, some of these properties are defined: Let \mathbf{R} be a binary relation on a set \mathbf{A} . The relation \mathbf{R} is called *reflexive* if $\langle a, a \rangle \in \mathbf{R}$ for every element a of \mathbf{A} , and it is called *irreflexive* if $\langle a, a \rangle \notin \mathbf{R}$. It is called *symmetric* if $\langle a, b \rangle \in \mathbf{R}$ implies that $\langle b, a \rangle \in \mathbf{R}$, *antisymmetric* if $\langle a, b \rangle \in \mathbf{R}$ implies that $a = b$, and *asymmetric* if $\langle a, b \rangle \in \mathbf{R}$ implies that $\langle b, a \rangle \notin \mathbf{R}$. The relation \mathbf{R} is called *transitive* if whenever $\langle a, b \rangle \in \mathbf{R}$ and $\langle b, c \rangle \in \mathbf{R}$, implies that $\langle a, c \rangle \in \mathbf{R}$.

Based on these properties, a number of relation types can be identified, e.g. an *equivalence relation* is a relation that is reflexive, symmetric and transitive. In practice, an equivalence relation partitions a set into *equivalence classes* such that each element of a class is related to all the other members of that class, an no member of any other class.

A relation is called a *partial ordering* if it is reflexive, antisymmetric and transitive. A set \mathbf{A} together with a partial ordering \mathbf{R} is called a *partially ordered set* or a *poset* and is denoted (\mathbf{A}, \mathbf{R}) . In practice, a partial ordering puts an order relation between some pairs of elements in \mathbf{A} , but not all. A relation that puts an order relation between all pairs, is called a *total order* or a *linear order*. It is common to use the symbol \preceq for orderings.

A.2 Point-Set Topology

An application of the set-theory is obtained by defining sets whose members are *points* in some space. The set of real numbers \mathbb{R} is an example of such a space; the Euclidean plane \mathbb{E}^2 is another such space that is of great interest to us. In the sequel we use the symbol \mathbf{S} to indicate the space of concern. The point-set theory provides the basis of what we know as the point-set topology. In the point-set theory the sets are called *spaces* or *metric spaces*, and in order to be called a metric space, the set must be non-empty and a metric must be set up in the set.

A.2.1 Metrics and Metric Spaces

A *metric* is set up in \mathbf{S} by associating every pair of points p_1 and p_2 of \mathbf{S} a non-negative number $\delta(p_1, p_2)$ called the *distance* between them, that satisfies the following three axioms:

$$\delta(p_1, p_2) = \delta(p_2, p_1) \quad (\text{A.15})$$

$$\delta(p_1, p_2) = 0 \iff p_2 = p_1 \quad (\text{A.16})$$

$$\delta(p_1, p_2) + \delta(p_2, p_3) \geq \delta(p_1, p_3) \quad (\text{A.17})$$

To every point p in a space \mathbf{S} we can define a *spherical neighbourhood* $N(p, r)$ which is the set of points in \mathbf{S} that is within the positive distance r from p . In other words:

$$N(p, r) = \{q \in \mathbf{S} \mid \delta(p, q) < r\}, \quad (r > 0) \quad (\text{A.18})$$

We will now study subsets of \mathbf{S} and certain properties of these sets such as openness and closedness. Indeed, open and closed sets in metric spaces correspond

to the concepts of whether the points on the boundary of a set are included in the set or not. However, the formal definitions of such sets may be a bit more elaborate as we will see in the following section.

A.2.2 Open and Closed Sets

Let A be a subset of S . A point p is called an *interior point* of a set A if there is a neighbourhood $N(p, r) \subset A$. The *interior* of a set A , denoted $\mathcal{I}(A)$ is then the set of all interior points of A . If all points in A are interior points, then A is called an *open set*.

A point p of S is called a *limit point* of A if every neighbourhood $N(p, r)$ contains at least one point in A different from p . A limit point of A is not necessarily a member of A , but all interior points are also limit points. A point in A that is not a limit point is called an *isolated point*. The set of all limit points of a set A is called the *derived set* and is denoted A' . If we compute the union of A and A' we obtain the *closure* of A , denoted $\mathcal{K}(A)$. In other words:

$$\mathcal{K}(A) = A \cup A' \tag{A.19}$$

If $A = \mathcal{K}(A)$ then A is said to be a *closed set*.

A point p of S is called a *boundary point* of A if it is not an interior point of A , but every neighbourhood of p has at least one point in common with A . The *boundary* of A , denoted $\mathcal{B}(A)$ is then defined as

$$\mathcal{B}(A) = \mathcal{K}(A) \setminus \mathcal{I}(A) \tag{A.20}$$

A boundary point is either a limit point or an isolated point and a limit point is either a boundary point or an interior point. A point that is neither an interior point nor a boundary point of A , is called an *exterior point* of A .

A.2.3 Some Types and Properties of Point Sets

In general, we may classify sets in metric space according to a number of different properties. For example, a set is said to be *bounded* if there exist a finite upper bound for the distance between any two pairs of points in the set. A set is said to be *connected* if it is possible to join any pair of points in the set with a curve such that all points on the curve are also members of the set. A set that is both connected

and open is also referred to as a *domain*. A set is said to be *convex* if it is possible to join any pair of points in the set with a straight line segment such that all points on the segment are also in the set. From these definitions it is clear that a convex set is also a connected set.

A *region* is an open connected set (i.e. a domain), plus some or all of its boundary points. If we add all the boundary points to a domain, we obtain a *closed region*.

A set of points \mathbf{A} is called a *discrete set* if it has no limit points, in other words, if all the points of \mathbf{A} are isolated points. A set \mathbf{A} is *dense* in a set \mathbf{B} containing \mathbf{A} , if every spherical neighbourhood that contains a point of \mathbf{B} also contains a point of \mathbf{A} . In other words, a set \mathbf{A} is dense in \mathbf{B} if

$$\mathcal{K}(\mathbf{A}) = \mathbf{B}, \quad (\text{A.21})$$

e.g. the set of rational numbers is dense in the set of real numbers.

A set \mathbf{A} of points is *nowhere dense* if

$$\mathcal{I}(\mathcal{K}(\mathbf{A})) = \emptyset \quad (\text{A.22})$$

A set \mathbf{A} of points is *dense-in-itself*, if $\mathbf{A} \subseteq \mathbf{A}'$, that is, if every point of \mathbf{A} is a limit point. A set that is both dense-in-itself and closed is called a *perfect* set.

A.2.4 Topological Spaces and Homeomorphisms

A set \mathbf{S} along with a family \mathbf{T} of open subsets of \mathbf{S} is said to be a *topology* if the subsets in \mathbf{T} satisfy the following three properties:

1. The set \mathbf{S} and the the empty set \emptyset are members of \mathbf{T} .
2. Whenever \mathbf{A} and \mathbf{B} are members of \mathbf{T} , so is $\mathbf{A} \cap \mathbf{B}$.
3. Whenever \mathbf{A} and \mathbf{B} are members of \mathbf{T} , so is $\mathbf{A} \cup \mathbf{B}$.

A space \mathbf{S} for which a topology \mathbf{T} has been specified is called a *topological space*.

A function $f : \mathbf{S}_1 \rightarrow \mathbf{S}_2$ from a space \mathbf{S}_1 to a space \mathbf{S}_2 is *continuous* provided that for each open set \mathbf{X} in \mathbf{S}_2 the inverse image

$$f^{-1}(\mathbf{X}) = \{p \in \mathbf{S}_1 | f(p) \in \mathbf{X}\} \quad (\text{A.23})$$

is open in S_1 . A one-to-one correspondence $f : S_1 \rightarrow S_2$ for which both f and f^{-1} are continuous is called a *homeomorphism*. The topologies T_1 and T_2 formed on the topological spaces S_1 and S_2 respectively, are called *homeomorphic* if there exist a homeomorphism between the elements of S_1 and S_2 .

If S_1 and S_2 are topological spaces with topologies T_1 and T_2 respectively, the topology on the product $S_1 \times S_2$ can be defined by taking, as a base, the collection of all open sets of the form $A_1 \times A_2$ where $A_1 \in T_1$ and $A_2 \in T_2$. This is called the *product topology* for $S_1 \times S_2$.

