Reza S. Mirzaei, Spring 2005

Master Thesis

Spatio-Temporal Databases for Indoor Positioning Systems

Department of Computer Science and Information Technology
Norwegian University of Science and Technology

Preface

This thesis was proposed to me in March 2004. The idea was based on the concept of *indoor location aware systems*, which intrigued me more and more, the more I learned about it. One area of particular interest was long term storage and management of information received from such systems. This information can for example be used to not only determine present, but also past positions of objects and relate that to their status or role. Storing this type of information requires a special type of database, one which can relate data to both time and space, hence the concept of *spatio-temporal databases*.

The work in this thesis is largely based on previous research conducted in the field of *location modeling* and general purpose spatio-temporal databases. By extending and combining these two aspects, guidelines for storing and managing indoor location aware information are given.

I would like to thank Bjarte S. Karlsen for cooperation and useful suggestions, and my supervisors Dag Svanæs and Hallvard Trætteberg for their guidance and clever ideas. Thanks also to Torbjørn Skramstad for extensive feedback and comments on how to improve my research.

A special thanks goes out to my brother for his continuous and unconditional support, and the ability to always make me laugh.

Reza S. Mirzaei

Trondheim March 7, 2005

Contents

List of Figures

Abbreviations

ACID	Atomicity, Concurrency, Isolation, Durability
ANSI	American National Standards Institute
ADT	Abstract Data Type
DAG	Directed Acyclic Graph
ER	Entity-Relationship
GIS	Geographic Information Systems
GPS	Global Positioning System
GUI	Graphical User Interface
IBM	International Business Machines
ISO	International Standards Organization
LBS	Location Based Service
RAUM	Relation of Application objects for communicating Ubicomp event Messages
SDT	Spatial Data Type
STER	Spatio-Temporal Entity-Relationship
STDT	Spatio-Temporal Data Type
SQL	Structured Qury Language
TDT	Temporal Data Type
UML	Unified Modeling Language

1 Introduction

1.1 Motivation

As computers become more and more advanced, so do the applications and services provided by them. Computers are becoming physically smaller at the same time as their processing power is increasing. This makes them easy to carry around, and the availability of these mobile devices can be used to develop new and powerful applications to cope with the ever growing demands of users. One such demand is the need for location information and location based services, i.e. services that are "aware" of the context in which they operate and perform certain task according to their location. We now have the technology to track the position of a given mobile device, such as a cell phone or a laptop, using a positioning system. In a world where information is an essential part of our everyday life, using this technology to create location based services may become useful in many contexts. One already well known example of this is the *Global Positioning System (GPS)* [77], a satellite-based navigation system used for determining the precise location of a GPS receiver.

On a smaller scale, like indoor positioning, location based services may be used to track and locate objects within a given domain. Services like these can for example be used to effectively manage equipment and personnel. Imagine for instance if we could track the position of office appliances so that we could know the whereabouts of a particular object over a given timeframe. This information could be used to permanently place that object in the position where it was located most often during the timeframe. The result would be a more effective use of that object.

The same service could be used to determine the position of office personnel to find out where they are at what time, assuming that privacy and legal issues associated with such services have been dealt with and clarified in advance. Since an assumption can be made about an individual's location in relation to his interests, this information could for example be used to relocate personnel in a more efficient manner, reducing personnel traffic in the office. As an example of this, consider tracking employees to see who they meet and cooperate with

over a given period. This information could then be used to relocate personnel so that the ones that meet and work together most often have offices near each other.

With location based services for indoor positioning, the domain of the application is most often a building which must be modeled to meet the requirements of the service that is to be provided. However, buildings are not static objects. Over the course of time rooms and sections are added and removed, and names and labels for them change. These changes have to be reflected in the model as they occur. At the same time, we must be able to track an object to its previous locations despite the fact that rooms and sections might have been added, removed or had their name changed. This means in practice that historical information must not be lost due to changes in our symbolic understanding of the domain. The information itself must be stored in a database system that has been modified to support dynamic location information and operations on that information. To model and implement databases that are able to handle spatial as well as temporal aspects which may exist in dynamic environments thus becomes one of the challenges of designing indoor positioning systems.

As a preparation for the coming chapters, we begin by outlining some of the main issues discussed in this thesis.

1.2 Location Modeling

A *location model* describes the area (usually building) where a positioning system is to be implemented. Location modeling can be done in different ways, depending on the application. The model can be *geometric*, *symbolic* or *hybrid* (both symbolic and geometric). A geometric location model is based on a coordinate system and can be used to precisely determine the position of an object. However, the often complex and hierarchical relationship between objects is not present in such a model. A symbolic location model clearly shows the relationships between different objects in its domain, but unlike the geometric model, it is not suited for precise positioning. The hybrid model is a solution that shows the hierarchical relationship between objects and at the same time allows these objects to define their own coordinate systems for precise positioning.

1.3 Spatio-Temporal Databases

A *spatio-temporal database* is in most cases a conventional database system with added support for storage and management of *spatial* and *temporal* data. Spatial data is a term used to describe geometric information about an object in a given environment, i.e. information regarding *space*. Temporal data is data associated with *time* values to indicate its periods of validity. The combination of these two data types is referred to as *spatio-temporal data*, which in practice means *moving geometries*. Spatio-temporal databases also have added support for *operations* on spatio-temporal data, for example calculation of distance or intersection, or definition of temporal validity.

1.4 Other Requirements

An indoor positioning system, not including the hardware (i.e. sensors or mobile devices used for positioning), requires other services than just a location model and a database system. It also needs applications used to manage the data in the database and to create an interface towards the users. We shall describe these requirements in more detail:

- **Access to data**: A database with information is of no use to us unless we can access the information it has stored. We therefore need a query-language for operations on the data. Most database systems have such a language, but it is usually not suited for use on spatial and temporal data. We thus need to add extensions for spatial and temporal data management to the query language.
- **User interface**: Finally, we have to create an interface towards the users that allows them to use the service effectively. The interface must allow the users to query the database about the information they need, in a language that is easy and intuitive for them to understand. A user might for example enter the query "list all employees near the cantina on Thursday afternoon". The application should to some extent be able to interpret such queries. To do this, that is, to map a query like the example to

an executable query, the application must have an understanding of the abstract notions "near" and "afternoon", to mention a few. The interface should allow the users to access the data by letting them define terms that they are familiar with, i.e. "close to", "morning", "noon", and mapping these to values that the application understands.

1.5 Research Questions

The main purpose of this thesis is to ***propose guidelines for modeling and constructing spatio-temporal databases for indoor positioning systems that are flexible enough to deal with changes in their underlying location models***. We are not interested in creating a spatio-temporal database *system*, rather a database for managing spatio-temporal data. It is therefore important that these proposals are general enough so that they can be used independently of specific database systems. But before we can model and construct the database itself, we must consider how to create the underlying location model. Thus, our research and its contribution to the field of indoor location based services will be based on answering the following questions:

1. What are the requirements of location modeling and how do we fulfill these requirements in order to create location models used for symbolic as well as geometric positioning?

2. How can we capture the spatial and temporal aspects of data when modeling spatio-temporal databases, and how do we implements these as data types with corresponding operations in a database system?

3. Based on spatial, temporal and spatio-temporal data, how can we construct spatio-temporal databases that are able to reflect changes in their location models?

In order to fulfill the purpose of this thesis and to answer the research questions, we need comprehensive knowledge of location modeling, spatio-temporal databases and access methods to spatio-temporal data. We try to achieve this knowledge in chapters 4, 5 and 6, presenting in detail the required concepts.

We also want our research to have a realistic practical potential, which is why we must answer these question in relation to current needs and demands posed on indoor location aware systems. Therefore, as a part of our discussion, we also present example applications and scenarios for practical use of both location models and spatio-temporal databases.

1.6 Limitations

The focus of this thesis is location modeling and modeling of spatio-temporal data and its integration into a database system. Since the application domain is so wide, we limit the research to the requirements and examples of indoor positioning systems. These systems operate in dynamic environments where both space and time are essential aspects of their functionality. We therefore feel that the challenges posed by them and the examples used to describe these challenges are adequate enough to capture the fundamentals of our research.

Furthermore, our research will be limited to the design and construction of spatio-temporal databases. The effectiveness of querying mechanisms and the speed and efficiency of index methods are therefore beyond the scope of this thesis.

Another limitation is technology used to determine positions. Our research aims at developing models which are independent of particular positioning technologies, which is why we do not cover specific technological aspects of location aware systems.

Finally, we limit our discussion to purely conceptual aspects of location-awareness and spatio-temporal databases. Ethical and legal issues regarding the use (and potential abuse) of information gathered by and stored in such systems is not covered by our work.

1.7 Structure of Thesis

The structure of this thesis is as follows: We begin the next chapter by presenting related work. We discuss different related areas and present an overview of

relevant literature and research. We also present general concepts of database systems which are relevant in the course of our work.

In chapter 3 we discuss research methodology in the field of informatics (computer science) and use the terminology to present our research approach.

Chapter 4 presents one of the main concepts of this work: *location modeling*. We present existing models and compare their strenghts and weaknesses. We then propose our own model, and use this to create a location model of a building.

The concepts of spatio-temporal databases are discussed in detail in chapter 5. We introduce the basic concepts required to properly model a spatio-temporal database. We also present spatio-temporal data types and how to define these and perform operations on them.

In chapter 6, we show the implementation of a flexible spatio-temporal database, based on the location model and the spatio-temporal data types and extensions described in previous chapters.

Evaluation of, and reflections over our research is done in chapter 7. Here, we evaluate our results according to the three research questions. The purpose of this is to evaluate if our research has been thorough enough to give satisfactory answers to those questions.

Finally, in chapter 8, we summarize the thesis and propose topics for further research.

2 Related Research

The main objective of this chapter is to discuss related research in the field of location modeling, location based services and spatial and temporal database systems. We consider related work and already implemented solutions to identify their requirements and consequences of the decisions made to meet them. The knowledge gained and the lessons learned from previous and existing projects will be used in our own proposal for location modeling and construction of databases for positioning systems. As a part of our discussion, a brief introduction to database system models will also be given.

2.1 General Location Based Services

Location based services (LBS) are becoming possible due to advances in wireless communication and computer science. These services are often described as applications which operate according to geographic information. LBS can be used to offer services to users based on their location, user profile and context. Applications like these are used for many purposes and are constantly increasing in popularity. This is because vendors and operators regard LBS as an integral and inevitable part of their service offering, allowing them to become more competitive and increasing their revenue [46, 50, 82]. Future predictions of LBS are therefore positive. Kamil and Kirk [41] say:

> *These services, which include personal security, navigation, gaming, security and fleet management, have experienced tremendous growth over the last few years. By looking at the factors contributing to this successful growth, LBS can be deployed in Europe with equal success.*
>
> *[41], p. 105*

These positive predictions are also supported by the independent analyst and consulting company *Ovum*, which in 2001 estimated that the LBS market in Western Europe would reach USD 6.6 billion by 2006, with 44% of mobile users

using some kind of LBS [82]. Although this estimate might seem extremely positive, it still remains to see if it will be fulfilled.

An introduction to a few currently available location based services is given in the following:

Geographic Information Systems - GIS

Geographic Information Systems (GIS) are used to store, retrieve, map and analyze spatial (geographic) data. This is usually done by storing spatial data in a coordinate system which references a particular geographic area. In practice, this means that GIS operate with digitized maps and present geographic and thematic information [2, 75]. According to Leonhardt [45], the functionality of GIS can be extended to track the position of mobile objects. Examples of GIS software are *ArcView* [68] and *ArcInfo* [67].

Global Positioning System - GPS

The *Global Positioning System (GPS)*[1] is based on a constellation of 27 *Navstar* satellites developed originally for the *U.S. Department of Defense*. Each satellite transmits a unique digital code sequence which is picked up by a GPS receiver. The receiver uses these code sequences to determine its altitude, latitude and longitude and thus its position within submeter accuracy. Besides 3-dimensional positions, GPS also offers velocity and highly accurate time information to users with GPS receivers. GPS is currently a *dual-use system*, i.e. both military and civil. It is controlled by a joint civilian/military executive board of the U.S. government and monitored by the U.S. Department of Defense [77].

GSM-Positioning

GSM stands for *General System for Mobile Communication* and is a mobile digital cellular radio-communications system. It was introduced in commercial form in 1991 and is currently the most popular standard for mobile phones in the world. A GSM-network is divided into cells containing *base tranceiver stations (BTS)*, usually referred to as *base stations*. The base stations provide the radio interface with the mobile devices. They regularly send out beacon signals which are used by the mobile devices to monitor the quality of available cells. Based on

[1] Also referred to as the *Navstar Global Positioning System*.

this measurement, the mobile device decides when to switch cells. The process of switching cells is known as a *handover* (also called handoff). Because each cell covers a given geographic area, the position of a mobile device can be determined by knowing which cell it is currently in, often by using information from more than one base station. During a handover, a location update is performed by the network, that is, location information about the mobile device that changed cell is updated [56, 76].

2.2 Location-Awareness

The term *location-awareness* is used to describe the capability to detect the exact or relative location of a device (e.g. wireless device like a laptop). Leonhardt [45] defines location-awareness as follows:

> *Generally, location-awareness facilitates an application's awareness of its environment or context.*
>
> *[45], p.16*

Location-awareness is used by components in a system to interact with each other based on location. In order to support location-awareness, a data model is required that can adequately represent the location of mobile and fixed objects. Such a model, referred to as a *location model*, can be designed in two principal ways: as an n-dimensional coordinate system or as a set of symbols with relationships between them [15, 16, 40, 45].

2.2.1 Location Models

At the basis of most positioning systems there is a *location model*. This model is used to define the domain that the location service is meant to cover and thus supply the necessary support for location-awareness. It is therefore important to create location models that are accurate enough in modeling the real world according to the needs of the system.

Leonhardt [45] distinguishes between *geometric* models which define locations using geometric coordinates, and *symbolic* models which define locations and relationships between them using symbols. This classification is also used by Domnitcheva [15]:

> *In a "geometric model", both locations and located objects are represented by sets of coordinate n-tuples, better understood by a human as points, areas and volumes. [...] "Symbolic models" refer to a location by some abstract symbols.*
>
> *[15], p. 3*

The symbolic approach is better suited for describing relationships between locations in the model, such as a building containing rooms. Such relationships are known as *spatial containment relationships*, where a spatial object like a building contains other spatial objects like floors and rooms. Spatial relationships may also be used to describe the connection between two or more spatial objects, i.e. to describe *closeness*. The symbolic approach does however lack the possibility for accurate positioning. Models based on the geometric approach are more suited for this purpose. The geometric approach is also better suited for calculations such as determining the euclidean distance between two objects, but it cannot describe spatial relationships.

As mentioned above, the model must be designed so that it corresponds to the needs of the system. If the system for example requires that spatial relationships like containment or closeness between location are presented in the model, the symbolic approach is preferable. If the system is meant to perform accurate positioning and complex calculations according to locations, the geometric approach is adviced. There are however systems which require the use of both the symbolic and geometric approaches. For this purpose, Leonhardt [45], Jiang and Steenkiste [40], and Dürr and Rothermel [16] propose a combination of the two approaches referred to as a *hybrid* model. This approach allows the model to show locations as symbolic elements with spatial relationships between them, and at the same time allows these elements to define geometric coordinate systems used for accurate positioning.

2.2.2 Hierarchical (Set-Based) vs. Graph-Based

Designing a location model can be done using either a *hierarchical* (also called *set-based*) or a *graph-based* structure. The hierarchical approach organizes objects into a treelike branching structure where each component has only one owner or is contained in one higher level object (a $1 : N$ structure). It is "based on the containment relation" [7], p. 25, and is best suited for describing spatial containment and closeness relationships. Data organized using this structure is best suited for *range-queries*. These queries retrieve data according to some range, for example containment. A specific example of this may be to query the data about who has been in a particular room at a given time, or which rooms are contained in a specific floor of a building [7].

Graph-based solutions (e.g. lattice) are $N : N$ structures where there is no clear hierarchy between the objects. Any object can be symbolically connected to any number of other objects, depending on the relationship between them. According to Becker and Dürr, this approach "supports the definition of the topological relation *connected to* as well as the explicit definition of distances between symbolic coordinates" [7], p. 26. These structures do not describe spatial relationships like containment as well as hierarchical structures, but they are better for so called *nearest-neighbor-queries*, i.e. queries concerning distance between objects. Examples of such queries may be to find the nearest color printer or to find the shortest route between two locations. It is important to emphasize that we are not talking about euclidean distance here, but distance in buildings where we have to consider doors between locations, stairs, corridors and elevators (personal correspondence with Frank Dürr[2]).

2.2.3 Combined Location Model

As seen from the discussion above, the set- and graph-based symbolic location models each have their benefits. In order to utilize these benefits, Becker and

[2]Frank Dürr is a scientific staff member of the "Distributed Systems" section of the "Institute for Parallel and Distributed High-Performance Systems (IPVR)" of the *University of Stuttgart*. His current fields of research are geographic communication and context-aware computing.

Dürr propose a combination of the two, referred to as *combined symbolic location model* [3]. The combined model consists of two parts:

The set-based part represents locations as a set of symbolic coordinates with a set/sub-set relationship such as building containing floors, which again contain rooms etc. This part is best suited for range-queries.

The graph-based part connects locations in the model with edges if such a connection between those locations exists in the real world. An example of this can be stairs between different floors, or a door between two rooms. This part is best suited for nearest-neighbor-queries.

In addition to combining these benefits, Becker and Dürr show that the combined approach can be used to "generate views with different levels of detail" [7], p. 26. By this they mean that a combined symbolic location model can on one level be used to show the rooms on a particular floor along with connections (doors) between them, and on a higher level only show connections between the floors of the same building via their stairways [7].