A.3 Fibre Bundles

In mathematics, a bundle is, roughly speaking, a topological space with another topological space embedded in each point. By definition a *bundle* is a triple $\langle E, B, \pi \rangle$ where E is a topological space called the *total space*, B is a topological space called the *base space* and π is a continuous and surjective mapping

$$\pi : E \rightarrow B, \quad (\text{A.24})$$

known as the *projection mapping*. For each point p in B , the set of points $\pi^{-1}(p)$ is called the *fibre* over p .

The mathematicians have been playing around with this construct for some time, using different topological spaces such as the Möbius strip or the Klein bottle. However, it is only the simplest bundle that is of interest to us, viz. the Cartesian product bundle $\langle B \times F, B, \pi \rangle$, where F and B are topological spaces and $\pi : B \times F \rightarrow B$ is defined as

$$\pi(p_1, p_2) = p_1 \quad (\text{A.25})$$

for all $p_1 \in B$ and $p_2 \in F$.

If we set the space F to the Euclidean plane \mathbb{E}^2 and the base space B to the time domain \mathbb{T} , a Cartesian product bundle indeed can be used to describe a region (or point-set) that is varying over time. Then the region that is changing over time, will be given by the function

$$\pi^{-1}(t) \quad (\text{A.26})$$

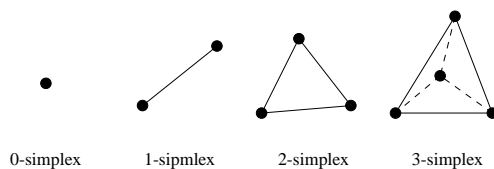


Figure A.1: Simplexes

A.4 Algebraic Topology

A.4.1 Simplexes and Complexes

In general, a polyhedron is like a connected set in a metric space. However, we define them in terms of vertices, line segments, triangles, tetrahedrons and so on, joined together in a so-called *modulo 2 homology*. A number of definitions follows:

A d -simplex σ^d , spanned by a set of vertices $\{p_0, \dots, p_d\}$ in \mathbb{E}^d is the set of points contained in the convex hull of the vertices $\{p_0, \dots, p_d\}$. Figure A.1 shows the simplexes for $d = 0, \dots, 3$.

A simplex σ^d is a *face* of a simplex σ^n if $d \leq n$, and each vertex of σ^d is also a vertex of σ^n . The faces of σ^n other than σ^n itself are called *proper faces*. Two simplexes σ^m and σ^n are *properly joined* if, and only if their intersection $\sigma^m \cap \sigma^n$ is a face of both σ^m and σ^n .

A *geometric complex* (or simplicial complex or complex) is a finite family \mathbf{K} of properly joined simplexes, such that each face of a member of \mathbf{K} is also a member of \mathbf{K} . The *dimension* of \mathbf{K} is the largest positive integer r such that \mathbf{K} has an r -simplex. The union of the members of \mathbf{K} is called the *geometric carrier* of \mathbf{K} or the *polyhedron* associated with \mathbf{K} and is denoted $|\mathbf{K}|$.

If there is a geometric complex \mathbf{K} whose carrier $|\mathbf{K}|$ is homeomorphic to a topological space S , then S is said to be a *triangulable space*, and the complex \mathbf{K} is called a *triangulation* of S .

Two simplexes s_1 and s_2 of a complex \mathbf{K} are *connected* if either of the following conditions is satisfied:

1. the two simplexes share a common boundary.
2. there is a sequence $\sigma_1, \dots, \sigma_p$ of p 1-simplexes such that σ_i and σ_{i+1} share a boundary for $i = 1, \dots, p-1$, and such that the sequence itself share a

boundary with s_1 and s_2 .

Thus, a complex \mathbf{K} is connected if every pair of simplexes of \mathbf{K} are connected.

A.4.2 Oriented Simplexes and Complexes

An *oriented n -simplex*, with $n \geq 1$, is obtained by ordering the vertices of the n -simplex. The equivalence class of simplexes with an even permutation of the ordering is called a *positively oriented simplex*, denoted $+\sigma^n$, while those simplexes with odd permutations of the ordering are called *negatively oriented simplexes*, denoted $-\sigma^n$. An *oriented complex* is thus obtained by assigning an orientation to each of the simplexes of the complex, e.g. given a 2-simplex with the vertices p_0, p_1, p_2 and the ordering $p_0 < p_1 < p_2$, then we have the ordered simplexes:

$$+\sigma^2 = \langle p_0, p_1, p_2 \rangle, \langle p_1, p_2, p_0 \rangle, \langle p_2, p_0, p_1 \rangle \quad (\text{A.27})$$

$$-\sigma^2 = \langle p_2, p_1, p_0 \rangle, \langle p_1, p_0, p_2 \rangle, \langle p_0, p_2, p_1 \rangle \quad (\text{A.28})$$

If \mathbf{K} is an oriented complex with simplexes σ^{p+1} and σ^p . Then an *incidence number* $[\sigma^{p+1}, \sigma^p]$ can be defined such that:

$$[\sigma^{p+1}, \sigma^p] = \begin{cases} 0 & \text{if } \sigma^p \text{ is not a face of } \sigma^{p+1} \\ 1 & \text{if the orientation of } \sigma^p \text{ agrees with } \sigma^{p+1} \\ -1 & \text{if the orientation of } \sigma^p \text{ disagrees with } \sigma^{p+1} \end{cases} \quad (\text{A.29})$$

For example, Figure A.2 shows a small complex, and we may identify a number of incidence numbers:

$$\begin{aligned} [\langle abc \rangle, \langle ab \rangle] &= 1 \\ [\langle abc \rangle, \langle ac \rangle] &= -1 \\ [\langle abc \rangle, \langle bd \rangle] &= 0 \\ [\langle ab \rangle, \langle a \rangle] &= -1 \\ [\langle ab \rangle, \langle b \rangle] &= 1 \\ [\langle ab \rangle, \langle c \rangle] &= 0 \end{aligned}$$

The matrix

$$\eta(p) = [\eta_{i,j}(p)], \quad (\text{A.30})$$

where $\eta_{i,j}(p) = [\sigma^{p+1}, \sigma^p]$, is called the *p th incidence matrix of \mathbf{K}* .

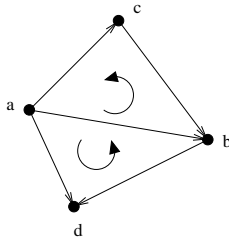


Figure A.2: A simplicial 2-complex

A.4.3 Simplicial Homology Groups

In general, the homology group of a complex describes the arrangement of the simplexes in the complex, and thereby telling us about ‘holes’ in the associated polyhedron. First some definitions:

A p -dimensional chain or p -chain is a formal finite sum of simplexes σ_i^p

$$d_p = \sum g_i \sigma_i^p \quad (\text{A.31})$$

where the expression $g\sigma^p$ is called an *elementary p -chain* and the index i ranges over all p -simplexes in a complex \mathbf{K} , e.g. based on Figure A.2 we may identify the p -chains

$$c_0 = g_0 \langle a \rangle + g_1 \langle b \rangle + g_2 \langle c \rangle \quad (\text{A.32})$$

$$c_1 = h_0 \langle ab \rangle + h_1 \langle bc \rangle + h_2 \langle ac \rangle \quad (\text{A.33})$$

The set of p -chains of a complex \mathbf{K} is called the p -chain group of \mathbf{K} and is denoted $C_p(\mathbf{K})$

The *boundary* of an elementary p -chain $g\sigma^p$ is defined by

$$\mathcal{B}(g\sigma^p) = \sum [\sigma^p, \sigma_i^{p-1}] g \sigma_i^{p-1} \quad (\text{A.34})$$

for σ_i^{p-1} in \mathbf{K} . The boundary of a 0-chain is 0. Thus, the boundary of a p -chain c_p is defined as

$$\mathcal{B}(c_p) = \sum \mathcal{B}(g\sigma^p) = \sum_i \sum_j [\sigma_j^p, \sigma_i^{p-1}] g_j \sigma_i^{p-1} \quad (\text{A.35})$$

A special type of p -chains is called p -cycles. A p -chain c_p is a p -cycle if

$$\mathcal{B}(c_p) = 0. \quad (\text{A.36})$$

and it is called a p -boundary if there is a $(p + 1)$ -chain c_{p+1} such that

$$\mathcal{B}(c_{p+1}) = c_p \quad (\text{A.37})$$

The subgroup $Z_p(\mathbf{K})$ of p -cycles of $C_p(\mathbf{K})$ is called the p -cycle group of \mathbf{K} , and the set of set $B_p(\mathbf{K})$ of p -boundaries of $C_p(\mathbf{K})$ is called the p -boundary group of \mathbf{K} .

Two p -cycles w_p and z_p in s complex \mathbf{K} are *homologous*, denoted $w_p \sim z_p$, if there is a $(p + 1)$ -chain c_{p+1} such that

$$\mathcal{B}c_{p+1} = w_p - z_p. \quad (\text{A.38})$$

This relation of homology for p -cycles is an equivalence relation and partitions $Z_p(\mathbf{K})$ into *homology classes*.

Finally we can define the p -dimensional homology group of an oriented complex \mathbf{K} as the quotient group

$$H_p(\mathbf{K}) = \frac{Z_p(\mathbf{K})}{B_p(\mathbf{K})} \quad (\text{A.39})$$

A.4.4 Polyhedrons and Manifolds

The simplicial complexes define a set of polyhedrons built from simplexes in a triangulation. However, in constructive geometry it is also common to build objects from faces with more that three sides.

A term that most users of GIS are familiar with is the *polygon*. A polygon is a closed plane figure with n sides. If all sides and angles are equal, the polygon is called regular. Regular polygons can either be a convex polygon such as a simple pentagon or star polygon where two sides may intersect.

A *rectilinear polyhedron* in \mathbb{E}^3 is a solid bounded by properly joined *convex polygons*. The boundary polygons are called *faces*, the intersection between two faces are called *edges* while the intersection between two or more edges are called *vertices*. If a rectilinear polyhedron is homeomorphic to a sphere, then it is called a *simple polyhedron*, if the faces of the polyhedron are regular polygons with same

size and shape, it is called a *regular polyhedron*. There are totally five regular convex polyhedrons.

In GIS, regular polyhedron are very rare, but there are some results that can be of use in GIS, e.g. Euler's theorem. Euler's theorem states that if V is the number of vertices E is the number of edges, and F is the number of faces of a simple polyhedron, then the following equation holds:

$$V - E + F = 2 \quad (\text{A.40})$$

A generalization of Euler's formula applies to 2-manifolds that have faces with holes:

$$V - E + F - H = 2(C - G), \quad (\text{A.41})$$

where H is the number of holes in the faces, G is the number of holes that pass through the object and C is the number of separate components (parts) of the objects.

A.5 Fuzzy Set Theory

A problem with the traditional set-theory is that it is based on boolean logic where a fact is either true or false. Thus, an element is either a member of a set, or it is not. In the *fuzzy set theory*, objects can be partly members of a set.

A.5.1 Basic Definitions

The degree of membership in a set is taken from the unit interval $\mathbb{I} = [0, 1]$ and a fuzzy set μ can thereby be defined as a function from a universe \mathbf{U} into \mathbb{I}

$$\mu : \mathbf{U} \rightarrow \mathbb{I} \quad (\text{A.42})$$

that for each element x in \mathbf{U} returns the a number in the range $[0, 1]$ representing the *grade of membership* x has in μ . If $\mu(x) = 0$, then the element x has no membership in μ , if $\mu(x) = 1$, then x has a full membership in μ . To avoid confusion with traditional sets, we use the retronym *crisp set* to denote a traditional set as described in Section A.1. The set of all fuzzy sets over a universe \mathbf{U} can thereby be identified by the symbol $\mathbb{I}^{\mathbf{U}}$.

The *support* of a fuzzy set μ , denoted $\text{supp}(\mu)$, is the crisp set obtained by assembling all elements whose grade of membership in μ is larger than 0. In other words,

$$\text{supp}(\mu) = \{x \in \mathbf{U} \mid \mu(x) > 0\} \quad (\text{A.43})$$

However, we often need to define a crisp set of those elements that has a membership grade larger than some value α . This leads us to the definition of the so-called α -*cut*, which is denoted \mathbf{A}_α :

$$\mathbf{A}_\alpha(\mu) = \{x \in \mathbf{U} \mid \mu(x) \geq \alpha\} \quad (\text{A.44})$$

In most situations, the underlying universal set \mathbf{U} is a metric space. If so, we can say that a fuzzy set μ is *convex* if for all λ in $[0, 1]$

$$\mu(\lambda x_1 + (1 - \lambda)x_2) \geq \min\{\mu(x_1), \mu(x_2)\} \quad (\text{A.45})$$

for all x_1 and x_2 in \mathbf{U} .

For a finite fuzzy set μ the *cardinality* of μ denoted $|\mu|$ is defined as

$$|\mu| = \sum_{x \in \mathbf{U}} \mu(x) \quad (\text{A.46})$$

whereas the *relative cardinality* is defined as

$$\|\mu\| = \frac{|\mu|}{|\mathbf{U}|} \quad (\text{A.47})$$

A.5.2 Basic Operations on Fuzzy Sets

If μ_A and μ_B are fuzzy sets, then the *intersection* of μ_A and μ_B is defined by

$$(\mu_A \cap \mu_B)(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad (\text{A.48})$$

the *union* is defined by

$$(\mu_A \cup \mu_B)(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad (\text{A.49})$$

the *difference* is defined by

$$(\mu_A \setminus \mu_B)(x) = \max\{\mu_A(x) - \mu_B(x), 0\}, \quad (\text{A.50})$$

and the complement of a fuzzy set μ_A is defined by

$$\overline{\mu_A}(x) = 1 - \mu_A(x). \quad (\text{A.51})$$

A.5.3 Fuzzy Relations

Let \mathbf{A} and \mathbf{B} be crisp sets, then a fuzzy relation ρ from \mathbf{A} to \mathbf{B} is the fuzzy set

$$\rho : \mathbf{A} \times \mathbf{B} \rightarrow \mathbb{I} \quad (\text{A.52})$$

Then, if $a \in \mathbf{A}$ and $b \in \mathbf{B}$ then the value $\rho(a, b)$ is a number that determines the grade of which the elements a and b are related to each other.

Let ρ be a binary fuzzy relation on a crisp set \mathbf{A} . If, for all a in \mathbf{A} , $\rho(a, a) = 1$ then ρ is called *reflexive*, if for all a, b in \mathbf{A} , $\rho(a, b) = \rho(b, a)$ then ρ is called *symmetric*. If for all a, b, c in \mathbf{A}

$$\rho(a, c) \geq \min\{\rho(a, b), \rho(b, c)\} \quad (\text{A.53})$$

then ρ is called *max-min transitive*, and if

$$\rho(a, c) \geq \rho(a, b) \cdot \rho(b, c) \quad (\text{A.54})$$

then ρ is called *max-product transitive*.

A.5.4 Fuzzy Numbers and Fuzzy Regions

A *fuzzy number* η is a convex fuzzy set, whose underlying space \mathbf{U} is the set of real numbers \mathbb{R} , such that

1. there is exactly one $x \in \mathbb{R}$, called the mean value, such that $\eta(x) = 1$
2. $\eta(x)$ is piecewise continuous.

A fuzzy number η is called *positive* if $\eta(x) = 0$ implies that $x < 0$ and *negative* if $\eta(x) = 0$ implies that $x > 0$

A *fuzzy region* is a set whose underlying space \mathbf{U} is the Euclidean plane \mathbb{E}^2 . However, a fuzzy regions does not need to satisfy the conditions above, and it does not need be convex.

A.6 Graphs

There are many phenomena in the real world that exhibit a graph-like structure, e.g. a road network or a computer network. In the following, we will define a number of graph types and associated concepts.

A.6.1 Graph Types

A *simple graph* is a structure $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ that consist of a nonempty set of *vertices* \mathbf{V} , and a set of unordered pairs of distinct elements of \mathbf{V} called *edges*. In a simple graph we can have at most one edge between any distinct pair of vertices.