2.3 Indoor Location Based Services

Research in the field of location based services and positioning systems has enabled us to design and implement functional *indoor* location based services that are precise and effective to use. These are applications which operate within a limited area, i.e. a building. In the following, we introduce three such known implemented services and present the overall design of them.

2.3.1 Active Badge

The *Active Badge System* [34, 65] was first developed at the *Olivetti & Oracle Research Laboratory* during the years 1989-92. The purpose of this system is to locate people and equipment (objects) within a building. This is done using

[3]This combination must not be mistaken for the combination of symbolic and geometric location models (hybrid location model). The combination presented here concerns two different types of *symbolic* location models.

small devices (active badges) worn by people or attached to equipment. These devices periodically transmit infra-red signals (containing a globally unique code) that are detected by sensors placed within the building. The location of the badge, and hence the bearer, can be determined by the information received.

As part of the system architecture, *location* and *name* of an object are both attributes of the object. The value of "location" is chosen from some symbolic or geometric naming scheme and used to relate co-located objects. The value of "name" is the name by which the object is known. The location attribute is dynamic and stored in global databases of location information. Naming information is managed by name servers which use hierarchies of names or directory services to provide global naming. The data model for location information about a particular object is given by a tuple consisting of the *badge address*, *location address* and a *timestamp*. A badge address is the unique identifier for a given badge. A location address describes the latest position of an object within a given domain. It consists of a domain name, network address and sensor address. A timestamp is the time at which the location operation was performed.

Design of the Active Badge system follows a client-server approach. Services required by the system are provided by the following servers:

- **Location Server**: The location server collects information from the sensors and maintains a cache of the latest badge "sightings", that is, the latest location information sent by active badges.
- **Name Server**: The name server maps badge and location addresses into detailed information such as for example the symbolic name of a place or the name of a bearer.
- **Message Server**: The message server coordinates message delivery to and from the badges.
- **Exchange Server**: The exchange server distributes information between organizations and encapsulates the issues of security, access control and information exchange between information domains.

For our purposes, only the location server is of interest. It polls the badges for data, screens the received data for error, timestamps it and stores it. The location server can be thought of as a four-layer system consisting of a *network controller* responsible for polling all sensors on the network, *representation* of valid data received from the network controller, *data processing* to reflect changes in badge locations or recent history of a badge, and a *display interface* that uses the location information from the previous three layers as input for a display function showing textual information about the changing location of badges [65].

Although the location server has a vast array of functionality, the lack of a general location model ties it to the Active Badge system. The system thus lacks abstraction. Furthermore, location in the Active Badge system is described and communicated symbolically. There is no geometric description of the location information, which makes it impossible to perform accurate positioning [45].

2.3.2 MemoClip

The *MemoClip* system [6] is one of three major prototypes built at the *TecO Research Lab* at the *University of Karlsruhe* from 1998 to 2002. The other two prototypes are *MediaCup* and *Smart-Its*. Because the underlying location model is the same in all three projects, we find it adequate to describe and evaluate one of them. The MemoClip is a wearable artifact that reminds its bearer of things he should do depending on his location. The bearer can download onto the MemoClip tasks/information he wants to be reminded of with a description of a location, and then be notified when entering the selected location.

The location model used in the MemoClip project (and the other two projects) is based on the *RAUM*[4] model (location-based **R**elation of **A**pplication objects for communicating **U**bicomp event **M**essages). This model provides a framework for describing and expressing location information. RAUM consists of two parts, the *Location Representation Model (LRM)* and the *Communication Model (CM)*. In our case, only the LRM is of relevance. It defines how loca-

[4]In German the word *raum* stands for *space* as well as *room*.

tion information is represented, stored and communicated between artifacts in a system.

Initially, the location model used in the project was based on a flat hierarchy of places that were identified by a number. Use of this model assumed a fixed knowledge of the environment and thus reduced the flexibility, scalability and portability of the system due to the fact that environments often change (i.e. new rooms are added). This was the motivation behind the development of the RAUM model. The initial location model was first modified to support the representation of an organization. The information was ordered into a tree structure with the organization identifier as the root and sublayer identifiers as the leaves. The model was then modified by adding geometric information in the leaves for positioning an object inside a sublayer. An example of location model based on RAUM is given in figure 1. The location-tree consists of three general layers:

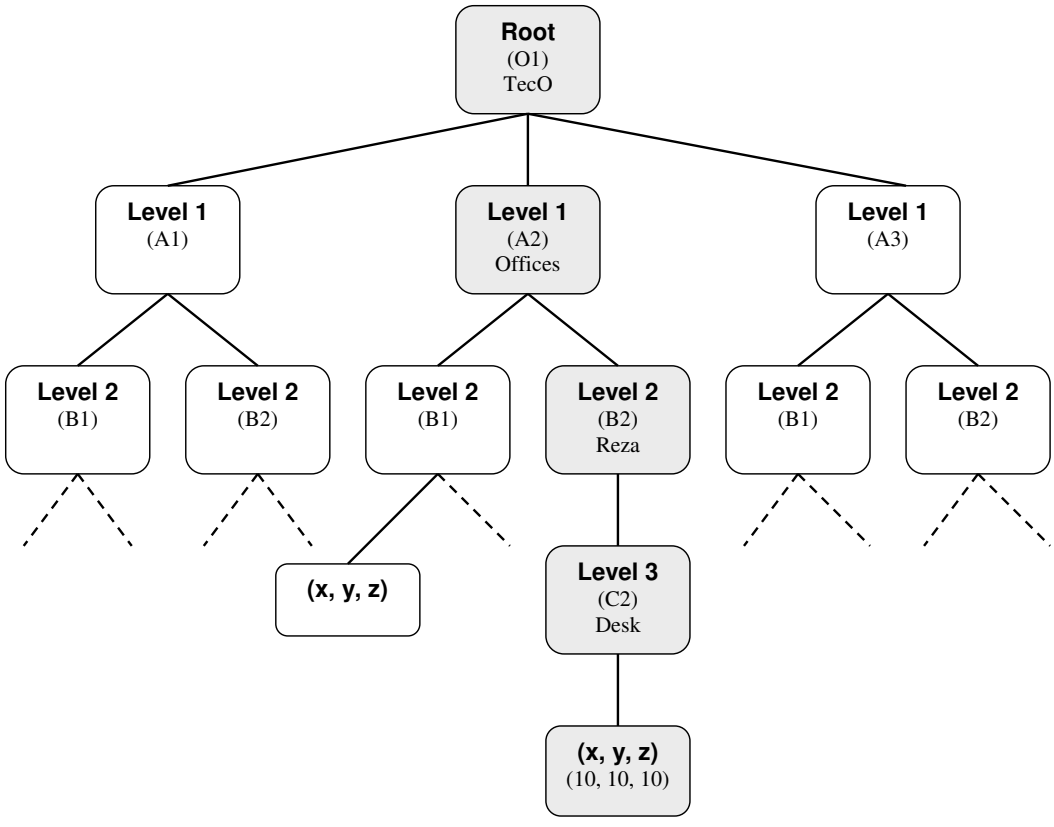

Figure 1: Example of a RAUM location model, as shown in [6]

- Tree-root: The tree-root consists of a textual description (e.g. name) of the organization running the system and a unique identifier (e.g. IP or IPv6 address of the organization). The identifier is used for communication between different systems.

- Semantic sublayers: The semantic sublayers are used to define different physical sublayers of the organization running the system, such as for example rooms, offices, desks or cupboards. Up to three semantic sublayers are optional to provide flexibility.

- Three-dimensional coordinates: These are Cartesian coordinates used to state the position of artifacts accurately within a predefined sphere of interest (e.g. in a room or on a desk).

Despite the RAUM model's flexibility and support for geometric as well as symbolic positioning, it has an apparent weakness: *the ordering of location information into a tree*. The structure of locations represented by a tree does not allow an object in a sublayer to have more than one parent. This causes problems in situations where the symbolic representation of a building forces us to have objects with multiple parents, e.g. a room that belongs to a section as well as a floor, and the section and floor are on the same level in the hierarchy [40, 45, 16].

2.3.3 Ekahau

The *Ekahau Positioning Engine (EPE)* is a WLAN[5] positioning system based on the 802.11 wireless networking technology made for indoor and campus areas where GPS does not perform adequately. Besides a standard 802.11 a, b or g infrastructure, it requires no proprietary hardware and can run on any hardware platform with a minimum of a Pentium II processor, 256 MB of RAM, 200 MB of hard-disk and running Windows XP or 2000. The EPE also contains a Java SDK (Software Development Kit) used for integrating received location information to external applications [72, 73, 74].

[5]Wireless Local Area Network

Ekahau also provides client software which can be installed on devices running Windows XP, 2000, WinCE 3.0, PocketPC 2002 or PocketPC 2003 and used to track these through the EPE. Another options is provided by the *Ekahau T101 WiFi Tag*. This is a battery powered, active radio tracking tag which can be attached to equipment in order to locate its position. The T101 tag can be used to turn a standard 802.11 infrastructure into a *Real Time Location System (RLTS)* with "virtually no disruption to current Wi-Fi operations" [73], p. 2.

The EPE software concept consists of the following components [72, 74]:

- **Ekahau Client**: This is software that runs on a client device and enables it to be tracked by the positioning engine. The device can be a laptop, PDA or a T101 tag.
- **Ekahau Positioning Engine**: This is software that runs on a desktop PC and manages location information from the clients by calculating their x and y coordinates, floor, heading, speed, area name etc.
- **Ekahau Manager**: This application is used to record field data received from the Wi-Fi network for a positioning model. It tracks the client devices on a map and analyzes the positioning accuracy.
- **Ekahau Application Framework**: This is the Java SDK mentioned earlier in this chapter. It provides programmers with an interface used to create application that can utilize the EPE location information.

Unlike the MemoClip system (described in the previous chapter), the EPE uses no explicit location model. Instead it uses a floor map image[6] of the area in which it is meant to operate and requires explicit calibration (using the Ekahau Manager) in order to map location information in the form of x, y, floor and heading to a logical area on the map [72, 73, 74].

The floor map used by the Ekahau Manager can be seen as an implicit location model. The map does not show hierarchical or containment relationships between objects in a building, but is used to create a coordinate system of

[6]The image can be in JPG, PNG or BMP format.

each floor. Although symbolic names are used to identify different geographic locations on the map, this implicit location model must be said to be purely geometric, since no symbolic relationships are present and recieved coordinates are mapped to symbolic names by the EPE.

2.4 Database Systems

Consistent and secure storage of data has always been a vital factor in computer systems. Research in this area has been conducted ever since the first computers came into use. The main question in this research has been, and still is: *How can we store and manage information effectively, with minimum risk to data integrity, minimum occurrences of redundancy, and maximum availability to users?* The result of the research has been a number of general concepts and design principles which are used in most commercial database systems today.

2.4.1 General Concepts

There are a number of database management systems available. Common for these systems, regardless of type, is that they are used to store and manage information in a consistent and secure manner. This is done by facilitating support for persistent data management, secondary storage, concurrency control and crash recovery. These features are implemented by making sure that transactions committed in the system follow the *ACID* model. ACID stands for:

- **A**tomicity: This is the "all or nothing rule". Transactions are said to be "atomic". This means that if one part of a transaction is aborted, then the entire transaction is aborted.
- **C**onsistency: This rule states that only consistent data should be written to the database. If a transaction is found to have violated the consistency rules of the database, it is rolled back. The database is restored to the most recent consistent state.

- **I**solation: Multiple transactions that are running concurrently are not to interfere with each other. Each transaction is seen as an isolated instance.
- **D**urability: The final rule says that committed transactions will not be lost, regardless of software or hardware failure. Committed changes in the database are made durable.

The ACID model is used to keep the data in the database consistent during transaction processing and at the same time allow multiple users to access the same data without interfering with each other, i.e. *concurrency control*. As another security mechanism, in case of system failure, every action that is performed is written to a *log* before the action is made permanent in the database. If a crash should occur while processing transactions, the log is used during restart to restore the databases to a consistent state. Using the ACID model and the log during runtime and crash-recovery ensures that the database is free of anomalies [8].

Another important feature of database systems is that they offer different types of *indexing* mechanisms. The purpose of indexing is to improve system performance by reducing the amount of data that has to be read every time the database is accessed. However, since indexing is beyond the scope of our research, we will not discuss this feature further. Interested readers are referred to [8] for general indexing techniques and [2, 28, 32] for index mechanisms regarding spatial and temporal data.

2.4.2 Data Models

A data model is a collection of concepts that can be used to describe the structure of a database. It separates the principals of design from the concerns of implementation and allows us to treat a database as an abstract machine. The model thus provides the necessary means to achieve data abstraction by hiding details of data storage that are not needed by most database users [8].

The three most common data models today are the *relational*, *object-oriented* and *object-relational*. Although most database systems based on these models

follow the ACID model, write logs and offer the same indexing possibilities, there are fundamental differences between them.

- **Relational Data Model**: This data model is an implementation of relational algebra and set theory from mathematics. It was first described by Edgar Frank "Ted" Codd in 1969 [13]. In database systems based on this model, data and relations between them are organized in tables. Tables are made up of rows and columns, where each row is unique and values in each column are of the same type. The rows in a table are known as records, and the records in one table contain the same fields. Each column is required to have a unique name, and the sequence of columns and rows is insignificant. A declarative programming language known as *Structured Query Language (SQL)* is used to specify the relational algebra necessary to access and manipulate the data. Systems built on the relational model are currently the most popular in commerce and industry. Some of the more popular relational database systems are *Oracle* [87], *DB2* [70], *MySQL* [79] and *PostgreSQL* [90].

- **Object-Oriented Data Model**: Systems based on this model integrate database capabilities with objects-oriented programming capabilities of languages such as *C++*, *SmallTalk* and *Java*. This unifies the application and database development into a seamless data model and language environment. The major benefit of this is that less code has to written in application development, because we now can treat persistent data, as found in databases, and transient data, as found in executing programs, without having to perform any mapping. This also reduces performance overhead during store and retrieve operations on the database. Characteristics required by database systems based on this model were described by Atkinson et al. [3] in 1989. Known object-oriented database systems are O_2 [4] and *ObjectDB* [83], to name a few.

- **Object-Relational Data Model**: The object-relational model[7] extends the relational model with key features of the object-oriented. Also referred

[7]There is no official definition of the object-relational model.

to a *post-relational* or *extended relational*, they allow the definition of user defined data types, nested relations, triggers[8] and stored procedures [8, 59, 37, 38]. As with the relational model, data is stored in tables, but the table entries may now have a richer structure based on *Abstract Data Types (ADTs)* and user defined data types. Access to and maintenance of the database is performed with SQL. Object-relational features are supported among others by Oracle, DB2 and PostgreSQL.

2.4.3 Spatial Databases

Research in the field of *spatial database systems* covers a vast area, from modeling the database itself, to indexing and retrieval mechanisms. Spatial database systems are used to store and manage *geographic*, *geometric* or *spatial* data, i.e. data related to space [28, 29]. The space of interest can be anything from the two-dimensional abstraction of the surface of the earth, to a three-dimensional abstraction of a building. Spatial data can be further divided into geometric information, i.e. position or size, or spatial relationship descriptions, i.e. distance or direction [2].

Over the years, several terms have been used to describe database systems that offer support for storing and managing spatial data. Terms like *pictoral* or *image database systems* arose from the fact that the information to be managed often came from raster images (e.g. remote sensing by satellites). *Geographic* or *geometric database systems* referred to the apparent geographical and geometrical aspects of these systems. The most common term however is *spatial database systems*, a term that has become popular due to the series of conferences "Symposium on Large Spatial Databases (SSD)" held every second year since 1989. According to Güting [28], the reason for this is:

> *[the] view of a database as containing sets of objects in space rather than images or pictures of a space.*
>
> *[28], p. 358*

[8] A "trigger" is an SQL procedure that is activated when a certain event occurs.

Spatial databases must offer support for *spatial data types (SDTs)* and operations on them in their data models, query languages and implementations. SDTs are abstract data types such as points, lines and regions. They are used to define geometric objects such as for example cities on a map, roads, rivers and geographic regions. Operations on these objects are for example calculation of distance, intersection or overlap. With SDTs and their operations we can model the structure of geometric objects in space as well as their relationships. Proposals for definition of SDTs, modeling of spatial databases and spatial query languages are presented in [17, 18, 28, 29] and in the lecture notes from the SSDs [1, 11, 19, 27, 30, 33, 39, 53].

2.4.4 Temporal Databases

Conventional databases contain the latest state of the modeled system. They present only the current information available and reflect a static view of the environment. The validity of the information according to time span is not given in these systems. *Temporal* databases extend conventional ones by providing support for time-varying data. According to Torp et al., such databases can:

> *[...] store multiple version of data by associating time-periods with the tuples, thus indicating when they were logically in the database.*
>
> *[62], p. 1*

Temporal databases incorporate the notion of time by adding a *start time* and an *end time* to each record ($[t_{start}, t_{end}]$), indicating the record's validity. The focus is to extend data in the database to include history. Thus, as the database evolves with time, its previous states are preserved. Research in this field has been driven by the growing demand to handle time related information in databases. The main issues of the research are according to Abraham and Roddick [2]:

> *[...] the representation of time, the selection of appropriate temporal granularity, the level at which temporality should be introduced, support for temporal reasoning, and other database topics.*
>
> *[2], p. 63*

As was the case with spatial databases and SDTs (section 2.4.3), a temporal database must support *temporal data types (TDTs)* and operations on them. TDTs are not just simple time-attributes, but are complex data types which handle different "types" of time, such as continuous, cyclic and terminating time [2]. Continuous time is time as typically described in physics, moving along a linear path with no given end. Cyclic time is used to describe events which occur repetitively within a given timeframe, e.g. commercial breaks every fifteen minutes on cable-TV, while terminating time is used in cases where there is a clear start and end to the timeframe in question, e.g. the existence time of an object. TDTs are used to enhance the data model of a database by adding time and time-related operations. Early proposals for such *temporal data models* are reviewed by Roddick and Patrick in [52]. Further research regarding modeling, implementing and temporal query languages is covered comprehensively by [2, 9, 23, 55, 57] and in the since 2001 bi-annually held "Symposium on Large Spatial and Temporal Databases (SSTD)" [33, 39].