A graph that can contain more than one edge between any pair of vertices is called a *multigraph* and is defined as a structure $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ consisting of a set \mathbf{V} of vertices, a set \mathbf{E} of edges, and a function $f : \mathbf{E} \rightarrow \{\{v_1, v_2\} \mid v_1, v_2 \in \mathbf{V}, v_1 \neq v_2\}$ that associates every edge with a pair of vertices. The edges e_1 and e_2 are called *multiple* or *parallel edges* if they connect the same vertices, i.e. if $f(e_1) = f(e_2)$.

A multigraph that may contain edges from a vertex to itself is called a *pseudograph*. An edge of a pseudograph is called a *loop* if $f(e) = \{v\}$ for some v in \mathbf{V} .

Simple graphs, multigraphs and pseudographs are undirected graphs which means that the edges have no inherent direction. However, in a road network, we may have some one-way streets. A *directed graph* or *digraph* $\langle \mathbf{V}, \mathbf{E} \rangle$ consist of a set of vertices \mathbf{V} and a binary relation \mathbf{E} on \mathbf{V} containing edges. Similarly a *directed multigraph* $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ consists of a set of vertices \mathbf{V} and a set of edges \mathbf{E} , and a function $f : \mathbf{E} \rightarrow \mathbf{V} \times \mathbf{V}$. The edges e_1 and e_2 are called *multiple edges* if $f(e_1) = f(e_2)$.

A.6.2 Terminology of Graphs

Two vertices v_1 and v_2 in an undirected graph \mathbf{G} are called *adjacent* or *neighbours* in \mathbf{G} , if $\{v_1, v_2\}$ is an edge in \mathbf{G} . If $e = \{v_1, v_2\}$, the edge e is called *incident with* the vertices v_1 and v_2 . The edge e is also said to *connect* v_1 and v_2 . The vertices v_1 and v_2 are called *endpoints of the edge* $\{v_1, v_2\}$.

The *degree* of a vertex in an undirected graph is the number of edges incident with it, except that a loop at a vertex contributes twice to the degree of that vertex. The degree of the vertex v is denoted $deg(v)$.

When $\langle v_1, v_2 \rangle$ is an edge in a directed graph \mathbf{G} , v_1 is said to be *adjacent to* v_2 and v_2 is said to be *adjacent from* v_1 . The vertex v_1 is called the *initial vertex* of $\langle v_1, v_2 \rangle$ and v_2 is called the *terminal* or *end vertex* of $\langle v_1, v_2 \rangle$.

A simple graph is called *bipartite* if its vertex set \mathbf{V} can be partitioned into two disjoint nonempty sets \mathbf{V}_1 and \mathbf{V}_2 such that every edge in the graph connects a vertex in \mathbf{V}_1 and a vertex in \mathbf{V}_2 .

A *subgraph* of a graph $G = \langle V, E \rangle$ is a graph $H = \langle W, F \rangle$ where $W \subseteq V$ and $F \subseteq E$. The *union* of two simple graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$, denoted $G_1 \cup G_2$, is the simple graph with vertex set $V_1 \cup V_2$ and edge set $E_1 \cup E_2$.

The simple graphs $G_1 = \langle V_1, E_1 \rangle$ and $G_2 = \langle V_2, E_2 \rangle$ are *isomorphic* if there is a one-to-one correspondence f , called an *isomorphism*, from V_1 to V_2 with the property that a and b are adjacent in G_1 if and only if $f(a)$ and $f(b)$ are adjacent in G_2 , for all a and b in V_1 .

A *path* from a to b in an undirected graph is a sequence of one or more edges

$$(\{v_0, v_1\}, \{v_1, v_2\}, \dots, \{v_{n-1}, v_n\}) \quad (\text{A.55})$$

in G where $v_1 = a$ and $v_n = b$. When the graph is simple, we denote the path by its vertex sequence (v_0, v_1, \dots, v_n) and has the number of edges in the path n is called the *length* of the path.

A path that begins and ends in the same vertex is called a *cycle* or a *circuit*, and it is called a *simple path* if it does not contain the same edge more than once. An undirected graph is called *connected* if there is a path between every pair of distinct vertices of the graph. A *directed acyclic graph*, also known as a *DAG*, is a directed graph that contain no cycles.

BIBLIOGRAPHY

- [AB90] K. K. Al-Taha and R. Barrera. Temporal data and GIS: An overview. In *Proceedings of GIS/LIS '90*, November 1990.
- [Ala97] Suad Alagic. A temporal constraint system for object-oriented databases. In Volker Gaede et al., editors, *Constraint Databases and Their Applications, Second International Workshop on Constraint Database Systems*, volume 1191 of *Lecture Notes in Computer Science*, pages 208–218, Delphi, Greece, January 1997.
- [Ale96] Mario Aleksic. Incremental computation methods in valid & transaction time databases. Master's thesis, Universität Stuttgart, Fakultät Informatik Germany, December 1996.
- [All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11), November 1983.
- [All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2), 1984.
- [Alt94] David Altman. Fuzzy set theoretic approaches for handling imprecision in spatial analysis. *Int. Journal of Geographical Information Systems*, 8(3):271–289, 1994.
- [ASS94] Khaled K. Al-Taha, Richard T. Snodgrass, and Micahel D. Soo. Bibliography on spatiotemporal databases. *Int. Journal of Geographic Information Systems*, 8(1):95–103, 1994.

- [Aus62] J.L. Austin. *How to do things with words*. Harward University Press, Cambrigde MA, 1962.
- [Bar93] D. Bartlett. Space, time, chaos and coastal GIS. In *Proceedings of the 16th International Cartographic Conference*, pages 539–551. International Cartographic Association, 1993.
- [Bel91] Aaron Beller. Spatial/temporal events in a GIS. In *Proceedings of GIS/LIS'91*, pages 766–775, Bethesda, Maryland, 1991.
- [Ben83] J. F. A. K. Van Benthem. *The Logic of Time*. D. Reidel Publishing Company, 1983.
- [Ber81] J. Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter, Berlin, 1981.
- [BF96] Peter A. Burrough and Andrew U. Frank. *Geographic Objects with Indeterminate Boundaries*. Taylor & Francis, 1996.
- [BFA91] R. Barrera, A. Frank, and K. K. Al-Taha. Temporal relations in geographic information systems: A workshop at the University of Maine. *SIGMOD Record*, 20(3):85–91, September 1991.
- [BFG96] E. Bertino, E. Ferrari, and G. Guerrini. A formal temporal object-oriented data model. In *Advances in Database Technologies (EDBT'96)*, volume 1057 of *Lecture Notes in Computer Science*, pages 342–356, Avignon, France, 1996.
- [Bir79] Graham Birtwistle. *DEMOS A System for Discrete Event Modelling on Simula*. MacMillan, 1979.
- [Bjø95] Jan T. Bjørke. Digital kartografi, del 1. Compendium in course Digital Cartography, The Norwegian Institute of Technology, 1995.
- [Bjø98] Jan T. Bjørke. Fuzzy regions: Topological properties and binary topological relations. *Submitted Geoinformatica*, 1998.
- [BJS96] Michael Böhlen, Christian S. Jensen, and Bjørn Skjellaug. Spatio-temporal database support for legacy applications. Technical report, Time Center, July 1996. TR-20.