2.4.5 Spatio-Temporal Databases

Development of *spatio-temporal databases* has come under focus during the last decade. Prior to that, spatial and temporal databases were separate areas of research. Although this was the initial situation, most researchers agreed that the two fields had much in common, since they both dealt with *dimensions* or *spaces* of some sort. It therefore became obvious that an integration of the two fields should be studied and that this would result in useful applications. Efforts to combine spatial and temporal database concepts gave rise to the term *spatio-temporal databases*. Sellis [54] defines spatio-temporal databases as follows:

> *Put briefly, a spatio-temporal database is a database that embodies spatial, temporal and spatio-temporal database concepts, and captures simultaneously spatial and temporal aspects of data.*
>
> *[54], p. 309*

The integration of space and time means in effect that we are dealing with *geometries changing over time* [21, 22]. These changes can occur *continuously* or *discretely*. The difference between these two types of change is related to *how often during a time interval* they occur. Continuous change is related to objects like persons, cars or trains moving, or weather phenomena like storms or typhoons changing shape. These are objects which, once they start moving/changing, constantly continue doing so, i.e. they change several times over a given time interval. Discrete change deals with objects changing their position or extent in one single step, like for example a country changing its border, or a building changing its architectural design. It is up to the application/database developers to decide what type of change they are dealing with depending on the frequency of its occurrence.

Several new approaches for modeling *spatio-temporal data* have been proposed. The initial suggestions extended either the spatial model with temporal concepts [66], or the temporal model with support for spatial data [55]. However, as argued by Erwig et al. [21, 22] and Güting et al. [31], such a simple aggregation of space and time is not adequate. The reason is that these approaches are not capable of modeling *continuous* change of spatial objects over time, since they do not consider "the temporal development of a spatial value as a function of time" [31], p. 3.

A solution using *spatio-temporal data types (STDTs)* is described in [21, 22, 31]. This approach captures the temporal aspects of geometric objects by explicitly adding *time* to their spatial properties[9]. It is based on the definition of *abstract data types (ADTs)* and can be integrated into any existing data model or query

[9]This means in effect that spatial and temporal properties of an object are not stored in for example separate columns in relational tables, but in one single column containing one value which represents both properties.

language for databases. The idea is to extend the basic data model to capture time and space. Using these ADTs, we can follow the continues movement and change of spatial objects.

Proposals for STDTs, *spatio-temporal query languages* and spatio-temporal modeling is found in [2, 20, 21, 22, 63]. An overview on research regarding conceptual and logical modeling of spatio-temporal data in given in [54].

Regardless of how we might model the spatio-temporal database or which STDTs or access methods we might use, Erwig et al. [21] conclude that:

> *[. . .] spatio-temporal databases are essentially "databases about moving objects".*
>
> *[21], p. 270*

2.5 Summary

The aim of this section has been to present related research and existing systems in the field of location modeling and spatial and temporal database systems. We have presented general location based services in order to show the extent in which such systems may be applied. A more detailed presentation of research in the field of location modeling and indoor location based services has been given in order to draw relevant knowledge for our purposes, that is, knowledge related to location modeling. Finally, related work in the field of general, spatial, temporal and spatio-temporal databases has been presented, along with the three most common data models used in database systems.

3 Research Methodology

In this chapter we present our research methodology. We begin by highlighting different paradigms, approaches and methods for research in informatics. The purpose of this is to familiarize the reader with the theory and terms of research methodology and to present our research approach. The terminology presented will also be relevant in chapter 7, where we reflect over and evaluate our work.

3.1 Research in Informatics

Computer and information science, also referred to as "informatics", is a multi-disciplinary field. Research conducted in such broad sciences allows for different directions concerning approach and method. Exactly which approach or method that is selected depends on the research questions and how the researcher wishes to answer them.

3.1.1 Paradigms

The term *paradigm* comes from the Greek words *paradeigma* meaning "pattern" or "example" and *paradeiknunai* meaning "demonstrate". According to Kuhn [44] a paradigm is a set of assumptions, concepts, values, and practices that constitutes a way of viewing reality for the community that shares them, especially in an intellectual discipline. The paradigm involves valid research methods, topics and criteria for solution of scientific problems. Two well known research paradigms in informatics, as described by Cornford and Smithson [14], are the *positivist* and the *interpretive*.

The positivist paradigm aims at explaining different phenomena by specifying relationships between variables. The basis for these explanations are laws and theories. The positivist paradigm is based on natural science and assumes a reality independent of human perception. It separates facts from values and explains events by relating them to general laws [51]. It is "an approach that directly reflects the methods of (natural) science and a belief in their generality for all spheres of enquiry" [14], p. 38.

Unlike the positivist paradigm, the interpretive does not assume an independent reality [51]. The purpose of research within this paradigm is to explain and understand different phenomena based on human perception, i.e. based on the way we perceive reality and the meanings which we give to our surroundings. An information system is thus something that "can only be 'interpreted', never fully specified or reduced to theories" [14], p 47.

3.1.2 Perspectives

Research in both paradigms can be conducted with either a *quantitative* or a *qualitative* perspective. The quantitative approach describes the topic and analyzes the data by developing metrics and providing generalizable knowledge. It is based on theories and laws and uses a set of standardized tools.

The qualitative stance relies on other means than numbers and seeks to provide specific knowledge through deep insight and descriptions. Data using this approach is often gathered through observation and documented in the form of text, images or audio/video recordings (e.g. videotaping an interview). It has an exploratory style and the tools used are often adapted to, i.e. made to fit the research topic.

As further perspectives to research, Cornford and Smithson [14] speak in loose terms of "theoretical" (non-empirical) and "empirical" research. Theoretical research is carried out by systematically applying a theory or hypothesis to existing knowledge with the intention of uncovering, changing or integrating new knowledge. The main focus is on ideas and concepts. Empirical research is carried out by observations of the real world where the purpose is to thoroughly explain the observed phenomena. Here, the emphasis lies on observation and data. These two styles of research influence each other mutually:

> *Theory gives motivation to empirical research; [...] Empirical research, in turn, provides evidence to drive processes of theory development.*
>
> *[14], p. 43*

The positivist paradigm relies to a large extent on empirical research [14] p. 37, [51] p. 20. However, it is not thus said that empirical knowledge is only used with a positivist point of view, while theoretical knowledge belongs only to the interpretive paradigm. Research in both paradigms can be conducted using either one of the perspectives. Interpretive research can for example be done by observing some phenomenon over time to describe its behavior, just as positivist research can be conducted using theories to prove/reject laws or theories.

3.1.3 Approaches and Methods

Regardless of a positivist or interpretive basis, there are three major approaches to research: *constructive*, *nomothetic* and *ideographic* [14] pp. 43. The approaches are used to define how the research is to be conducted and what type of results the scientific work is aimed at yielding.

The constructive approach is concerned with answering questions through the construction of models, diagrams and plans. Nomothetic research has the purpose of discovering, supporting or rejecting general laws of behavior through the principles of creating, testing, and applying scientific knowledge. Research based on the ideographic approach seeks to give deep, "thick" descriptions of phenomena by getting involved with the research subject and observing general principals and behavior over time.

Although this classification is helpful in organizing different approaches, further specialization is required in order to define specific methods that can be used to perform scientific work. The different methods systematically define steps that have to be taken along with tools that are to be used when researching a topic.

These Methods can be either quantitative or qualitative. Typical quantitative ones are *surveys* and *laboratory experiments*. Surveys are conducted using questionnaires and structured interviews, while experiments are performed in controlled environments by testing a theory through manipulation of data.

Examples of qualitative methods are *action research* and *case studies*. Action research requires a high degree of subject and researcher involvement. It is

a participatory process concerned with integrating action and reflection, practice and theory. Case studies are systematical empirical studies of individuals, groups or events in their natural environment.

The methods are each somewhat "specialized" for a specific approach. For example, ideographic research is usually conducted using case studies or action research. Nomothetic research is often done with the aid of experiments and surveys. Constructive research is somewhat different because it relies on *design research* and other conceptual or technical development methods. Here, the main objective is to design new or extend existing construction and evaluation techniques such as methods, algorithms or models. This can be done either conceptually or technically.

3.2 Research Approach

Research in this thesis is conducted using a positivistic *constructive* approach. The scientific basis is thus the empirical observation of changes regarding geometrical objects. In order to understand such phenomena, we must first observe them to determine their characteristics regarding movement and change. Only then will we be able to use this type of data in application development. The view is positivistic because these changes occur despite what we might feel about the geometrical objects. Subjective relationships towards these objects are irrelevant: they are there and change regardless of our perception of them.

The empirical observation is done with a *qualitative* perspective. We do not measure or calculate the changes of geometrical objects in any way; we describe them relative to other objects using terms such as "contains", "belongs", "traverses" "intersects" and "crosses". These description are given with respect to time, meaning that relationships between geometrical objects exist/are valid over a given timeframe.

The constructive approach in informatics is aimed at creating frameworks and guidelines for technical design and development. It is also referred to as "design-oriented research". Cornford and Smithson [14] classify two styles of research within the constructive approach: *conceptual development* and *technical develop-*

ment. Conceptual development in informatics seeks to express the requirements of applications without the use of computer metaphors. It is used to describe a system in terms that are independent of computer systems. Technical development in informatics is the design and development of new software or hardware. Our research is aimed at developing frameworks for modeling and constructing databases for spatio-temporal data. We are not concerned with the design and development of the database-system itself, its storage-management, indexing techniques, or the hardware it runs on. We are therefore performing conceptual development.

The method used in our research is "design research". The steps in this method are (1) awareness of the problem, (2) suggestion for solution, (3) development of an artifact (i.e. modeling technique) to test the solution, (4) measurement and evaluation of the results and (5) a conclusion based on (4). Figure 2, taken from [60] depicts the steps in the design cycle.

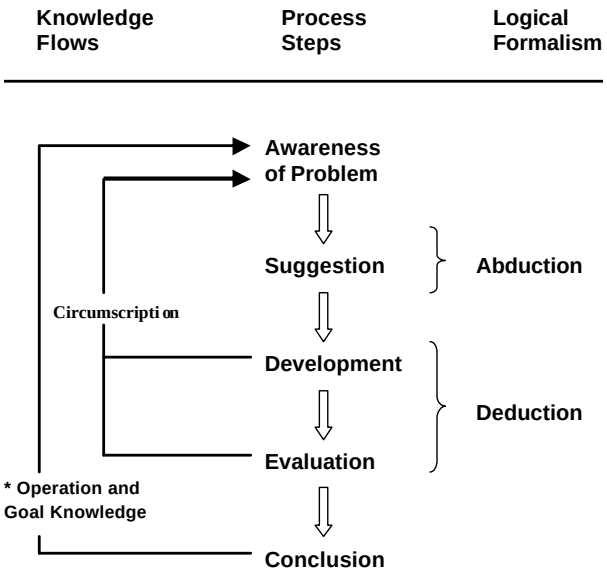

Figure 2: Steps in design research, as shown in [60]

The steps do not have to be followed strictly. A researcher may at any time during the development or evaluation of an artifact decide to take a step back to redefine the problem. The complete set of steps may be performed several times, based on whether or not the conclusion satisfies the problem the researcher

wished to solve.

Our problem concerns location modeling and the construction of spatio-temporal databases, and we suggest solutions based on previous research in those areas. We then use these suggestions to create models of real scenarios and evaluate the results they yield.

Since our problem deals with two different topics, we perform design research on two separate occasions. We begin by solving the issue of location modeling for indoor positioning systems. This is done by following the steps in design research. The second issue, modeling of spatio-temporal databases, is done based on the location model and solved by once again applying design research.

The results of both iterations are conceptual models. The location model is used by indoor location aware applications for management of information. The conceptual model of a spatio-temporal database is mapped to database model which should be general enough so that it can be implemented on any database management system that has the necessary spatial and temporal support.

4 Location Modeling

In this chapter, we give an introduction to location information and its properties and define guidelines which should be followed when building a location model. We then describe the symbolic and geometric approaches to location modeling, and identify their strenghts and weaknesses. Further, we introduce the hybrid location model which is meant to cope with the shortcomings of the geometric and symbolic approaches. Finally, we formally describe our own location model using the *Unified Modeling Language (UML)* [92]. As basis for our model we use the hybrid location modeling approach and explain why we have chosen this particular type.

4.1 What is a Location Model?

The notions of location and location representation are quite familiar to most people. Surely all of us have at some time in our lives seen and possibly even used a map of some sort to find our way to a destination. Maps are location models of a given environment, usually with symbolic names for buildings, streets, cities, countries, mountains, lakes etc. Maps may also have a coordinate system, showing the location of a particular object in geometric terms according to the map's defined coordinate system. The apparent weakness of these maps is that they only allow visualization and reference, but do not show the (often) complex hierarchical relationships between places and objects. More modern methods of location representation take this weakness into consideration and try to build models of environments in a way that not only visualize places and objects, but also show their relationships in a hierarchical manner. Such *location models* are used by location aware applications to relate objects, compute with them (for example compute distance between two objects) and present the results to users.

Just as we humans depend on information about our environment in order to navigate, so do location aware systems depend on information about their environment. The success of these systems depends on how well we are able to provide the system with the necessary information. It is therefore important

that systems which require *spatial data* [28], i.e. geometric information about objects in their environment, are based on location models that support the desired functions. For example, it would be difficult for a human to find the fastest way to his car if the car's position was given as $(38°18'\text{N}, 23°5'\text{W})$. On the other hand, a computer would have a hard time calculating the distance between a car and its owner if all it had to work with were their symbolic names.

As computer systems become an integrated part of our everyday lives, they are expected to perform complex computations and at the same time communicate with their users (humans) in a human understandable way. Taking this into consideration, location modeling becomes a difficult and time consuming task, because it can be done in many ways depending on the application domain and the operations which the application using the model has to perform [42].

However, before we discuss different approaches to location modeling any further, we present the concepts of *location information* and general location modeling guidelines. We also present example scenarios in which location models can be useful.

4.2 Practical Use of Location Models

Location models can be used as a basis for indoor location based applications by supporting different types of functions such as tracking the position and status of objects. Below we present examples of scenarios in which location models have practical use[10]:

Office Management System

One area of applied use is the tracking of equipment and personnel in office complexes. The buildings can be modeled using the constructs of a location model, where the building itself along with its floors, wings and room can be modeled geometrically as 3-dimensional spaces. Transitions (e.g. doors) between these spaces can modeled as 2-dimensional spaces. Associations and relationships

[10]We emphasize that both scenarios presented here have privacy and legal issues associated with them. We do not discuss these issues here, since they are beyond the scope of our work.

between these spaces, along with their symbolic names can be modeled symbolically using a branching structure such as tree or a graph. Equipment inside the buildings can be anything from printers to tables or file cabinets. These can be modeled as objects with positions and the the ability to move to other locations.

Persons can also be considered as moving objects. They can have different roles, occupy offices and attend events. Examples of roles can be "project-manager" or "customer". Events can be meetings, workshops or presentations which take place inside a 3-dimensional space at some given time. The application can thus use these constructs along with their geometrical and symbolic information to determine the position and status of any object and present this in an understandable way to users.

Hospital Monitoring System

Another scenario of practical use is location based applications in hospitals, used to improve medical care. Patients, medical equipment and staff could be tracked at all times, along with their status and roles. Doctors would at all times know where the nurses are and what they are doing, and vice versa. The staff would also know where different equipment is and what condition it is in. Events related to spaces would for example be surgery or planned CAT-scans[11]. Using this information, necessary and available resources could be kept or sent where they are most needed. In extreme situations, this could prove to be lifesaving.

As an example of roles and statuses of persons, a doctor can have the title "cardiologist" and the role of "attending surgeon on call" on a particular day. At a given time during this day, his status can be "available". The nurses can use the location aware application to access this information along with the current position of the doctor and call him if necessary.

[11] Computerized Axial Tomography, also referred to as CT-Scan.

4.3 Location Information

Location modeling of the physical world is an essential part of location aware applications. These applications require highly accurate and flexible information regarding the surroundings in which they are meant to operate. This type of information, referred to by Korkea-Aho and Haitao [42] as *location information*, may be used to query the position of a mobile object, or to calculate the distance between two stationary objects.

The location information describing an object (or a place) is contained in a location model, which also describes the relationships between the objects. The scope of the information in the model is defined by the operations that the application is designed to perform. The quality of the application thus depends on the properties of the underlying location model [42, 10]. These properties are:

- **Accuracy:** How accurate is the location information provided by the model? Does it represent the physical world accurate enough to the extent required by the application?
- **Representation of objects:** How are objects represented in the model? Are the object identifiers proper representations according to names and events associated with the objects?
- **Representation of relationships:** What are the relationships between the objects and how are they represented in the model? Do these representations reflect the relationships in the physical world?
- **Dependency:** What are the dependencies between the objects in the model? Are some objects dependent on other objects in order to exist, and are these dependencies coherent with the dependencies in the physical world?
- **Flexibility:** How flexible is the model regarding changes in the physical world? Is the model flexible enough to reflect such changes?

As we see, there are many factors that have to be considered if we are to build a quality location model. There are also apparent problems concerning accuracy and representation of objects. These properties are somewhat contradictive. Trying to model the physical world accurately with respect to precise positioning would demand an n-dimensional coordinate system, while making the model more understandable to humans would require objects to be represented by their symbolic names and associations. However, a symbolic representation could never be as accurate as coordinates used for precise positioning, while coordinates would be far more difficult for people to understand.

4.4 Modeling Guidelines

The purpose of a location model is to describe the physical world in an abstract and virtual way, so that physical objects (places) and their relationships (and dependencies) can easily be mapped to their equivalent representation in the model. Despite the fact that there are basic standard algorithms for how the technical visualization of physical objects to virtual objects is done, there are no standard demands on how a location model should be created. As already stated, the model has to be constructed according to the needs and specifications of those who want to use it. There are however guidelines as how to proceed when constructing such a model. In the case with indoor positioning, Brumitt and Shafer [10], Funk and Miller [25] and Korkea-Aho and Haitao [42] suggest that the following questions be dealt with:

- What type of operations is the model supposed to support?
  - Activity, needs, purpose, surveillance or administration?
- How is the relationship between objects represented?
  - Absolute (geometric), descriptive or relative (symbolic)?

Once these questions have been dealt with, a decision has to be made as to building an implicit or explicit model. In an implicit model the application contains the model information, while in an explicit model, a service stores

model information and allows an application to query the model data. To achieve better scalability, an explicit model is preferred because such a model is independent of the services which are provided, and can therefore be modified in an easier way. An explicit model would also reduce the workload performed by the clients when they need to access information in the model, because most processing based on the information in the location model would be done by the service provider.

Regardless of the model being implicit or explicit, Dürr and Rothermel [16] propose that it should be a *lattice*[12], where the nodes represent objects (places), and the edges represent the relationships between them.

An alternative to the lattice is a tree, as shown by Jiang and Steenkiste in [40]. The problem with a tree, pointed out by Dürr and Rothermel in [16], is that it allows objects to only have one parent, i.e. exclusive membership. They argue that this approach makes it difficult (if not impossible) to model objects that are subsets of more than one set (i.e. objects that have more than one parent). Because of this, trees will in most cases bee too limited. Consider the following example:

The building in figure 3 has two floors, F_0 and F_1. F_1 contains the room R_{01}, while F_1 contains the rooms R_{11} and R_{12}. So far, it is easy to model this information as a tree. The rooms are all subsets of the floors where they are located, the floors are all subsets of the building, and each object has exactly one parent. But imagine now if the building was also divided into two wings, W_{left} and W_{right}. In this case, wings and floors would overlap, and the rooms would be subsets of floors as well as wings, e.g. room R_{01} would be on floor F_0 as well as in wing W_{left}, and rooms R_{11} and R_{12} would be on floor F_1 as well as wing W_{right}. An object would now have more than one parent, and we would no longer have a tree.

The above example clearly shows why a tree might in some cases not be suited for modeling location information. The more general and powerful lattice over-

[12]A lattice in a *Directed Acyclic Graph (DAG)* where there is at least one node that can reach every other node and there is at least one node that can be reached by every other node.

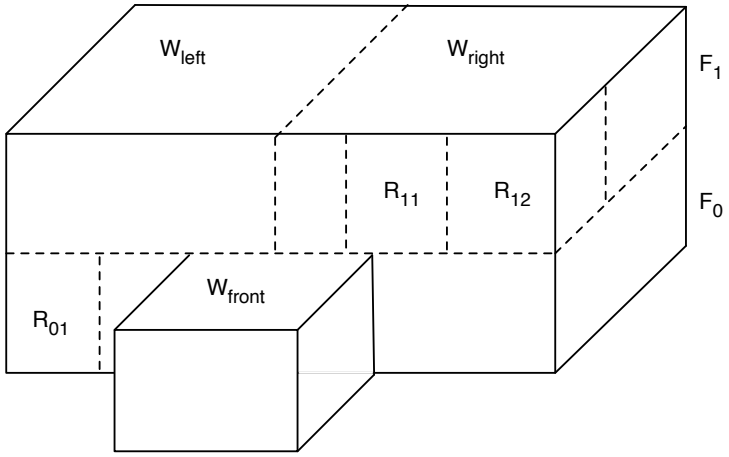

Figure 3: Building divided into sections, floors and rooms

comes the weaknesses of the tree in these situations, and is more suited for our purposes. However, the solution proposed by Dürr and Rothermel [16] also has a weakness. Their model is hierarchical (set-based) and solely based on spatial containment relationship, i.e. a building contains sections which contain rooms. This makes it ideal for *range-queries* like "which objects are within $W_{right}/F_1/R_{12}$", but not suited for *nearest-neighbor-queries* like "find the fastest way between rooms R_{11} and R_{12}". Using the building in figure 3, the problem can be described as follows:

Imagine if there were a door between the rooms R_{11} and R_{12}. In the hierarchical structure proposed in [16], this door would be impossible to model. Nearest-neighbor-queries would thus be answered insufficiently, because all the possibilities of the building would not be modeled. The model would show the rooms connected only through a wing (W_{right}) or a floor (F_1), and the answer to the query would suggest either of these two as a path between the rooms.

To solve this problem, graph-based location models that are well suited for nearest-neighbor-queries can be applied. The edges in the graph may additionally be weighted to express distances between locations. A graph however, as opposed to a hierarchical structure, does not show the spatial containment relationships present in a building. These relationships are important factors concerning indoor positioning systems, and we wish to keep them present in our

location model. We therefore use a combined structure (described in chapter 2.2.3) as a basis for our location model in order to make it suitable for both types of queries.

4.5 Basic Location Models

The basic modeling types for location services are the symbolic and geometric approaches. These are also referred to as *hierarchical* and *coordinate* location models. For the sake of clarity and to avoid confusion, we use the already established terms symbolic (as opposed to hierarchical) and geometric (as opposed to coordinate).

4.5.1 Symbolic Modeling

In a symbolic location model, objects are referred to by names (i.e. abstract symbols), such as "Office 51", "second floor", or "Department of Computer Science". This approach stores information in a way that is meaningful to a person. Brumitt and Shafer [10] use the term *friendly names* which allow location queries performed on the model to be answered with replies that make sense according to the terms and values that people associate their surroundings with.

Since all objects are represented as symbols, they can be divided into *sets* and *subsets*, with basic mathematical set-operations. Consequently, an object is a member of another object if it is physically contained within that object, and objects that are present in overlapping areas between two other objects are members of both sets represented by the overlapping objects [16, 40, 45].

- Advantages of symbolic models:
  - Well suited for implicit representation of spatial relationships (e.g. containment, closeness).
  - Easy to read and understand for humans.

  - Supports better scalability, manageability and adaptation (supports multi-resolution processing).

- Disadvantages of symbolic models:
  - Lack of position accuracy.
  - Not suited for calculating distance between objects.
  - The number of objects in the model depends on the application domain. This could lead to a potentially large number of objects. Examples of this can be large buildings, containing several floors, wings, rooms, corridors, elevators, doors and open spaces.

A symbolic model of the building in figure 3 would look like the one shown in figure 4. The graphical syntax is taken from [16] and is to our knowledge the latest proposed syntax for location modeling.

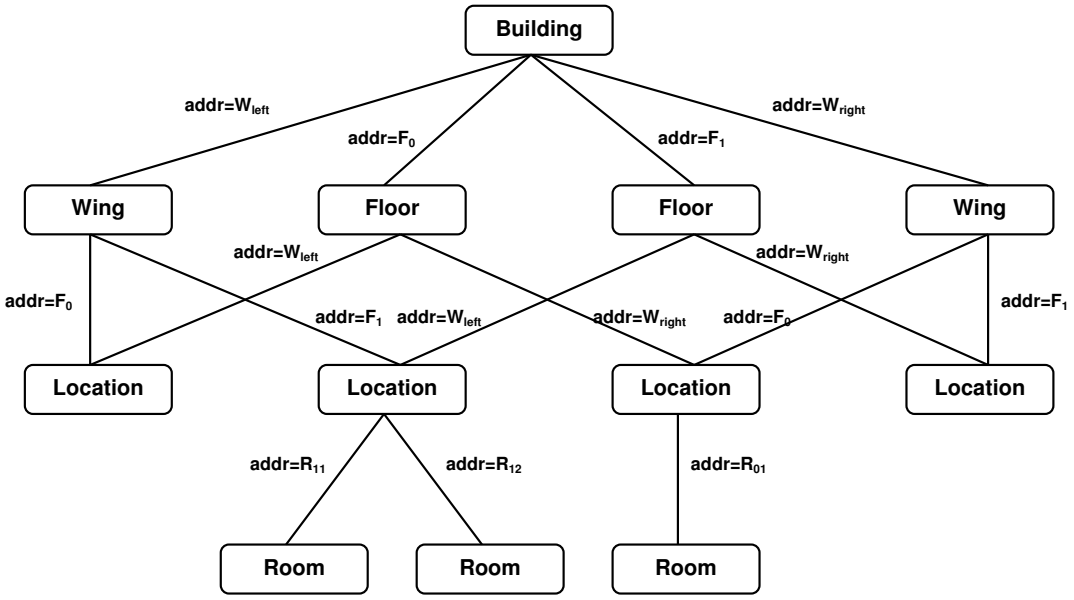

Figure 4: A symbolic model of the building from figure 3, as a lattice

The attribute "addr" in the figure is used to represent the symbolic path from one location to another in both directions of the hierarchy. As the model shows, neither floors nor wings are completely contained in each other. Furthermore,

each room can be seen as a subset of both a floor and a wing. Modeling these aspects using a tree is impossible, as shown in figure 5. This is due to the fact that floors and wings can be seen as set/subsets of each other, and the different rooms would in this case have more than one parent (i.e. room R_{12} is both in floor F_1 and wing W_{right}). These relationships are shown in the model (figure 5) with dashed lines and question marks, indicating that they are unfeasible when constructing a tree.

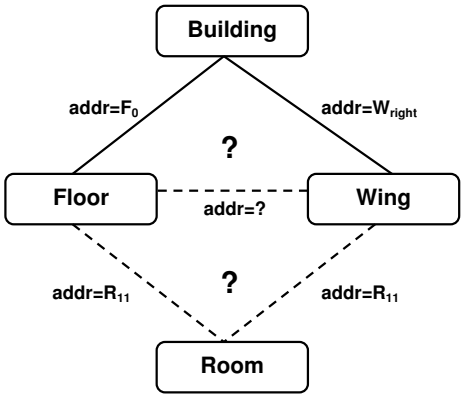

Figure 5: A symbolic model of the building from figure 3, as a tree

4.5.2 Geometric Modeling

In geometric location models, objects are represented as points, lines or regions (or volumes in 3-dimensional applications) within one or more reference coordinate systems. Since everything in a geometric model is described by a set of coordinates, there is no set/subset relationship between the objects, which was the case with a symbolic location model. A well known example of an application based on the geometric model is the GPS coordinate system, in which locations are defined by longitude, latitude and altitude [40, 45].

- Advantages of geometric models:
  - Well suited for for specifying accurate positions.
  - Well suited for calculating euclidean distance between objects.

- Offer a more flexible mean of retrieving location information.

- Disadvantages of geometric models:
  - Hide hierarchical relationships (cannot describe spatial relationships).
  - Geometric data is weakly structured (makes efficient design more difficult).
  - Extra computing is needed to map coordinates into data that is meaningful to both applications and humans.

Two geometric models are shown in figure 6. In model **a)** we see a 2-dimensional geometric model of room R_{01} in the building from figure 3. It consists of a coordinate system used to accurately locate objects within the room. The location of an object within the room is given by a pair of coordinates as (X, Y). The model can be extended to 3-dimensions by adding a third axis, as shown in model **b)**. An object's location is in this case given by an (X, Y, Z) coordinate.

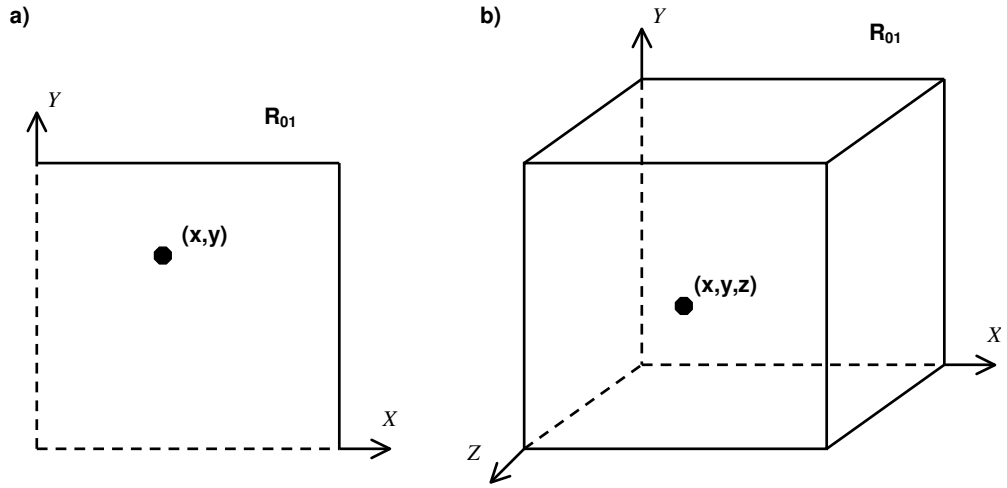

Figure 6: 2- and 3-dimensional geometric models of room R_{01}

A coordinate system may also be used to define subspaces within another space, as shown in figure 7. The figure shows a room divided into the subspaces N, S, E, W and C. This may be necessary when a room or section is symbolically divided (not physically, i.e. by walls) into subsections. An example of this may

be to consider the geographical space in front of a door as a separate area. This way, queries like"outside whose door am I standing" can be answered. We will attempt to answer such queries when validating our model. For now however, we are interested in designing a location model, not using it.

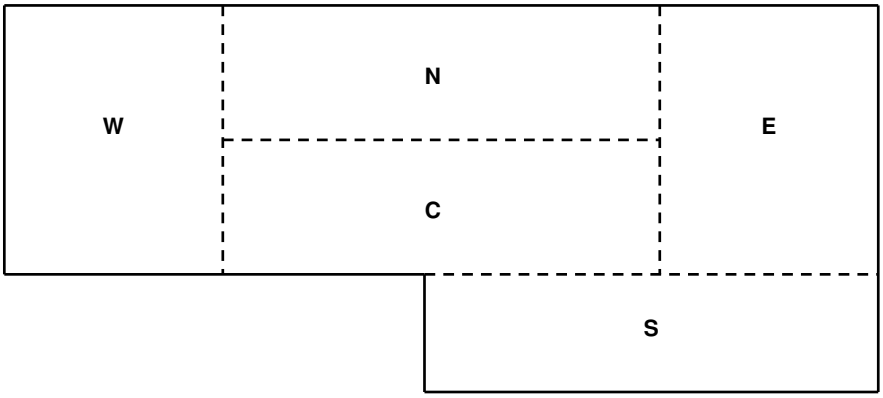

Figure 7: A space geometrically divided into subspaces

4.6 Hybrid Location Model

As we have seen so far, the symbolic and geometric models cover vastly different aspects used by location aware systems. Although either of these models can be used independently form the other, they cannot replace each other. In some cases, especially for closed systems with predefined, static requirements, one of the models alone can provide the necessary service needed. For example, GPS systems use geometric models, while the Active Badge system (section 2.3) is based on a symbolic model. However, for more general services where applications are required to perform precise calculations as well as submit the results to the users in an understandable fashion, it becomes clear that neither the symbolic nor the geometric approach alone is sufficient. To overcome their shortcomings, Leonhardt [45], Domnitcheva [15], and Jiang and Steenkiste [40] suggest a combination of the two, referred to as a *hybrid model*. This approach is also taken by Dürr and Rothermel in [16], where they propose a combination of the symbolic and geometric models.

A hybrid location model combines the benefits of both the geometric and symbolic model. The basis for this approach is the symbolic model, where objects are organized in a hierarchy in which every level is a refinement of the previous. The geometric model is then used to give every object in the hierarchy its own coordinate system, so that points, areas and volumes can be defined within that object.

The symbolic aspect of this approach divides the objects (spaces) into set/subset relationships. For example, the building shown in figure 3 will be divided into the sets F_0, F_1, W_{left} and W_{right} (floors 1 and 2 and wings left and right). As shown in chapter 4.5.1, there exists no set/subset relationship between the floors and wings. They are all on the same level in the object hierarchy, and we therefore also have the sets (W_{left}, F_0), (W_{left}, F_1), (W_{right}, F_0) and (W_{right}, F_1). These sets contain the subsets R_{01}, R_{11} and R_{12} (the rooms), and are in turn self subsets of the building as a whole. Since the symbolic approach produces a lattice, it will be easy to see whether a set/subset relationship exists between two or more objects.

The geometric aspect of the hybrid approach allows each object to define its own coordinate system. Within their coordinate systems, the objects possess geometric attributes such as points, shapes, areas etc. This is necessary if we are to be able to compute geometric relationships such as distance and intersection. Once again, we consider the building in figure 3. Imagine if wing W_{left} was logically (not physically, for example with a wall) divided into two subsections: north and south, W_{north} and W_{south} respectively. We would thus have wing W_{left} as the *superspace* of the subwings W_{north} and W_{south}, and the subwings would be *subspaces* of W_{left}. All wings would have their own coordinate system, and the position of an object in one of the subwings (W_{north} or W_{south}) could be expressed in coordinates of either the superspace's coordinate system (W_{left}) or the subspaces' coordinate system. This is possible because the subspaces' coordinate systems are defined within their superspace's coordinate system, and we can therefore translate coordinates between the spaces [40].

4.7 Domain Model

The purpose of the domain model is to define the characteristics and relationships between objects that a system represents. The domain model is based on our discussion regarding the combination of symbolic and geometric modeling (i.e. the hybrid approach). The objects presented in the domain model are those we deem important for indoor location aware systems, including events occurring in different spaces, static and movable objects with their statuses, and persons moving, occupying spaces and having different roles.