- [Boo96] Grady Booch. *Object Solutions: Managing the Object-Oriented Project*. Addison-Wesley, 1996.
- [Bou89] C. Boutoura. Line generalization using spectral techniques. *Cartographica*, 26(3 & 4), 1989.
- [BR99] Jan Terje Bjørke and Agnar Renolen. Temporal fuzzy regions: Concepts and measurements. In *Proceedings of the 7th Scandinavian Research Conference on Geographical Information Systems ScanGIS99*, pages 53–62, Aalborg, Denmark, June 1999. Aalborg Universitetsforlag.
- [Bra93] Svein Erik Bratsberg. *Evolution and Integration of Classes in Object-Oriented Databases*. PhD thesis, The Norwegian Institute of Technology, June 1993.
- [BS93] Olav Mork Bjørnås and David Skogan. Multimodels and spatio-temporal modeling in object-oriented GIS. Master's thesis, Norwegian Institute of Technology, 1993.
- [BVH96] L. Becker, A. Voigtmann, and K. H. Hinrichs. Temporal support for geo-data in object-oriented databases. In *Database and Expert Systems Applications (DEXA'96)*, volume 1134 of *Lecture Notes in Computer Science*, pages 79–93, Zürich, Switzerland, 1996.
- [BZ82] J. Ben-Zvi. *The Time Relational Model*. PhD thesis, Computer Science Department, UCLA, 1982.
- [CC87] J. Clifford and A. Croker. The historical relational data model (HRDM) and algebra based on lifespans. In *Proceedings, Third International Conference on Data Engineering*, 1987.
- [CDS95] A. Cappelli, C. De Castro, and M. R. Scalas. A modular history-oriented access structure for bitemporal relational databases. In *SOFSEM'95: Theory and Practice of Informatics*, volume 1012 of *Lecture Notes in Computer Science*, page 369ff, 1995.
- [CDT93] Eliseo Clementini, Paolino Di Felice, and Cianfranco Torlone. Experimenting the usability of an O-ODBMS for the representation of

- topology on GISs. In *Proceedings of the EGIS'93*, volume II, pages 1421–1430, Genoa, Italy, 1993.
- [CHB92] Derek Coleman, Fiona Hayes, and Stephen Bear. Introducing objectcharts or how to use statecharts in object-oriented design. *IEEE Transactions on software engineering*, 18(1), January 1992.
- [Che76] P. P. Chen. The entity-relationship model: Towards a unified view of data. *ACM Transactions on Database Systems*, 1(1), 1976.
- [CLL96] Gwo-Dong Chen, Yeong-Hsen Lee, and Chen-Chun Liu. Extending OODB with behavioral temporal management capability. In *Proceedings of the 20th Annual IEEE International Computer Software and Applications Conference (COMPSAC'96)*, pages 361–366, 1996.
- [CM98] Tao Cheng and Martien Molenaar. A process-oriented spatio-temporal data model to support physical environmental modeling. In *Proceedings, 8th International Symposium on Spatial Data Handling*, pages 419–430, Vancouver, Canada, July 1998.
- [Cou96] Helen Couclelis. Towards an operational typology of geographic entities with ill-defined boundaries. In Peter A. Burrough and Andrew U. Frank, editors, *Geographic Objects with Indeterminate Boundaries*, GISDATA II, pages 45–55. Taylor&Francis, Great Britain, 1996.
- [CT95] C. Claramunt and M. Thériault. Managing time in GIS: An event-oriented approach. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 23–42, Zurich, Switzerland, September 1995. Proceedings of the International Workshop on Temporal Databases, Springer Verlag.
- [CT96] C. Claramunt and M. Thériault. Toward semantics for modelling spatio-temporal processes in GIS. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 2.27–2.41, 1996.
- [DeM78] T. DeMarko. *Structured Analysis and System Specification*. Yourdon, New York, 1978.

- [DGS95] C. De Castro, F. Grandi, and M. R. Scalas. On schema versioning in temporal databases. In S. Clifford and A. Tuzhilin, editors, *Recent Advances in Temporal Databases*, pages 272–294, Zurich, Switzerland, September 1995. Proceedings of the International Workshop on Temporal Databases, Springer Verlag.
- [DGS97] Cristina De Castro, Fabio Grandi, and Maria Rita Scalas. Schema versioning for multitemporal relational databases. *Information Systems*, 22(5):249–290, 1997.
- [DN66] Ole-Johan Dahl and Kristen Nygaard. SIMULA, an ALGOL-based simulation language. *Communications of the ACM*, 9(9):671–678, September 1966.
- [DTR96] T. Devogele, J. Trevesian, and L. Raynal. Building a multi-scale database with scale-transition relationships. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, 1996.
- [EA92] Max J. Egenhofer and Khaled K. Al-Taha. Reasoning about gradual changes of topological relationships. In *International Conference on GIS — From Space to Territory: Theories and Methods of Spatio-Temporal reasoning in Geographic Space*, 1992.
- [EGSV97] Martin Erwig, Ralf Hartmunt Güting, Makus Schneider, and Michalis Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. Technical Report 224, FernUniversität Gesamthochschule in Hagen, 1997.
- [EHS93] J.M Edwards and B. Henderson-Sellers. A graphical notation for object-oriented analysis and design. *Journal of Object-Oriented Programming*, 5(9):53–74, February 1993.
- [EP96] Robert Edsall and Donna Peuquet. A graphical user interface for the interaction of time into GIS. Technical report, Department of Geography, Penn State University, 1996.
- [ESG97] Martin Erwig, Markus Schneider, and Ralf Hartmunt Güting. Temporal and spatio-temporal data models and their expressive power.

- Technical Report 225, FernUniversität Gesamthochschule in Hagen, 1997.
- [ESG98] Martin Erwig, Markus Schneider, and Ralf Hartmut Güting. Temporal objects for spatiotemporal data models and a comparison of their representations. In *Advances in Database Technologies, ER'98 Workshop on new Database Technologies for Collaborative Work Support and Spatio-Temporal Data Management (NewDB'98)*, volume 1552 of *Lecture Notes in Computer Science*, pages 454–465, Singapore, November 1998.
- [Fra94] Andrew U. Frank. Qualitative temporal reasoning in GIS –ordered time scales. In *Advances in GIS research*, pages 410–430, 1994. Sixth international Symposium on Spatial Data Handling.
- [FSG92] Farshad Fotouhi, Abad A. Shah, and William Grosky. Complex objects in the temporal object system. In *ICCI'92. Fourth International Conference on Computing and Information*, pages 381–384, Toronto, Canada, May 1992. IEEE Computer Science Press.
- [Gad93a] S. Gadia. Ben-Zvi's pioneering work in relational temporal databases. In *Temporal Databases: Theory, Design, and Implementation*, pages 202–207. Benjamin/Cummings, 1993.
- [Gad93b] Shashi. K Gadia. Parametric databases: seamless integration of spatial, temporal, belief and ordinary data. *SIGMOD Record*, 22(1):15–20, March 1993.
- [Gal97] A. Galton. Continuous change in spatial regions. In *Spatial Information Theory: A Theoretical basis or GIS (COSIT'97)*, volume 1329 of *Lecture Notes in Computer Science*, pages 1–13, Pennsylvania, USA, 1997.
- [GN93] Shashi K. Gadia and Sunil S. Nair. Temporal databases: A prelude to parametric data. In *Temporal Databases: Theory, Design, and Implementation*, pages 28–66. Benjamin/Cummings, 1993.
- [GS78] C. Gane and T. Sarson. *Structured system analysis: Tools and techniques*. Prentice Hall, 1978.

- [GV94] R. Galetto and F. Viola. Time as the fourth dimension of the topographic databases. In *Mapping and Geographic Information Systems*, pages 297–303. International Society for Photogrammetry and Remote Sensing, 1994.
- [HA92] Kauzo Hiraki and Yuichiro Anzai. Towards acquiring spatio-temporal knowledge from sensor data. In *International Conference on GIS – From Space to Territory: Theories and Methods of Spatio-Temporal reasoning in Geographic Space*, pages 368–378, 1992.
- [Ham94] Torill Hamre. An object-oriented conceptual model for measured and derived data varying in 3d space and time. In *Advances in GIS research. Sixth international Symposium on Spatial Data Handling (SDH'94)*, pages 868–881, 1994.
- [Ham95] Torill Hamre. *Development of Semantic Spatio-temporal Data Models for Integration of Remote Sensing and in situ data in a Marine Information System*. PhD thesis, University of Bergen, Norway, 1995.
- [Har87] David Harel. Statechart: A visual formalism for complex systems. *Science of computer programming*, 9(8):231–274, 1987.
- [HCJ94] Joanne N. Halls, David J. Cowen, and John R. Jensen. Predictive spatio-temporal modelling in GIS. In *Advances in GIS research. Sixth international Symposium on Spatial Data Handling (SDH'94)*, pages 431–448, 1994.
- [HE97] K. Hornsby and M. J. Egenhofer. Qualitative representation of change. In *Spatial Information Theory: A Theoretical basis or GIS (COSIT'97)*, volume 1329 of *Lecture Notes in Computer Science*, pages 15–33, Pennsylvania, USA, 1997.
- [HR85] P.H. Hughes and L.A. Rønningen. Demos activity digrams. Notes in Course on Discrete Simulation, Norwegian Institute of Technology, 1985.
- [HT96] Thanasis Hadzilacos and Nectaria Tryfona. Logical data modelling for geographical applications. *Int. Journal of Geographical Information Systems*, 10(2), 1996.