The syntax proposed by Dürr and Rothermel [16] will yield complicated models when the building that is to be modeled is large and consists of several wings, floors, rooms and doors. Because of its rich and powerful syntax, we instead use UML as our modeling language. A domain model describing the basic concepts and terms of the hybrid location model is for the sake of clarity divided into three separate models and presented in figures 8, 9 and 10. All three models use basic constructs from the UML syntax: Classes are modeled as squares, inheritance (generalization/specialization) is modeled with an open-headed arrow pointing from a subclass to its corresponding superclass, composition ("is part of" relationship) is modeled with a filled diamond at the end of a line, and associations are modeled with a straight line with a symbolic name and cardinality.

Figure 8 is the basic model. It shows a 3-dimensional space as a composite of itself (the class "3D Space"). This class contains geometrical information for all 3-dimensional spaces. Direct subclasses of it are "Building", "Floor", "Wing" and "Space". These classes are used to describe the basic concepts of a building. Floors and wings are contained within the building, while spaces are contained within floors and/or wings. Transitions between these spaces are given by "2D Space". The class "Transition", which is a subclass of "2D Space", can be a door or a logically defined border between two 3-dimensional spaces.

The geometrical information of 2- and 3-dimensional spaces is described in figure 9. The information is given by the classes "2D Geometry" and "3D Geometry". As the figure shows, these geometries can be implemented differently depending on the type of spaces that are to be modeled. 3-dimensional spaces are for

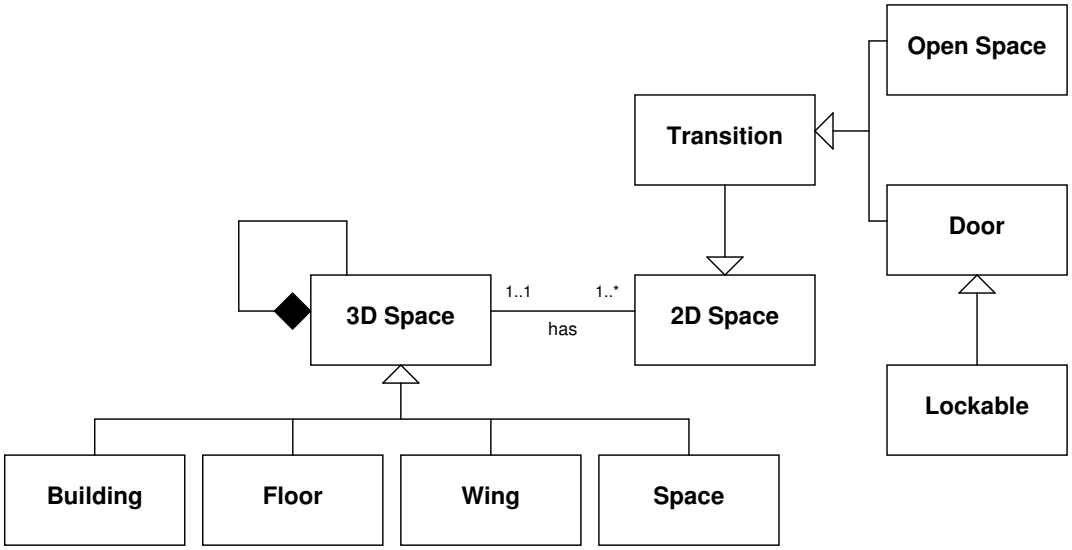

Figure 8: Basic domain model of the hybrid location model, constructed using UML

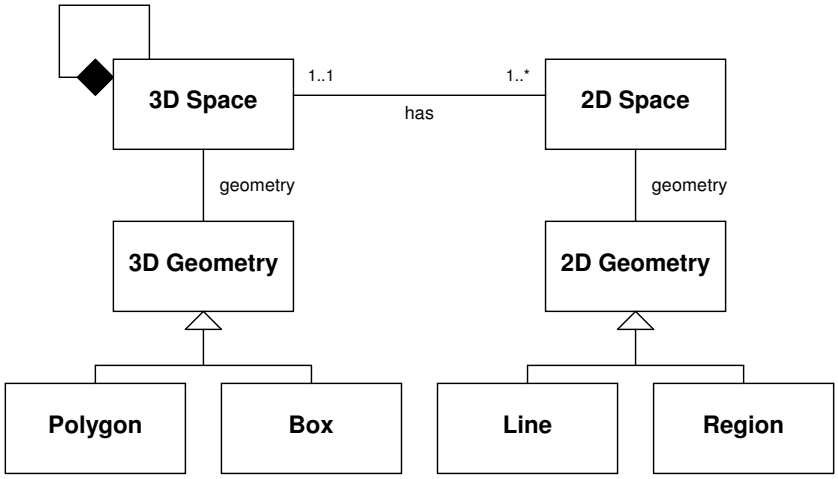

Figure 9: Detailed domain model of 2- and 3-dimensional spaces

example modeled as either polygons or boxes, while 2-dimensional spaces are lines or regions. The model can be extended with other types of geometries like for instance circles, cylinders and spheres. We have chosen the geometries "Polygon", "Box", "Line" and "Region" because we assume that most spaces and transitions within a building can be modeled using these classes.

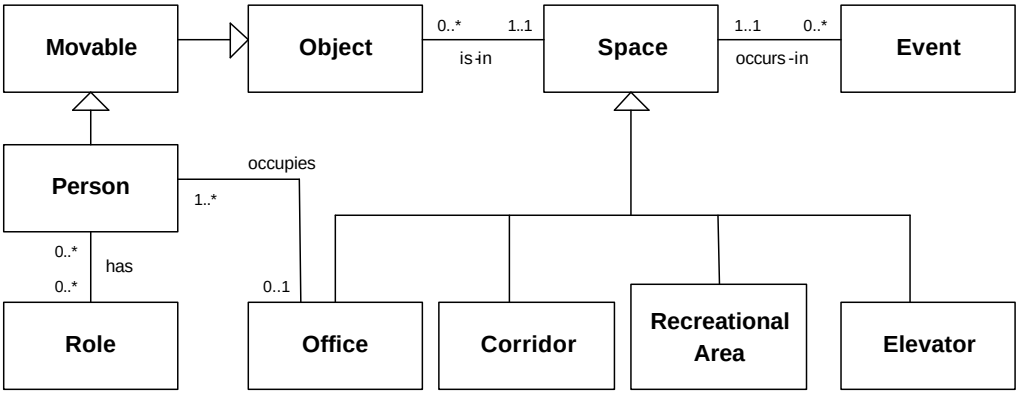

Figure 10: Detailed domain model of space, persons and roles

Figure 10 describes related classes of "Space" in more detail. A space is seen as any type of physically or logically defined area, such as offices, corridors, elevators and geometrically restricted areas ("Recreational Area"). Elevators are distinguishable from other spaces since they move and can therefore be in any floor. Spaces can have events occurring in them, or have movable or static objects placed inside. Offices can be occupied by more that one person (e.g. two persons sharing an office), but one person can occupy only one office. Similarly, an object can be located in one space at a time, while a space can be occupied by several objects at once. Because of this, relationships between objects and spaces are associated with time, indicating the interval in which an objects was located inside a space. Events are also associated with time describing when they are to (or did) occur.

Persons can have different roles at different times. This is shown in the model with the class "Role" and the relationship "has" between "Person" and "Role". The relationship "has" is time-dependent, indicating that the roles of persons may change with time. Roles are specific for persons, but all objects, static, movable or persons, can have a "'status". This property is also time-dependent, since the status of objects change over time.

The concepts of roles and status are here understood as temporary properties which persons possess during given time periods. They must not be mistaken for more permanent properties like title or profession, e.g. "professor".

4.7.1 Validation of Domain Model

Before we can create location models based on the constructs of our proposed domain model, we must validate it according to the properties defined in chapter 4.3. These were: *accuracy, dependency, flexibility and representation of objects and relationships*. However, creating location models of buildings using standard UML might yield complicated results. This is due to the fact that buildings often have many floors, wings, rooms, open spaces and doors. Because of this, we propose a few changes to the syntax in order to make the models more comprehensible.

The first proposal uses plain text instead of association classes when modeling the associative relationship between two objects. Again, because of the many doors that may exist in a building, modeling each one as a class could lead to an unnecessarily complicated model. Since doors have no other properties besides being open or locked (and locked door being a specialization of a door), it is easy to model them using plain text. This will reduce the number of classes in the model, making it less complicated. Figure 11 shows an example of this.

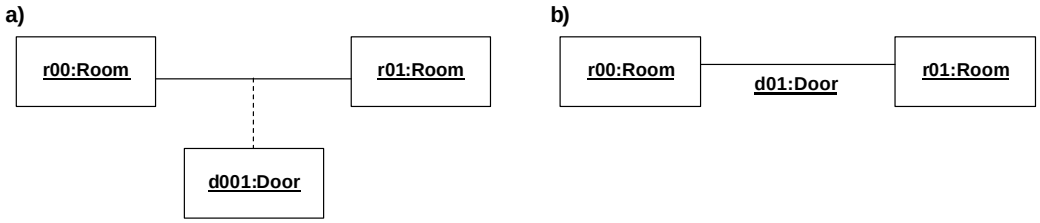

Figure 11: Association between two rooms, shown as an association class (a) and plain text (b)

The second proposal uses the package construct of UML instead of aggregation to describe that several objects are aggregated subsets of another. An example of this is shown in figure 12, where rooms belonging to a floor are modeled with aggregation (a) and as a package (b). The package syntax is "cleaner" since we do not have any overlapping lines. In cases where there are several floors, wings and rooms, the package syntax will reduce the number of overlapping and crossing lines, making the model easier to read and understand. It is important

to emphasize that the package syntax used here does *not* imply a conceptual relationship between the objects within it. We simply use the syntax without applying the semantic.

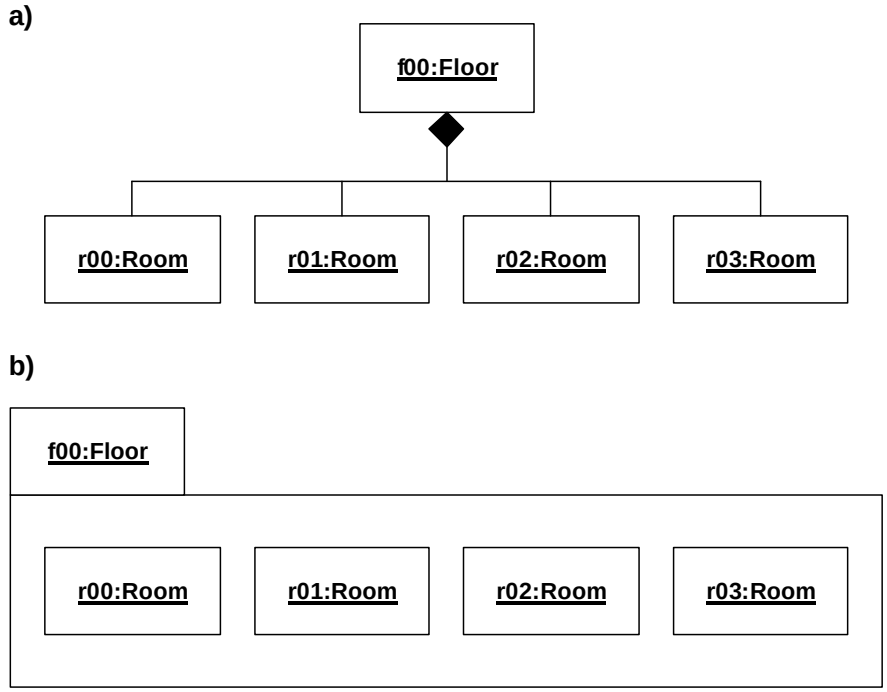

Figure 12: Rooms in a floor, shown as aggregation (a) and package (b)

As a scenario for a location model we use the building from figure 3. A detailed cross section of both floors is given in figure 13, where floors, wings, rooms etc. are numbered. As a numbering scheme, we use the first letter(s) of each type of object as a prefix together with a unique identifier, for example "o01" for "office 01", "e1" for "elevator 1" and "c11" for the corridor in the first floor.

We have omitted the numbering of doors and transitions between geometric areas in the model to make it easier to read. However, theses are numbered using the same scheme described above. We prefix the number of an office/corridor with "d" to show a door that belongs to it. If a room has more than one door, we use a letter from "a" to "z" as a postfix to the number in order to distinguish between them. For example, the door of office "o02" is numbered "do02" and the doors to stairway "sw01" are numbered "dsw01a" and "dsw01b". Transitions between geometric areas are numbered by prefixing the number of the area with

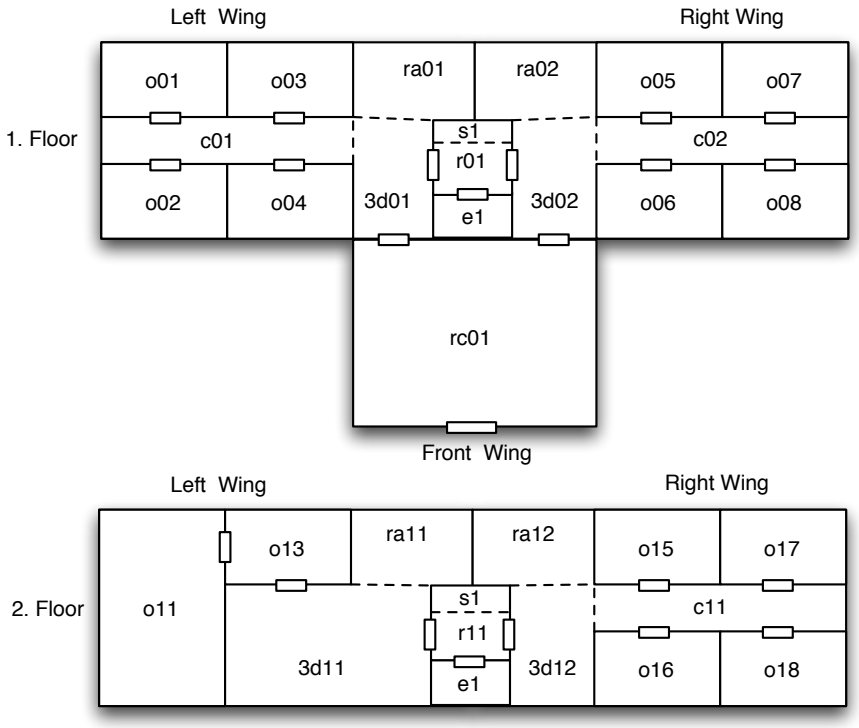

Figure 13: Plan showing both floors of building used for validation of domain model

"os" ("Open Space"). For instance, the transition between the recreational area "ra01" and the 3-dimensional space "3d01" is numbered "osra01".

Figure 14 shows how geometric areas can be defined outside different doors. These areas are numbered by adding the prefix "3d" to the object which they "belong" to, i.e. the object which is related to them. This concept can be used to answer queries such as "outside whos door am I standing?".

The location model for the example building is given in figure 15. The model assumes that floors are contained within wings, not vice versa. Because of this, floors are modeled as aggregated subsets of wings. The model also assumes that stairs between two floors are contained within the lowest of the two. The stairs "s1" are thus a part of the ground floor, and together with the room "r01" they form the stairway "sw01" from the ground up to the first floor. Geometric areas outside doors are not included for the sake of clarity.

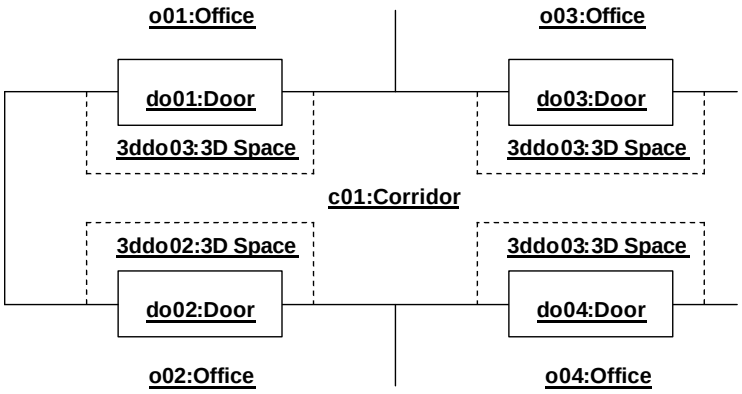

Figure 14: Geometric areas outside doors

Validation of the domain model was done in two steps: first by checking if the model fulfills the required properties of location models, then by attempting to answer the following queries:

- **Query 1**: You are in the reception. How do get from here to office "o02"?
- **Query 2**: You are standing in front of a door. Is there any way you can find out which door you are standing in front of?
- **Query 3**: Is it possible to find out if you are in an elevator or stairway? If so, which floor are you in?
- **Query 4**: Is an employee in his office?, If not, then where is he?

4.7.2 Validation Results

The domain model fulfills the required properties of accuracy, dependency, flexibility and representation of objects and relationships (chapter 4.3) acceptably. The first one, accuracy, depends on the particular needs the location aware application is meant to cover. In our case, we wish to track the position and status of different moving objects within buildings. The domain model shown in figures 8, 9 and 10 contains the necessary constructs in great enough detail

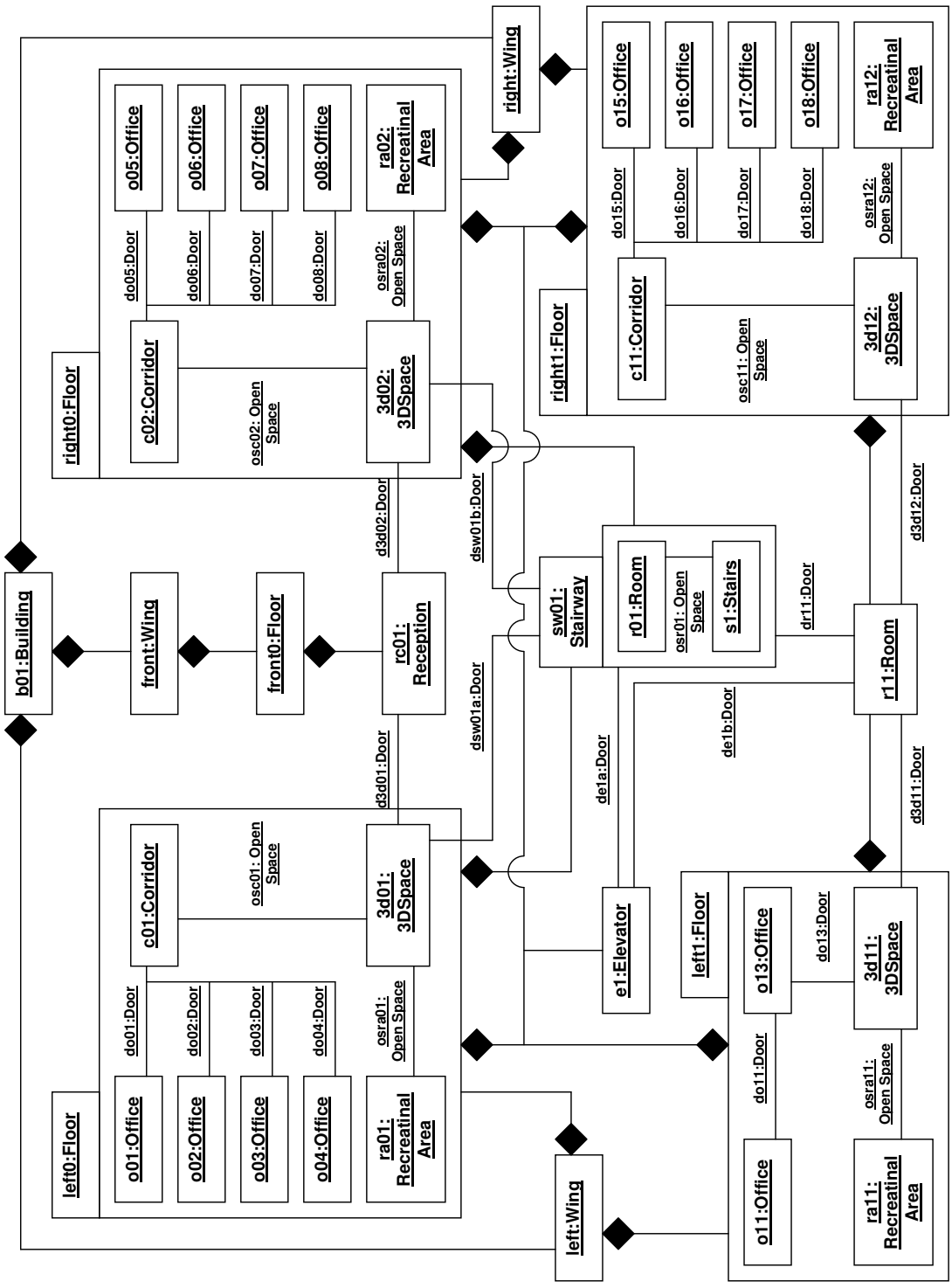

Figure 15: Location model of the building described in figure 13

for this. Examples of such constructs are the building itself with its spatio-temporal properties, structure divided into wings, floors and rooms, and events occurring in it.

Representation of objects and their relationships are given so that they correspond closely to their real world occurrences. For example, a building is divided into floors, wings, rooms, spaces and offices, along with doors and open spaces to indicate transitions between them. Relationships between spaces are shows as containment or through transitions. This corresponds to how buildings are in reality. Objects within the building are modeled as separate entities with their own properties and are associated with different spaces based on their current position.

Dependency is shown as spaces containing others using the UML-symbol of composition. This indicates existence-dependency such as a building containing floors which in turn contain rooms. If the building ceases to exist, so do the floors and thus also the rooms.

Finally, regarding the issue of flexibility, new spaces can be added to others without having to change any of the higher levels in the hierarchy. For example, an office can be added to a floor without any consideration to the building as a whole. Similarly, spaces can change their symbolic names without affecting any of the other constructs in the model. Changing geometric information is somewhat more complicated, because this might affect other spaces. It is up to the application to resolve problems that might occur in these cases.

The queries asked in the previous section were also answered in good enough detail to validate our domain model. The result of the queries were:

- **Query 1**: Office "o02" is in the first floor ("left0"), left wing ("left"). In order to get there from the reception ("rc01"), one must go through door "d3d01", cross the transition "osc01" and go through the door "do02" which leads to office "o02". The model clearly shows this path.
- **Query 2**: An area outside a given door can be defined as its own geometrical area (figure 14). Such an area is numbered so that it "belongs" to

that particular door. For example, the geographical area outside the door "do02" is numbered "3ddo02". Standing within this area thus means that one is standing outside door "do02".

- **Query 3**: The elevator is modeled as a space of its own. We can therefore determine if a person is inside it. We can use the geometrical information in the model to find which floor the elevator is currently in. Stairways are also modeled as spaces of their own. Stairways belong to the lowest of the two floors that they connect. Because of this, being in the stairway between two floors is regarded as being on the lowest one.

- **Query 4**: The domain model shows who occupies which offices. These persons are also considered as moving objects, so their position can be determined at any given time. To answer the query, we simply check whether or not the current position of a person in within the office he occupies.

Based on fulfillment of the location model properties and results of the queries, we deem our domain model as complete regarding the modeling needs of indoor location aware systems. As previously emphasized, the model can be extended to include other types of spaces, objects, events and geometries. The hospital monitoring system scenario presented in chapter 4.2 is a good example of this, where we need to further extend the domain model by introducing new classes such as "Emergency Room", "Surgery", "Morgue" and other hospital related rooms. These new classes would be subclasses of "Space" and thus inherit all the properties of "3D Space". Other extensions could be "Hospital Bed" derived from the class "Object" and "Patient" from the class "Person". A new relationship "lies-in" could then be associated between "Patient" and "Hospital Bed", showing which patients currently lie in which beds.

The basic classes however, which describe the relationship between 2- and 3-dimensional spaces, objects, persons, roles and events are essential for all location aware applications.

4.8 Research Question 1

Our first research question was (chapter 1.5): *What are the requirements of location modeling and how do we fulfill these requirements in order to create location models used for symbolic as well as geometric positioning*?

The purpose of this chapter has been to present the basic concepts and requirements of location information and modeling. Different approaches have been presented and a modeling technique which includes both symbolic and geometric information is given. The proposed model contains objects and relationships which were deemed as important for indoor location aware applications. It is also suited for nearest-neighbor- as well as range-queries. This chapter therefore answers the first research question.

The proposed model is created using UML, with some changes to the syntax in order to make models created with this technique less complicated. The model will be used throughout the thesis.

5 Spatio-Temporal Database Systems

In this chapter we present the concepts of spatio-temporal databases in more detail. The basis of our discussion is work carried out by *TimeCenter* and research done in the *CHOROCHRONOS* project. TimeCenter is an international center for research on temporal database management. Some work on spatio-temporal aspects has also been conducted by TimeCenter [91]. CHOROCHRONOS was a *Training and Mobility of Researchers Network* funded by the *European Commission*. Its objective was to study the design, implementation and application of spatio-temporal databases. Participating researchers from different European universities contributed to the research and studies conducted during this project. Their results and contributions are gathered in [43].

5.1 What is a Spatio-Temporal Database?

We sometimes need to access data in association with time. We might for example need to know where a person was at a particular time, or when a weather-phenomenon will reach a certain city. Often, this data also has a spatial nature, like persons moving from one geographical location to another or weather-phenomena covering a geographical area. This combination means in practice that we are referring to geometrical data that moves over time. In these cases, what we essentially wish to know is *when* and *where* our desired data is valid.

Spatio-temporal databases deal with the integration of time and space. Since this integration yields a dynamic combination, we can say that spatio-temporal databases handle moving and/or changing geometrical objects [20, 21]. Persons and weather-phenomena are examples of geometrical objects, because they are related to positions in space. They additionally have the ability to move, and weather-phenomena also have the ability to change (grow or shrink). The purpose of spatio-temporal databases is to track the movements and changes of these objects according to time. This capability can be used in many different applications, a few of which are listed below:

- Applications that track and determine the movement of individuals or groups of people/objects. This can for example be the movements of a particular person over some time, migration in the animal kingdom or replacement of military troops. Relevant questions might be: "What distance did they traverse?", "in which direction did they move/grow/shrink?" or "at what speed did this person move, and what was his top speed at a certain time?"

- Observation of weather-systems such as high/low pressure areas, storms, typhoons and hurricanes. Applications in this field could be used (and are indeed currently being used) to track the movement and extent of these weather phenomena. Questions like "how is this low pressure area expanding?" or "will this typhoon reach Bergen?" can be answered by spatio-temporal applications[13].

- Spatio-temporality can be used in applications for geographical surveys related to shiftment in land areas, movement of glaciers, growing/shrinking of forests or changes in country borders. Interesting questions are for example: "What was the largest extent of a given country at some time in history?" or "how fast is the Jostedalsbreen (a glacier in Norway) shrinking?"

As we see from the examples above, spatio-temporal data can be used to in some extent predict the future movement/change of geometrical objects. Such data can also be used in backward projections, that is, to show historical information about geometrical objects in relation to their position, movement, change, speed and direction. The examples also show that spatio-temporal data can be used to describe geometries which change in a continuous manner (i.e. a person moving) and geometries which change in discrete steps (i.e. a building changing its architecture) [21, 22].

The examples cover a vast area of applications, from cadastral to meteorological systems. We however, are only interested in *indoor location awareness*. Our

[13]This is true to some extent only. We are dealing with meteorological phenomena here, which is a science based on probability. There will therefore be a level of uncertainty with the predictions.

main focus is thus on continuously and discretely moving objects (e.g. persons and office appliances) within discretely changing architectural constructs (e.g. buildings). This has several practical areas of use, a few of which are presented in the following.

5.2 Practical Use of Spatio-Temporal Databases

We showed in chapter 4.2 that location models can be used to answer queries regarding the present status of spaces, objects, events and persons within the application domain. However, if we need the ability to query historical data, we must store information from the location aware application in a spatio-temporal database. Using the examples given in chapter 4.2, spatio-temporal databases can be useful in the following scenarios:

Office Management System

A spatio-temporal database can be used to store and query data about previous positions and roles of employees. This information can be used to determine the past position, status and role of persons and other office appliances. The database can for example be used to find out where certain employees were at given times, or which employees were in the same room at the same time.

Queries can also be related to the roles that employees might have, e.g. how often were project managers located in offices of one of their project co-workers, or how many meetings did a certain project manager attend during a given timeframe? We can also ask about the status of objects, such as which printers required the most maintenance, or how long a printer was operational since it was last repaired.

Hospital Monitoring System

The same type of data as in office management systems can be stored in spatio-temporal databases for hospital monitoring systems. The database can be asked about previous positions of equipment and personnel and the answers can be used to better administer these resources. This could for example be relocation of personnel to areas where they were most often located.

Other queries could be related to events and roles, e.g. how often were nurses in the same room as the "attending surgeon", or how many nurses in average attended events such as "heart-surgery" or "physical examination". Queries could also be asked to determine which employees most often meet with others, such as nurses meeting with doctors. Although spatio-temporal databases are incapable of explicitly answering queries on interaction between two or more individuals, an assumption can be made that if the distance between individuals is small enough over a long enough timeframe, it is highly probable that those individuals interacted in some manner.

5.3 Conceptual Model

Conceptual models are used to express the requirements of applications without the use of computer metaphors. Well known conceptual models include the *Entity-Relationship model (ER-model)* [12] and *Unified Modeling Language (UML)* [92]. We have already used UML to create a domain model containing the main objects and association necessary for indoor location aware applications, including events, objects and persons associated with different spaces (chapter 4.7, figures 8, 9 and 10). Unfortunately, the domain model does not capture the internal attributes and properties of these objects and associations. We are particularly interested in specific spatial and temporal properties associated with objects, events and actions that may exist or occur within the domain. Such event are for example people moving around in the building, or locations (i.e. rooms) changing shape or name. To model these aspects, we need a conceptual technique suited for spatio-temporality.

Tryfona and Jensen propose a *Spatio-Temporal ER-model (STER)* by extending the ER-model with spatial, temporal and spatio-temporal constructs [63]. Their proposal can be used to model spatio-temporal applications based on geometrical objects with added temporal properties. In a revised version of their proposal, including work done by Rosanne Price, they extend the UML syntax in the same manner [64]. We adopt the extended UML-solution proposed in [64] and extend the domain model from chapter 4.7.

We use UML because it is considered a stronger tool than ER for modeling

databases. The reason for this, as mentioned by Naiburg and Maksimchuck [48], is that UML allows us to:

> *[...] model elements such as domains, stored procedures, triggers, and constraints as well as the traditional tables, columns, and relationships.*
>
> *[48], p. 123*

This simply means that UML allows for a more accurate database model, containing details which are hard to model using the ER approach.

Since we also used an extended form of UML for location modeling (chapter 4.7), using UML instead of STER in the database modeling process reduces the complexity of the application development. This is due to the fact that we now can merely extend the domain model with necessary attributes and relationships to obtain a spatio-temporal database model. However, before we proceed, we must define the required spatial and temporal aspects.

5.3.1 Spatial Aspects

According to Güting and Schneider [28, 29] geometrical objects can be constructed using the fundamental abstractions of *points*, *lines* and *regions*. They also refer to these *Abstract Data Types (ADTs)* with the term *spatial attribute values*, or *Spatial Data Types (SDTs)*:

- **Point**: A point is used to represent the geometrical aspect of an object in space where only the object's location, not its extent, is relevant. For example, a person in a building may be modeled as point. The same can be done for a city on a large scale map.
- **Line**: Connections in space or movements through space are represented by lines. A line can be viewed as a curve in space, often represented by a polyline (sequence of line segments). Examples of lines are roads, rivers and different types of cables.

- **Region**: A region is used to model objects that have an extent of some sort in either 2- or 3-dimensional space. Such objects may be countries, lakes, or cities on small scale maps. Regions may be disjoint (consist of several pieces) and may even contain holes.

These SDTs have several common properties, such as coordinates within a reference system and operations that are valid for all three types, like calculation of distance or containment. We therefore extend the proposal of Güting and Schneider [28, 29] by adding an abstract general type called *geometry* which defines the common features of its specialized instances point, line and region (see figure 16).

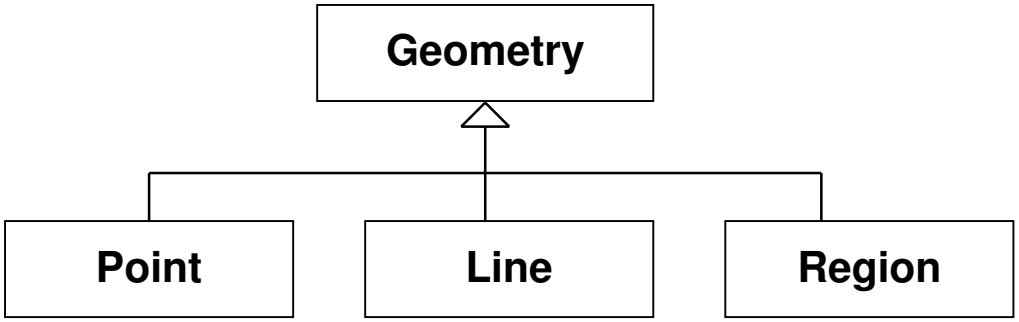

Figure 16: Geometry as superclass of Point, Line and Region

We assume that geometrical objects in the domain model can be expressed using these spatial constructs. For example, 2- and 3-dimensional spaces can be expressed using regions, while different objects (movable or static) can be expressed as either regions (if their extent is relevant, i.e. a table) or points. Lines (including curves) can be used to project the path of movable objects.

The above mentioned SDTs have no temporal dimension, i.e. they are *non-temporal*. Since we are dealing with spatio-*temporal* data, we need to include a temporal type instant (*time*) [31].

5.3.2 Temporal Aspects

Time is used to describe the temporal aspects of objects, attributes and relationships in the database. It can be used to define a period of validity or the

beginning of an event for which we don't know the ending time of. These aspects can be classified into the following *Temporal Data Types (TDTs)* [26, 63]:

- **Transaction time**: Transaction time is used when the temporal aspects of an element (i.e. object, attribute or relationship) evolve discretely. When an element is inserted into the databases, its temporal attribute has the form $[t_{start}, now)$, indicating that the ending time is unknown. Transaction time is used to trace past states of objects. Updates are only allowed on the present version of the element, and deletion is performed logically as opposed to physically. Once an element no longer exist within a database, its temporal attribute is given the form $[t_{start}, t_{end}]$. This means that elements that are no longer considered present in the database are still represented by a record, stating their past. The transaction time of an element is thus the time in which the element is a part of the current state of the database.

- **Valid time**: Unlike transaction time, valid time is used when the temporal aspect of an element change continuously. Valid time is applied to facts and events and used on object attributes and relationships between objects. On insertion, the starting and ending times are given in the form $[t_{start}, t_{end}]$. This shows the time when the fact or event was valid in the modeled reality. Deletion is performed physically, so the history of a deleted valid time attribute or relationship is not captured in the database.

- **Bitemporal time**: Bitemporal time is used to trace the evolution of a dynamic collection of valid time facts. This is a combination of transaction and valid time. It shows time as a set of intervals, where each interval (i.e. pair of timestamps) is valid time, except for the last interval which is transaction time. It has the form $[[t_{start}, t_{end}], [t_{start}, t_{end}], ..., [t_{start}, now)]$, where the intervals $[t_{start}, t_{end}]$ indicate its valid time properties, while the interval $[t_{start}, now)$ indicates its transactional properties. The valid time properties show when a fact was valid in the database. This might for example be the previous positions of an object. The transactional

property shows the present version of an element in the database, e.g. the current position of an object.