- [HT97] Thanasis Hadzilacos and Nectaria Tryfona. An extended entity-relationship model for geographic application. *SIGMOD Record*, 26(1):24–29, September 1997.
- [Hun88] Gary J. Hunter. Non-current data and geographical information systems: a case for data retention. *Int. Journal of Geographic Information Systems*, 2(3):281–286, 1988.
- [HW90] Gary J. Hunter and Ian P. Williamson. The development of a historical digital cadastral database. *Int. Journal of Geographic Information Systems*, 4(2):169–179, 1990.
- [JCE⁺94] Christian S. Jensen, J. Clifford, R. Elmasri, S. K. Gadia, P. Hayes, S. Jajodia, C. Dyreson, F. Grandi, W. Kafer, N. Kline, N. Lorentzos, Y. Mitsopoulos, A. Montanari, D. Nonen, E. Peressi, B. Pernici, J. F. Roddick, N. L. Sarda, M. R. Scalas, A. Segev, R. T. Snodgrass, M. D. Soo, A. Tansel, P. Tiberio, and G. Wiederhold. A consensus glossary of temporal database concepts. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(1):52–64, March 1994.
- [JS95] C. Jensen and R. Snodgrass. Semantics of time-varying attributes and their use for temporal database design. In *OOER'95 Object-Oriented and Entity Relationship Modelling*, volume 1021 of *Lecture Notes in Computer Science*, page 366ff, 1995.
- [JS96] Christian S. Jensen and Richard T. Snodgrass. Semantics of time-varying information. *Information Systems*, 21(4):311–352, June 1996.
- [Kad93] N. Kadmon. Topography, topology and time – the a + 3t concept of geographical space in GIS. In *Proceedings of the 16th International Cartographic Conference*, pages 901–909. International Cartographic Association, 1993.
- [KCLS95] Woo Saeng Kim, Duk Chul Chang, Tae Young Lim, and Young Ho Shin. Temporal object-oriented data model for the modification. In *Proceedings of 4th International Symposium on Database Systems for Advanced Applications (DASFAA)*, Singapore, April 1995.

- [Kim90] Won Kim. Object-oriented databases: Definition and research directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3):327–341, September 1990.
- [Kli93] N. Kline. An update of the temporal database bibliography. *ACM SIGMOND Record*, 22(4):66–80, December 1993.
- [Kol93] Curtis P. Kolovson. Indexing techniques for historical databases. In *Temporal Databases: Theory, Design, and Implementation*, pages 418–432. Benjamin/Cummings, 1993.
- [Kos94] Bogdan R. Kosanovič. Temporal fuzzy sets. Technical report, University of Pittsburg, September 1994.
- [KRS90] W. Käfer, N. Ritter, and H. Schöning. Support for temporal data by complex objects. In *Proceedings of the 16th Conference on Very Large Databases (VLDB'90)*, Brisbane, Australia, 1990.
- [KTF97] Anil Kumar, Vassilis J. Tsotras, and Christos Faloutsos. Designing access methods for bitemporal databases. Technical report, University of Maryland, College Park, March 1997.
- [KV92] Menno-Jan Kraak and Edward Vebree. Tetrahedrons and animated maps in 2D and 3D space. In *Proceedings, 5th international symposium on spatial data handling*, pages 63–71, Charlston, USA, 1992.
- [Lan89] Gail Langran. A review of temporal database research and its use in GIS application. *Int. Journal of Geographic Information Systems*, 3(3):215–232, 1989.
- [Lan90] Gail Langran. Tracing temporal information in an automated nautical charting system. *Cartography and Geographic Information Systems*, 17(4):291–299, 1990.
- [Lan91] Gail Langran. Dilemmas of implementing a temporal GIS. In *Mapping the Nations. Proceedings, Volume 2*, pages 547–555. International Cartographic Association, 1991. 15th Conference and 9th General Assembly. Bournemouth, UK.

- [Lan92] Gail Langran. *Time in Geographic Information Systems*. Technical Issues in Geographic Information Systems. Taylor & Francis, 1992.
- [Lan93] Gail Langran. Issues of implementing a spatiotemporal system. *Int. Journal of Geographical Information Systems*, 7(4):305–314, 1993.
- [Lau97] Sven-Eric Lautemann. Schema versions in object-oriented database systems. In Rodney Topor and Katsumi Tanaka, editors, *Database Systems for Advanced Applications*, pages 323–332, Melbourne, Australia, April 1997. World Scientific.
- [LC88] Gail Langran and Nicholas R. Chrisman. A framework for temporal geographic information systems. *Cartographica*, 25(3):1–14, 1988.
- [LKH94] J. Lin, D. C. Kung, and P. Hsia. Toward an object-oriented modeling approach with representation of temporal knowledge. In *Proceedings of the 18th Annual IEEE International Computer Software and Applications Conference (COMPSAC'94)*, pages 58–63, Taipei, Taiwan, 1994.
- [LMST91] P. Loucopoulos, P. MœBrien, F. Schumacker, and C. Theodoulidis. Integrating database technology, rule based systems and temporal reasoning for effective information systems: The TEMPORA paradigm. *Information Systems Journal*, 1(1), 1991.
- [LRSB96] Ling Lin, Tore Risch, Martin Sköld, and Dushan Badal. Indexing values of time sequences. In Ken Barker and M. T. Ozsu, editors, *Proceedings of the 1996 ACM CIKM International Conference on Information and Knowledge Management*, pages 223–232, Rockville, USA, November 1996. ACM Press.
- [LSW97] Jean-Bernard Lagorce, Arunas Stockus, and Emmanuel Waller. Object-oriented database evolution. In Foto N. Afrati and Phokion Kolaitis, editors, *Database Theory—ICDT'97, 6th International Conference*, volume 1186 of *Lecture Notes in Computer Science*, pages 379–393, Delphi, Greece, January 1997.
- [LT93] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems*. Academic Press, 1993.

- [Mac94] A. MacEachren. Time as a cartographic variable. In *Visualization in Geographic Information Systems*, chapter 13, pages 115–130. John Wiley & sons, 1994.
- [MAGM98] Darka Mioc, François Anton, Christopher M. Gold, and Bernard Moulin. Spatio-temporal change representation and map updates in a dynamic voronoi data structure. In *Proceedings, 8th International Symposium on Spatial Data Handling*, pages 441–452, Vancouver, Canada, July 1998.
- [Mar93] James Martin. *Principles of Object-Oriented analysis and design*. Prentice Hall, Englewood Cliffs, N.J., 1993.
- [MB91] William A. Mackaness and Barbara P. Battenfield. incorporating time into geographic process: A framework for analysing process in GIS. In *Mapping the Nations. Proceedings, Volume 2*. International Cartographic Association, 1991. 15th Conference and 9th General Assembly. Bournemouth, UK.
- [McA76] Robert P. McArthur. *Tense Logic*. D. Riedel Publishing Company, 1976.
- [MD91] Alan M. MacEachren and David DiBiase. Animated maps of aggregate data: Conceptual and practical problems. *Cartography and geographic Information Systems*, 18(4):221–229, October 1991.
- [MH69] J. McCarthy and P. J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In Bernard Meltzer and Donald Michie, editors, *Machine Intelligence*, volume 4, pages 463–502. Edinburgh University Press, 1969.
- [Mis93] Gunnar Misund. Multimodels and metamap — towards an augmented map concept. Master’s thesis, University of Oslo, 1993.
- [MK97] Alan M. MacEachren and Menno-Jan Kraak. Exploratory cartographic visualization: Advancing the agenda. *Computers & Geosciences*, 23(4):365–369, 1997. Guest Editorial.
- [MLW95] J-C. Müller, J-P. Lagrange, and R. Weibel. *GIS and Generalization: Methodology and Practice*. Taylor & Francis, 1995.

- [MM92] Alan M. MacEachren and Mark Monmonier. Introduction: Special issue on geographic visualization. *Cartography and Geographical Information Systems*, 19(4):197–200, 1992.
- [MMNS94] L. Mathiassen, A. Munk-Madsen, P.A. Nielsen, and J. Stage. Modelling events in object-oriented analysis. In *OOIS'94. International Conference on Object Oriented Information Systems*, pages 88–104, London, December 1994. Springer.
- [MMS93] Peter Milne, Scot Milton, and John L. Smith. Geographical object-oriented databases—a case study. *Int. Journal of Geographical Information Systems*, 7(1), 1993.
- [MNP⁺91] P. M^oBrien, M. Niézette, D. Panatazis, A. H. Seltveit, U. Sundin, B. Theodoulidis, G. Tziallas, and R. Wohed. A rule language to capture and model business policy specifications. In *Advanced Information Systems Engineering, CAiSE'91*, Trondheim, Norway, May 1991.
- [Mon90] Mark Monmonier. Strategies for the visualization of geographic time-series data. *Cartographica*, 27(1), 1990.
- [Mon96] M. Monmonier. Temporal generalization for dynamic maps. *Cartography and Geographic Information Systems*, 23(2):96–98, 1996.
- [Mou78] Alexander P. D. Mourelatos. Events, processes, and states. *Linguistics and Philosophy*, 2:415–434, 1978.
- [MP93] Angelo Montari and Barbara Pernici. Temporal reasoning. In *Temporal Databases. Theory, design, and implementation*, pages 534–562. The Benjamin/Cummings publishing company inc., 1993.
- [MS92] Robert B. McMaster and K. Stuart Shea. *Generalization in Digital Cartography*. Association of American Geographers, 1992.
- [MS93] Simon Monk and Ian Sommerville. Schema evolution in OODBs using class versioning. *SIGMOD Record*, 22(3):16–22, September 1993.

- [MSW92] Peter M^cBrien, Anne Helga Seltveit, and Benkt Wangler. An entity-relationship model extended to describe historical information. In *Proceedings: CISMODO '92*, 1992. Bangalore, India.
- [MWFF92] R. Medina-Mora, Terry Winograd, Rodrigo Flores, and Fernando Flores. The action workflow approach to workflow management technology. In *Proceedings of CSCW'92*, pages 281–288, November 1992.
- [Nat94] Stefano Nativi. A conceptual modelling for the GIS developing. In *Proceedings of EGIS/MARI '94*, 1994.
- [NDE96] M. A. Nascimento, M. H. Dunham, and R. Elmasri. M-IVTT: An index for bitemporal databases. In *Database and Expert Systems Applications (DEXA '96)*, volume 1134 of *Lecture Notes in Computer Science*, page 779ff, Zürich, Switzerland, 1996.
- [OBM92] Marc A O'Conaill, Sarah B. M. Bell, and David C. Mason. Developing a prototype 4D GIS on a transputer array. *ITC Journal*, (1):47–54, 1992.
- [OK98] Yuraka Ohsawa and Kyugwol Kim. A spatiotemporal data management method using inverse differential script. In *Advances in Database Technologies, ER'98 Workshop on new Database Technologies for Collaborative Work Support and Spatio-Temporal Data Management (NewDB'98)*, volume 1552 of *Lecture Notes in Computer Science*, page 1552, Singapore, November 1998.
- [PD95] Donna J. Peuquet and Niu Duan. An event-based spatiotemporal data model (ESTDM) for temporal analysis of geographic data. *Int. Journal of Geographic Information Systems*, 9(1):7–24, 1995.
- [Pet62] C. A. Petri. Kommunikation mit automaten. *Schriften des Rheinisch-Vestfallischen Institut für Instrumentelle Mathematik an der Universität Bonn*, 2, 1962.
- [Peu88] D. J. Peuquet. Representations of geographic space: Towards a conceptual synthesis. *Annals of the Associations of American Geographers*, (78):375–394, 1988.

- [Peu94] Donna J. Peuquet. It's about time: A conceptual framework for the representation of temporal dynamics in geographic information systems. *Annals of the Association of American Geographers*, 84(3):441–461, 1994.
- [PH92] Simon Pigot and Bill Hazelton. The fundamentals of a topological model for a four-dimensional GIS. In *Proceedings, 5th international symposium on spatial data handling*, pages 580–591, Charleston, USA, 1992.
- [PQ96] D. Peuquet and L. Qian. An integrated database design for temporal GIS. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 2.1–2.11, 1996.
- [Pri89] Stephen Price. Modelling the temporal element in land information systems. *Int. Journal of Geographic Information Systems*, 3(3):233–243, 1989.
- [PSZ+98] C. Parent, S. Spaccapietra, E. Zimanyi, P. Donnini, C. Plazanet, and C. Vangenot. Modeling spatial data in the MADS conceptual model. In *Proceedings, 8th International Symposium on Spatial Data Handling*, pages 138–150, Vancouver, Canada, July 1998.
- [PW94] Donna Peuquet and Elizabeth Wentz. An approach for time-based analysis of spatiotemporal data. In *Advances in GIS research*, pages 489–504, 1994. Sixth international Symposium on Spatial Data Handling.
- [QP98] Liujian Qian and Donna Peuquet. Design of a visual query language for GIS. In *International Symposium on Spatial Data Handling '98*, Vancouver, Canada, July 11-15 1998.
- [RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, f. Eddy, and W. Lorensen. *Object-Oriented Modelling and Design*. Prentice-Hall International, Inc., 1991.
- [Ren96] Agnar Renolen. History graphs: Conceptual modelling of spatiotemporal data. In *GIS Frontiers in Business and Science*, volume 2. International Cartographic Association, 1996. Brno, Czech Republic.

- [Ren97] Agnar Renolen. Conceptual modelling and spatiotemporal information systems: How to model the real world. In Hans Hauska, editor, *ScanGIS'97, proceeding of the 6th Scandinavian Research Conference on Geographical Information Systems*, Stockholm, Sweden, June 1997.
- [Ren98] Agnar Renolen. The temporal set-theory, topology, and their applications in spatiotemporal modelling. In *Advances in Database Technologies, ER'98 Workshop on new Database Technologies for Collaborative Work Support and Spatio-Temporal Data Management (NewDB'98)*, volume 1552 of *Lecture Notes in Computer Science*, pages 466–463, Singapore, November 1998.
- [Ren99a] Agnar Renolen. *Concepts and Methods for Modelling Temporal and Spatiotemporal Information*. PhD thesis, Norwegian University of Science and Technology (NTNU), 1999.
- [Ren99b] Agnar Renolen. On generalization and data reduction in spatiotemporal data sets. In *Proceedings of the 7th Scandinavian Research Conference on Geographical Information Systems ScanGIS99*, pages 81–96, Aalborg, Denmark, June 1999. Aalborg Universitetsforlag.
- [Rened] Agnar Renolen. Modelling the real world: Conceptual modelling in spatiotemporal information system design. *Transactions in GIS*, to be published.
- [RMD94] Bhaskar Ramachandran, Fraser MacLoad, and Steve Dowers. Modelling temporal changes in a GIS using an object-oriented approach. In *Advances in GIS research*, pages 518–537, 1994. Sixth international Symposium on Spatial Data Handling.
- [RS91] Joel Richardson and Peter Schwarz. Aspects: Extending objects to support multiple independent roles. *ACM SIGMOD Record*, 20(2):298–307, June 1991.
- [RS93a] Ellen Rose and Arie Segev. TOO: A temporal object-oriented algebra. In *ECOO'93 — Object-Oriented Programming*, volume 707 of *Lecture Notes in Computer Science*, pages 297–325, Kaiserslautern, Germany, 1993.