- **Existence time**: Existence time[14] is used on objects only, since it indicates the existence of an entity. Existence time of an object *obj* can be viewed as valid time for the related fact "*obj* exists". However, as Gregersen and Jensen imply [26], it is important to consider the aspects of valid and existence time separately because the recording of existence time is important in many applications.

These temporal aspects have one ting in common: they all have a *duration*. This common capability is captured by the general concept of *time*, i.e. transaction, valid, bitemporal and existence time are all specializations of time.

We assume that temporal aspects of objects in the domain model can be expressed using these constructs. For example, objects such as persons have existence time. This indicates the time interval in which that particular person existed as an object in the database. Relationships like persons occupying offices have transaction time. This shows the time interval from when a person occupies a particular office till the present. If the relationship changes in any way, e.g. the person moves out, the attribute in the database containing the temporal property of that relationship is updated. The position of a movable object (including a person) is recorded with bitemporal time because such an object can be considered to be in different locations at different time intervals. We do not use valid time alone in any cases, because this would conflict with our desire to trace previous information.

5.4 Extended Domain Model

Now that we have defined the necessary constructs and presented our assumptions, we can extend the domain model by adding spatial, temporal and spatio-temporal properties to classes and relationships. We do this by using the extended spatio-temporal UML symbols defined by Tryfona et al. [64]. The extension consists of five new symbols, shown in figure 17.

[14] Also referred to as *lifespan*.

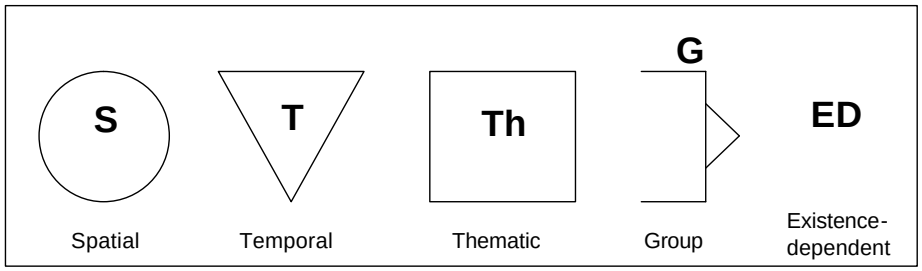

Figure 17: Extended UML spatio-temporal symbols, as shown in [64]

The spatial symbol (circle with an "S" inside) represents spatial aspects of objects as spatial data types available in the domain. The triangular symbol with a "T" inside represents temporal properties of elements. Also referred to as timestamp, it can be used to represent existence time for objects, valid time for attributes and relationships, and transaction time for objects, attributes and relationships.

Thematic data, i.e. data related to a particular type or topic is represented with a "Th" inside a square. This symbol describes attributes that are somehow related to space and/or time. It is is used together with the spatial and temporal symbols to "indicate changes to thematic data across time and space" [64], p. 105. An example of this may be the population density of a city. The value can be given as an integer that changes with respect to time and space (where space is the geographic extent of the city), shown in figure 18.

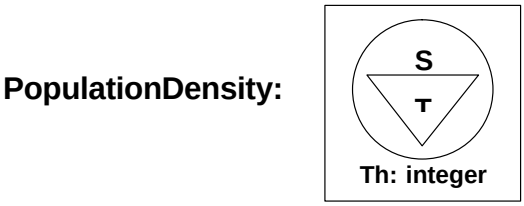

Figure 18: Example of thematic data that changes across time and space, as shown in [64]

The group symbol is used to group attributes with common spatio-temporal properties together. It does not describe any spatio-temporal properties, but is

used to reduce the complexity of the model. Finally, attributes or relationships that are dependent of object existence are marked with "ED".

The added symbols alone are not enough to specify exactly what particular spatial, temporal or spatio-temporal types that are required. To specify detailed spatio-temporal semantics, *Specification boxes* are used. These boxes specify the necessary spatial types based on a *SpaceModel*, while temporal aspects are given as a *TimeDimension*. The SpaceModel of an object can be any valid spatial data type defined within the domain. In our case, valid types are points, lines and regions. The same is true for TimeDimension. Here, available types are transaction, valid, existence and bitemporal time. A combination of SpaceModel and TimeDimension is used for spatio-temporal types.

Specification boxes are added as compartments to object classes. They are inherited from parent classes as with any other class property. Specification boxes of associations can be placed in the specification compartment of either of its participating classes.

Using these proposed extension on our domain model, the conceptual model for a spatio-temporal database is presented in figures 19, 20 and 21. The model has been divided into three for the sake of clarity, and only classes with spatial, temporal or spatio-temporal properties have been extended. These classes now contain necessary attributes, such as identifiers, names and spatial types, along with temporal properties of both attributes and relationships.

Figure 19 shows the classes "3D Space" and "2D Space" with their geometrical and temporal properties. The attributes "name" and "extent" in both classes have the temporal property of transaction time. This is because both names and extents of spaces can change and are thus time-dependent. We must therefore be able to trace past values of these attributes. The same is true for all attributes where a need to trace past values is required. In addition, both classes have the temporal property of "existence time", indicating the period in which they existed.

The relationship "has" between the classes "3D Space" and "2D Space" is temporal, i.e. a 3-dimensional space may be added transitions, or have them removed at different times. This relationship is also "existence dependent (ED)"

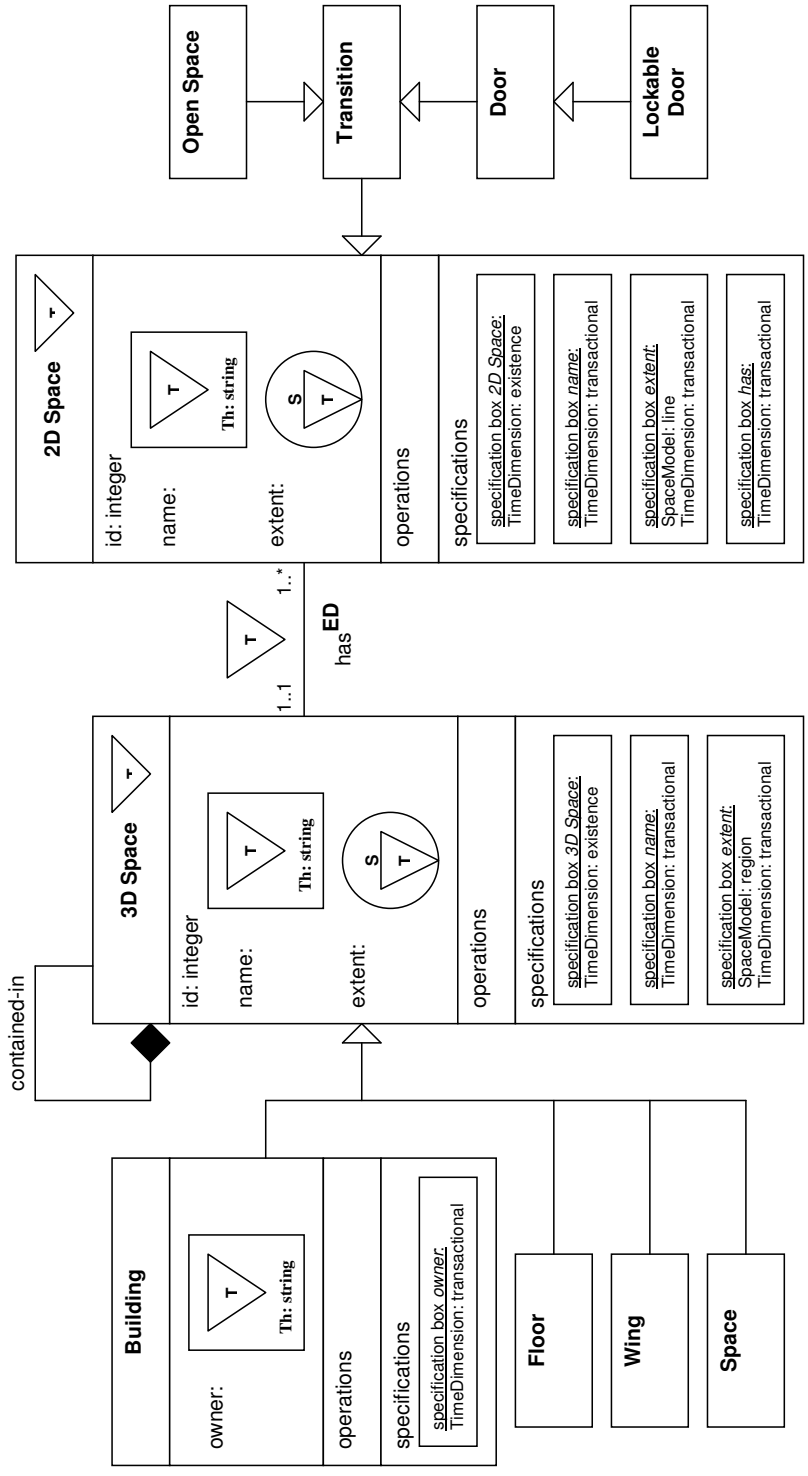

Figure 19: Extended domain model of 2- and 3-dimensional spaces

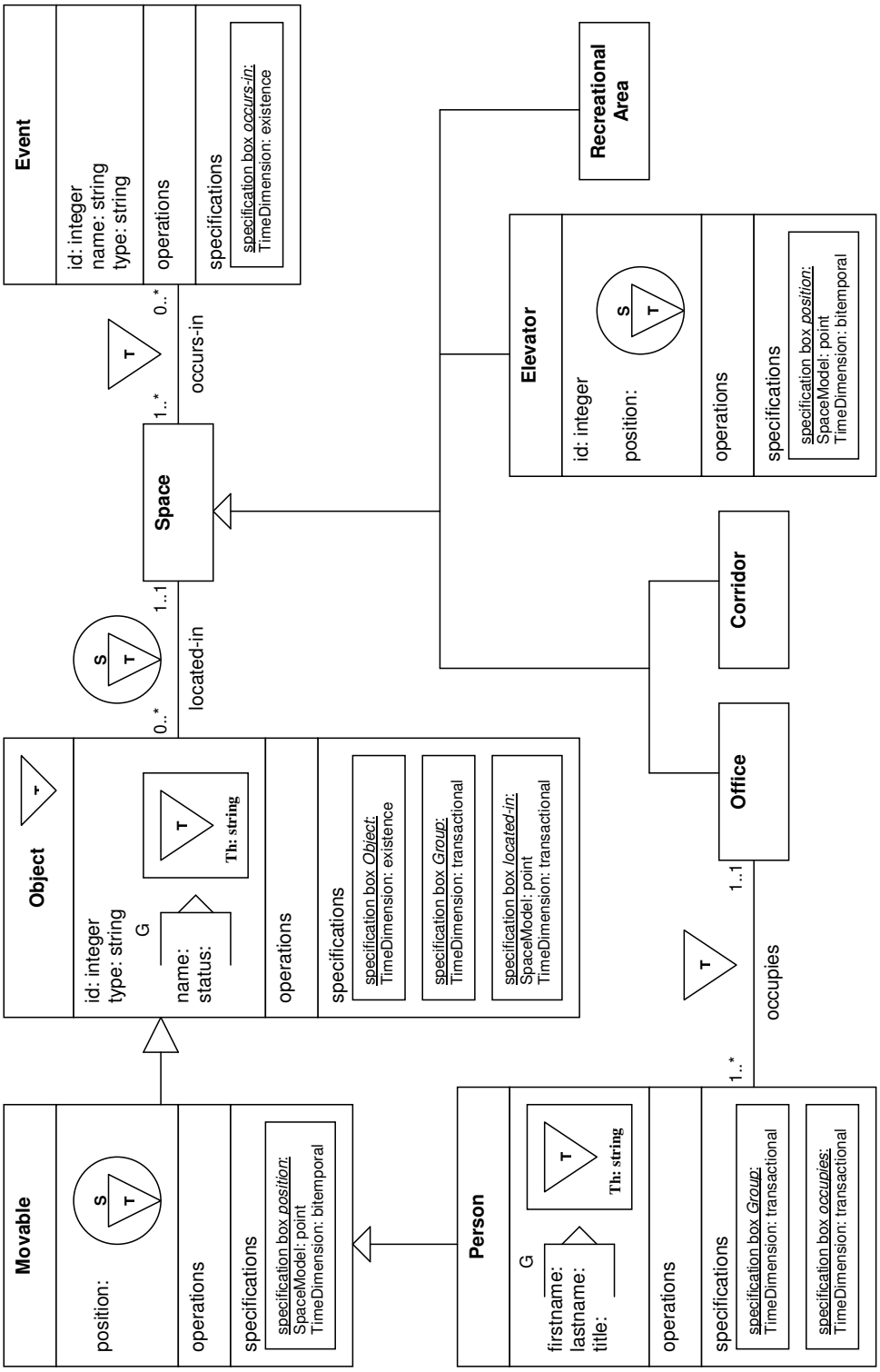

Figure 20: Extended domain model of objects and persons located in spaces

on “3D Space”. If a 3-dimensional space ceases to exist, so do all its transitions.

The classes “Building”, “Floor”, “Wing” and “Space” inherit all their spatio-temporal properties from “3D Space” and have non of their own. Additionally, buildings may have different owners at different times. This is regarded as a temporaly dependent attribute, and is shown in the model as “owner”. Since ownership of buildings change in discrete steps, the attribute’s temporal dimension is transactional.

Figure 20 shows the position of an “Object” as a point with transactional time. This is due to the fact that objects which are not considered as “Movable” change their positions in discrete steps. Moving objects however, normally persons, change locations frequently and their positions can be given as continuous intervals of time. The positions of such objects therefore have bitemporal properties.

As was the case with 2- and 3-dimensional spaces, so do all objects have existence time, indicating when they existed in our modeled reality. This also includes movable objects and persons, since they inherit the properties of class “Object”.

Elevators inherit all the properties of “Space”, such as id, name and extent, but have an additional spatio-temporal attribute “position” which is recorded with bitemporal time. This is necessary for the same reason as with movable objects, i.e. because elevators continuously move and are located in different floors at different times.

The temporal aspect of an “Event” occurring in a space is captured with existence time. Existence time is used because events exist in a short given time interval for which we know both the start and end time of. This property belongs to the relationship “occurs-in”.

Finally, figure 21 shows the relationship between persons and the different roles that they might have. This relationship, “has-role”, is time dependent, i.e. it has a temporal dimension which suggests that the roles are kept by persons during given time intervals. The specification box for this relationship is depicted in the class “Person”.

Now that we have extended our domain model to include spatio-temporal prop-

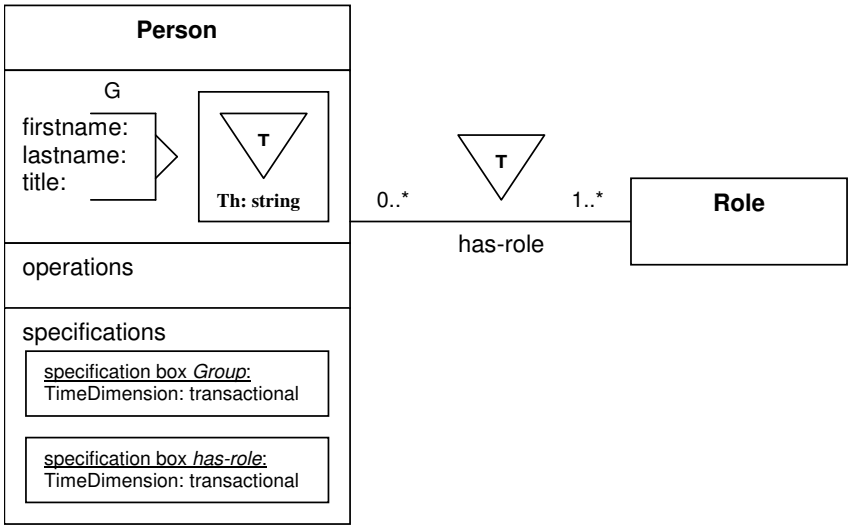

Figure 21: Extended domain model of persons and roles

erties, we must define a *data model* which is suited for implementation of these aspects.

5.5 Data Model

A *data model* is a scheme for describing data structures with their data types, integrity rules and operations. The data model shows what type of information a system can process, and how it processes it.

A complete formal definition of a data model for spatio-temporal databases along with the full extent of data types, available operations, their signatures and semantics is found in "A Foundation for Representing and Querying moving Objects" by Güting et al. [31]. In the following, we present basic concepts of this work, including practical problems associated with it, and introduce our own approach for a spatio-temporal data model.

5.5.1 Spatio-Temporal Data Types (STDTs)

Erwig et al. [20, 21] propose a 3-dimensional (2-dimensional space + time) approach based on ADTs together with a set of operations to capture the be-

havior of moving and evolving geometrical objects. Although their work only covers 2-dimensional space, they assert that the same approach can be used for 3-dimensional space by adding a third axis to the coordinate system[15]. They define a *moving point* as the basic abstraction if only the position of an object in space is relevant. If also the extent is of interest, the basic abstraction of *moving region* is used. Moving lines are normally not considered in the same sense, since lines themselves are abstractions or projections of movement. However, a *moving line* can be used to describe a river or road changing its trajectory. They also introduce new names for all moving types by prefixing the argument with an "*m*", such as *mpoint* for moving points.

Using our proposed abstract spatial data type *geometry* to describe a point, line or a region in 2-dimensional space and the purely temporal data type *time* to describe the valid time dimension, the spatio-temporal data type (STDTs) *mgeometry* (moving geometry) can be viewed as a mapping from time into space [20, 21]:

$$mgeometry \quad = \quad time \rightarrow geometry$$

According to this equation, a type *mpoint* describes a position as a function of time. The value of this type can be represented as a curve in a 3-dimensional coordinate system (x and y for space and t for time). Similarly, a type *mregion* represents a region as a function of time. The value of this type can be represented as a set of volumes in the same 3-dimensional coordinate system (see figure 22 a and b).