- [RS93b] Ellen Rose and Arie Segev. TOOSQL: A temporal object-oriented query language. In *Entity-Relationship Approach — ER'93*, volume 823 of *Lecture Notes in Computer Science*, pages 122–136, 1993.
- [RU71] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic*. Springer-Verlag, 1971.
- [SA86] Richard Snodgrass and Ilsoo Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, 1986.
- [Saa91] Alan Saafeld. An application of algebraic topology to an overlay problem of analytical cartography. *Cartography and Geographic Information Systems*, 18(1), 1991.
- [SBJS96] Richard T. Snodgrass, Michael H. Böhlen, Christian S. Jensen, and Andreas Steiner. Adding transaction time to SQL/Temporal. Technical report, International Organization for Standardization, March 1996. Change Proposal.
- [Sea69] J.R. Searle. *Speech acts*. Cambridge University Press, Cambridge, 1969.
- [Sea79] J.R. Searle. *Expression and Meaning*. Cambridge University Press, Cambridge, 1979.
- [Sha90] Murray Shanahan. Representing continuous change in the event calculus. In *9th European Conference on Artificial Intelligence, ECAI90*, pages 598–603, 1990.
- [SK93] A. Sølvsberg and D.C. Kung. *Information System Engineering, An Introduction*. Springer-Verlag, 1993.
- [SK96] Arne Sølvsberg and John Krogstie. *Information Systems Engineering – Advanced Conceptual Modeling*. Draft, 1996.
- [Skj97a] Bjørn Skjellaug. Temporal data: Time and object databases. Technical report, Univeristy of Oslo, Department of Informatics, 1997. Research Report 245.

- [Skj97b] Bjørn Skjellaug. Temporal data: Time and relational databases. Technical report, Univeristy of Oslo, Department of Informatics, 1997. Research Report 246.
- [Smi92] Terence R. Smith. Towards a logic-based language for modeling and database support in spatio-temporal domains. In *5th International Symposium on Spatial Data Handling*, pages 592–601. IGU Commission on GIS, 1992.
- [SN97a] A. Steiner and M. C. Norrie. Implementing temporal databases in object-orientes systems. In Rodney Topor and Katsumi Tanaka, editors, *Database Systems for Advanced Applications*, pages 381–390, Melbourne, Australia, April 1997. World Scientific.
- [SN97b] Andreas Steiner and Moira C. Norrie. A temporal extension to a generic object data model. Technical report, TimeCenter, May 1997. TR-15.
- [Sno87] Richard T. Snodgrass. The temporal query language TQuel. *ACM Transactions on Database Systems*, 12(2), July 1987.
- [Sno90] Richard T. Snodgrass. Temporal databases: status and research directions. *ACM SIGMOD Record*, 19(4):83–89, December 1990.
- [Sno92] Richard T. Snodgrass. Temporal databases. In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space, Int. Conference GIS — From Space to Territory*, volume 639 of *Lecture Notes in Computer Science*, Pisa, Italy, 1992.
- [SPZ98a] Stefano Spaccapietra, Christine Parent, and Esteban Zimanyi. Modelling time from a conceptual perspective. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM)*, pages 432–440, 1998.
- [SPZ98b] Stefano Spaccapietra, Christine Parent, and Esteban Zimanyi. Spatio-temporal information systems: A conceptual perspective. Tutorial held at ER’98 conference, 1998.

- [SS93] Arie Segev and Arie Shoshani. A temporal data model based on time sequences. In *Temporal Databases. Theory, Design, and Implementation*, pages 248–271. The Benjamin/Cummings publishing company inc., 1993.
- [SSP95] Paul Schleifer, Yuan Sun, and Dilip Patel. Modelling temporal semantics on an object-oriented database. In *OOIS'95. International Conference on Object Oriented Information Systems*, pages 337–350, Dublin, December 1995. Springer.
- [SSP96] Paul Schleifer, Yuan Sun, and Dilip Patel. The implementation of a chronical collection class in smalltalk/DB. In *Proceedings of the ACM Symposium on Applied Computing*, Philadelphia, USA, February 1996.
- [T⁺93] Abdullah Uz Tansel et al. *Temporal Databases. Theory, design, and implementation*. The Benjamin/Cummings publishing company inc., 1993.
- [Tan87] A. U. Tansel. A statistical interface for historical relational databases. In *Proceedings, Third International Conference on Data Engineering*, 1987.
- [Tii95] Tiina Kilpeläinen and Tapani Sarjakoski. Incremental generalization for multiple representations of geographical objects. In *GIS and Generalisation, Methodology and Practice*. Taylor & Francis, 1995.
- [TL91a] C. I. Theoudoulidis and P. Loucopoulos. A conceptual modelling formalism for temporal database applications. *Information Systems*, 16(4), 1991.
- [TL91b] C. I. Theoudoulidis and P. Loucopoulos. The time dimension in conceptual modelling. *Information Systems*, 16(3), 1991.
- [TLW91] C. Theodoulidis, P. Loucopoulos, and B. Wangler. The entity relationship time model and the conceptual rule language. In *10th Conference on the Entity Relationship Approach*, San Mateo, CA, October 1991.
- [Tom90] C. Dana Tomlin. *Geographic Information Systems and Cartographic Modeling*. Prentice Hall, 1990.

- [Tve92] Håvard Tveite. Sub-structure abstractions in geographical data modelling. In *Proceedings: Neste Generasjon GIS*. The University of Trondheim, the Norwegian Institute of Technology, 1992.
- [TWL92] C. I. Theoudoulidis, B. Wangler, and P. Loucopoulos. The entity-relationship-time model. In *Conceptual Modeling, Database and CASE, An integrated view of information systems development*, chapter 4. John Wiley & Sons inc., 1992.
- [TYF86] Toby J. Teorey, Dongqing Yang, and James P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2), June 1986.
- [Vas96] Irina Vasiliev. Design issues to be considered when mapping time. In C. H. Wood and C. P. Keller, editors, *Cartographic Design: Theoretical and Practical Perspectives*, chapter 11, pages 137–146. John Wiley & Sons Ltd., Chichester, 1996.
- [VBH96] Andreas Voigtmann, Ludger Becker, and Klaus H. Hinrichs. Temporal extensions for an object-oriented geo-data-model. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 11A.25–11A.41, 1996.
- [VR96] Gunnar Vatn and Sverre Halvard Rognås. The time dimension in GIS. Master's thesis, Norwegian University of Science and Technology. National Center for Geographic Information and Analysis, Buffalo, 1996.
- [Vra89] Ric Vrana. Historical data as an explicit component of land information systems. *Int. Journal of Geographic Information Systems*, 3(1):33–49, 1989.
- [WD92] Gene T. J. Wu and Umeshwar Dayal. A uniform model for temporal object-oriented databases. In *Proceedings, 8th International Conference on Data Engineering*, Los Alamitos, 1992.
- [WE96] J. Won and R. Elmasri. Representing retroactive and proactive versions in bi-temporal databases (2TDB). In *Proceedings of the 12th*

- International Conference on Data Engineering*, pages 85–95, Washington - Brussels - Tokyo, February 1996. IEEE Computer Society.
- [WHM90] Michal F. Worboys, Hillary M. Hernshaw, and Davis J. Maguire. Object-oriented data modelling for spatial databases. *Int. Journal Geographical Information Systems*, 4(4), 1990.
- [Wor92] Michael F. Worboys. A model for spatio-temporal information. In *5th International Symposium on Spatial Data Handling*, pages 602–611. IGU Commission on GIS, 1992.
- [Wor93] Michael F. Worboys. Object-oriented approaches to geo-referenced information. *Int. Journal of Geographical Information Systems*, 8(4), 1993.
- [Wor94a] Michael F. Worboys. A unified model for spatial and temporal information. *The Computer Journal*, 37(1):26–34, 1994.
- [Wor94b] Michael F. Worboys. Unifying the spatial and temporal components of geographical information. In *Advances in GIS research*, pages 505–517, 1994. Sixth International Symposium on Spatial Data Handling.
- [Wor95] Michael F. Worboys. *GIS: a Computing perspective*. Taylor & Francis, London, 1995.
- [WW89] Y. Wand and R. Weber. An ontological evaluation of system analysis and design methods. In E.D. Falkenberg and P. Lindgren, editors, *Information Systems Concepts: An in-depth Analysis*. Elsevier Science Publishers, New York, 1989.
- [Yua98] May Yuan. Representing spatiotemporal processes to support knowledge discovery in gis databases. In *Proceedings, 8th International Symposium on Spatial Data Handling*, pages 441–352, Vancouver, Canada, July 1998.
- [Zad65] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

- [Zdo90] Stanley B. Zdonik. Object-oriented type evolution. In François Bancilhon and Peter Buneman, editors, *Advances in Database Programming Languages*, chapter 16, pages 277–288. ACM Press, 1990.