Standard data types can also be "moving", i.e. changing with respect to time. Examples of these are *minteger* and *mreal* for time changing integers and real numbers. The concept of moving standard data types may seem odd, but they are important and useful nontheless. Moving real numbers can for example be used to describe the distance between two moving objects. Since the objects are moving, it is only natural to assume that also their distance will change over time.

[15]This would in effect give us 4 dimensions: 3-dimensional space + time.

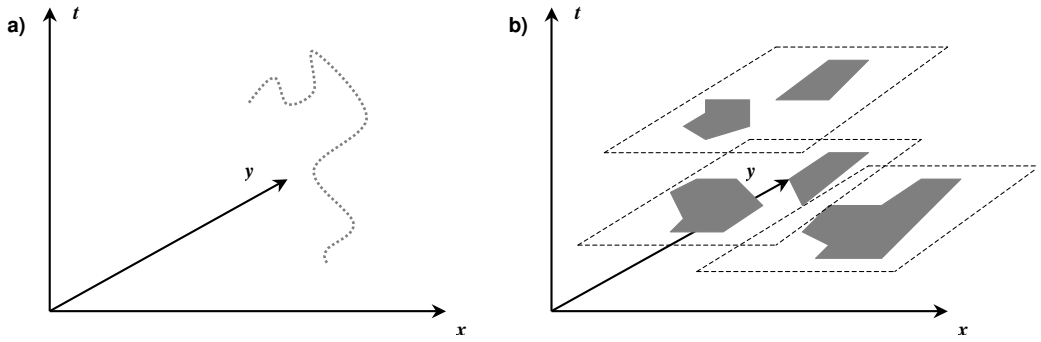

Figure 22: Representation of a moving point (a) and a moving region (b) in 3-dimensions (2-dimensional space + time)

Based on these STDTs Erwig et al. [20, 21] and Güting et al. [31, 43] define a set of operations to support different types of calculations concerning relationships between moving geometric objects. Examples of such operations are:

mgeometry × *mgeometry*	→ *mreal*	**mdistance**
mpoint	→ *line*	**trajectory**
mregion	→ *region*	**traversed**

The operation *mdistance* requires two moving geometries as operands. It returns the distance between these objects as a time changing real number. Projection of a moving point onto the plane is done with *trajectory*. This operation requires one operand, a moving mpoint, and returns a line which describes the "path" of the moving object. A similar operation, *traversed*, is defined for moving regions. It returns the total area ever covered by a moving region as a region.

These operations are merely a few examples used to describe the basic concepts of the data model. Other operations such as purely spatial and purely temporal ones are also required.

5.5.2 The Spatio-Temporal Data Model in Practice

Despite the fact that the proposed model is defined in great detail and covers most aspects of spatio-temporal database management, there are issues that

make it difficult to use in practice.

First of all, there are no database system vendors that offer support for the proposed data model. The defined data types and operations must be implemented manually, i.e. programmed as user defined data types and stored procedures. This leads to a great deal of effort in the application development. Although some additional operations have to be implemented for almost any approach we might take, using spatial extensions and supporting these with temporal data types is a less cumbersome approach than implementing a complete data model.

Another problem is that concepts such as different temporal types are not dealt with by Erwig et al. [20, 21] and Güting et al. [31]. In their proposed data model, time is not classified as transactional, valid, bi or existential. It is considered in a simple continuous manner. We however, feel that it is important to consider different aspects of time as classified by [63, 64, 26]. This is because different elements in the model require different temporal properties.

Because of these apparent shortcomings of the spatio-temporal data model, we adopt a timestamping approach to spatio-temporal data management.

5.5.3 Timestamping Data

We use the term *timestamping* as adding a temporal value to nontemporal data (including spatial) to register the time in which the data is valid, exists or is part of the current state of the database. As guidelines on how to efficiently timestamp data, we follow the work done by Torp et al. [62] in "Correct and Efficient Timestamping of Temporal Data". Although they do not cover any spatial data in their work, the general approach can be used to add a temporal dimension to any type of data which requires it.

An example of timestamping data is presented using the scenario of a person working in different departments during different time-periods. The current department is given as an attribute which changes its value over time as the person changes department. Because of this, the value of the attribute is valid only over a given time-period. To capture this temporal property, Torp et al. suggest we timestamp the attribute by registering this time-period in the form

$[t_s, t_e]$, where t_s indicates the beginning of the time-period and t_e indicates the end. t_e can be left undefined (null), meaning that the value of the attribute is valid from t_s until changed. Once the person changes department, that is the attribute changes value, we set t_e to the current time, showing the valid time period of the attribute's previous value [62], pp. 6.

Timestamping in this manner means that we do not perform any physical deletes in the database, but only logical ones. This is an important point in our work, since we wish to keep the past status of objects present in the database for querying.

This approach can be used to implement transactional and bitemporal aspects defined in chapter 5.3.2. Existence and valid time are also implemented using timestamps, but in the case of valid time, t_e cannot be left undefined. Existence time can be derived from transactional time as the minimum t_s and maximum t_e of an object which no longer has a t_e that is undefined (i.e. the objects no longer exists).

Güting et al. [31] argue that such separate representation of nontemporal and temporal data is a poor solution, since it does not capture continuous change. We however, argue that by using bitemporal time, we can indeed represent continuous change. This is because bitemporal time consists of intervals of valid time, which when made small enough can be deemed as continuous. Consider the following example:

Imagine a car moving along a road. We wish to track its position continuously. Imagine now that we, as database developers, decide that if an object is capable of changing more that once every 10 seconds, we consider its change to be continuous (remember: movement was considered as a form of change). It is reasonable to assume that a car in movement changes its position more that once every 10 seconds, so we stamp its spatial property (position) with bitemporal time, setting the granularity of each time interval to 10 seconds. Using this approach, we store the previous positions of the car in the database as facts valid over an interval of 10 seconds, while the current position is stored in the final interval given by the bitemporal property. Thus, all continuously changing objects can be stamped with bitemporal time where the duration of intervals

are set by the frequency of change deemed as continuous by the developers.

Discrete change can be captured using the same timestamping approach. Objects which move or change their geometric extent in discrete steps, such as buildings, can be stamped with transaction time.

5.6 Access Methods and Query Language

The spatial data types presented above must be integrated into a database data model before they can be used. We rely on the *object-relational* model for our implementation and illustration purposes. The object-relational model uses the declarative standard database sublanguage[16] SQL for data access [8]. SQL is arguably the most comprehensive standard in database management systems. In the upcoming sections, we justify our choice of object-relational over other data models and briefly introduce SQL and propose extensions in order to make it suitable for handling spatio-temporal data.

5.6.1 Why Object-Relational?

There are several reasons for our choice of object-relational over object-oriented or purely relational:

First of all, there is a lack of standards and experience in the field of object-oriented database management. An object-oriented solution would therefore not be general enough. There is also a lack of proper query languages, which due to encapsulation, makes it difficult to retrieve data about one or more objects simultaneously.

Secondly, issues such as security, data integrity and crash recovery are not dealt with as comprehensively by the object-oriented model as in the relational. This reduces the overall performance of the database system. When it comes to standardization, experience, security, concurrency control and crash recovery, the relational model by far exceeds the object-oriented.

[16] A database sublanguage is a programming language designed specifically for initiating database functions [8].

Finally, relational database systems have a strong commercial position are likely to remain there for quite some time. Their vendors have therefore been forced to create extensions to make their products cope with more complicated data management. This has lead to the development of several extensions for relational databases which make them more suited for managing complex data, hence the term post- or object-relational.

Although database systems based on the relational model have some clear advantages over object-oriented ones, they too have some limitations. These are among others support for complex data types and inheritance. The relational model is based on a fixed set of base data types (integer, real, etc.) and relational algebra operations (based on comparators from base data types). It does not allow definition of user defined data types and stored procedures. This strongly reduces development flexibility because database developers are left with fewer possibilities to define data types and operations.

The object-relational approach combines the strengths of the object-oriented and relational model. It supports complex data types (object-oriented) as well as the need for complex querying (SQL) [59]. It also maintains the strong integrity, concurrency and security features of the relational model. Besides these features, the object-relational model also allows the definition of *triggers*[17]

5.6.2 Structured Query Language (SQL)

SQL was developed by *IBM* in the 1970s. It was originally named *SEQUEL* and was implemented as a language to support IBMs prototype of a relational database system called *System/R*. The name was later changed to SQL.

In the 1980s, several computer system vendors, including IBM, released database systems with SQL as their query language. It thus became clear that SQL was here to stay, and work on standardization began. This work was done by the *International Standards Organization (ISO)* and the *American National Standards Institute (ANSI)*. The first standard, *SQL1*, was released in 1987 [35].

[17] A trigger is a user defined unit of logic embedded in the database and activated when certain events occur, for example on insertions, deletions, or updates.

However, due to criticism about it not fully showing the relational constructs, a new extended version with integrity enhancements called *SQL2*[18] was released in 1992 [36].

The latest version of the standard, *SQL3*, containing further enhancements to SQL2 was produced in 1999 [37, 38]. SQL3 provides mechanisms for creation of user defined data types and stored procedures in SQL. This is a prerequisite for the definition of spatial and temporal data types and operations. The added features in SQL3 are an attempt to extend the relational model by adding "object-oriented-nes" to tables. This is what we refer to as post-relational, or object-relational.

The different features of SQL can be divided into three separate groups: *data definition*, *data integrity* and *data manipulation*:

Data Definition

Data definition in SQL involves commands for declaring the structure of data. As mentioned previously (section 2.4.2), data in the object-relational model is stored in relations, also referred to as tables. Tables consists of columns, where the values in a column are all of the same type. The table name and column names are decided by the user. The data types can be standards like integer or string, or more complex data types defined by the user. A column may be specified as NOT NULL, which means that it can not be left "empty", i.e. the user is not allowed to enter null values into it. The use of the command UNIQUE prohibits the occurrence of the same value in a column. A column may also have a default value assigned to it.

Data Integrity

Integrity issues in SQL deal with *primary* and *foreign keys*, *propagation of changes to data* and *domain integrity*. A primary key may consist of several columns and is used to set a unique identifier for a row in the table. Foreign keys are used for referential integrity by propagating changes in a referenced column in another table. A table can only have one primary key, but several foreign keys. Propagation is performed using the commands ON DELETE and

[18] Also referred to as *SQL92*.

ON UPDATE, where the action may be to cascade the changes, set a default or null value, or no action at all. Finally, the command CHECK is used for domain integrity by making sure that values inserted into a columns fulfill the condition.

Data Manipulation

Data can be *inserted*, *updated*, *deleted* and *selected* from tables. Data can be retrieved using the SELECT command and by specifying selection criteria in a WHERE clause. A selection from several tables can be performed by joining them on a common column. The join-condition is given in the WHERE clause. The condition may itself be a SELECT statement, referred to as a *subquery*[19].

5.6.3 Extended SQL for Spatio-Temporal Data

The underlying data model of SQL3 is based on ADTs, so standard nontemporal data types like integers, reals, strings and booleans and temporal types like time, date and timestamp are implemented features of it. Because of this, only minor extensions are necessary in order to use SQL for spatio-temporal data management. Erwig and Schneider [22] propose a list of extensions based on the spatio-temporal data model. But seeing as we are not considering this particular data model any further, we adapt their proposals to our needs. Necessary extensions are:

1. A set of spatial data types, operations and predicates. These features are already available in extensions delivered by several relational database vendors. Examples are *Oracle Spatial* [88], *DB2 Spatial Extender* [71], *Spatial Extensions in MySQL* [81] and *PostGIS* for PostgreSQL [89].

2. A set of temporal data types, operations and predicates. Temporal data types such as for example date, time, datestamp and timestamp are available in all database systems. These can be used to create user defined data types corresponding to the necessary temporal aspects. Many temporal operations and predicates are also already available. Others can be created as stored procedures.

[19] A subquery in SQL is a query nested within another query.

3. A set of spatio-temporal operations and predicates. These are obtained by combining the spatial and temporal operations available in the database system.

The mentioned spatial extensions are all based on the *OpenGIS Geometry Model* defined in the *OpenGIS Simple Features Specification for SQL* [86]. The specification is proposed by the *Open Geospatial Consortium (OGC)*, a non-profit, international consensus organization that is managing the development of standards in the field of geospatial and location based services [85]. The model defines a geometrical object as having the following properties:

- It must be associated with a spatial reference system which describes the coordinate system in which the object is defined.
- It must belong to a geometry class which defines a set of common operations for all geometrical objects.

Figure 23 shows how the geometry model is constructed. It consists of classes for defining geometrical objects. The model distinguishes between *instantiable* and *non-instantiable* classes. Non-instantiable classes are "Geometry", "Curve", "Surface", "MultiCurve", and "MultiSurface", which contain a common set of properties for their subclasses. "Geometry" is a generalization of all other geometrical objects, and as the model shows, all such objects are composites of a spatial reference system. Instantiable classes are "Point", "LineString", "Polygon", "GeometryCollection", "MultiPoint", "MultiLineString" and "MultiPolygon". A "Point" represents the position of a geometrical object within the coordinate system. A "Curve" is usually represented as a sequence of points, i.e. a "LineString", "Line" or "LinearRegion". A "Surface" is defined as a "Polygon", while "GeometryCollection" is used to describe collections of geometrical objects such as points, curves or surfaces.

The model can easily be related to the work conducted by Güting [28] and Güting and Schneider [29] regarding geometrical objects. Using the terms defined by Güting and Schneider, we see that a "point" is the same as a "Point" in the OpenGIS model. Furthermore, a "line" can be seen as equivalent to a

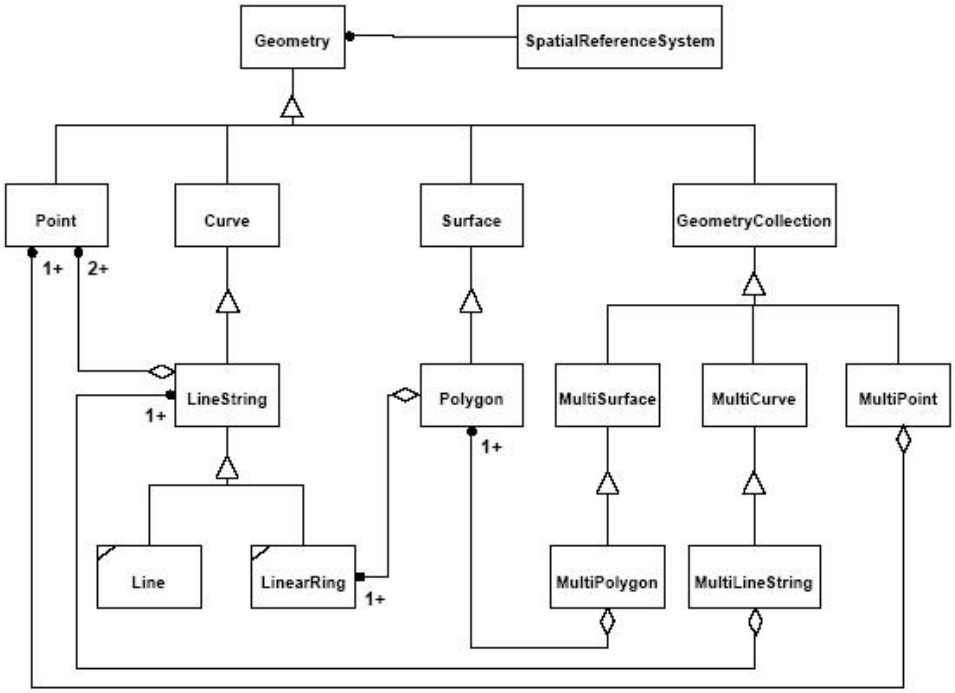

Figure 23: The OpenGIS Geometry Model, as shown in [86]

"Curve" in the model, while a "region" can be viewed upon as a "Surface". Although Güting and Schneider don't have a "GeometryCollection" type, this is no more than a logical collection of points, lines or regions.

The OpenGIS specification also defines a set of operations to use on spatial data types. Most of these operations are implemented in the mentioned spatial extensions. Others can be implemented manually as stored procedures. Of the operations presented in the specification, relevant ones for indoor location based services are:

- *contains(Geometry g1, Geometry g2)*: Indicates whether or not g1 completely contains g2.
- *within(Geometry g1, Geometry g2)*: Indicates whether or not g1 is spatially within g2.
- *intersects(Geometry g1, Geometry g2)*: Indicates whether or not g1 spatially intersects g2.

- *distance(Geometry g1, Geometry g2)*: Returns the euclidean distance between any two points in g1 and g2.

These operations are purely spatial, i.e. they do not consider the temporal aspects of moving objects. We need to add further operations in order to capture the temporal validity of the data. These can be created by defining stored procedures in the database system. Examples of such operations are given by Faria et al. [24]. The operands can be time values or objects (temporal or spatio-temporal). We propose the following operations as relevant for indoor positioning systems:

- *when(Operand op)*: Returns the valid time of op, i.e. the time when the object existed or was present in the database.
- *begin(Operand op)* and *end(Operand op)*: Returns the start and end time of op, respectively.
- *before(Operand op1, Operand op2)*: Indicates whether or not the temporal property of op1 is smaller than that of op2.
- *overlap(Operand op1, Operand op2)*: Returns whether or not the temporal property of op1 and op2 overlap (exist simultaneously).
- *duration(start, end)*: Returns the duration of a time interval having the start time given by *start* and the end time given by *end*.

Finally, we need a set of spatio-temporal operations which can be applied to objects with both spatial and temporal aspects. Such operations can be created by extending spatial operations with temporal dimensions [24], p. 4. The operands can be spatial, temporal, time values or spatio-temporal objects. Our proposal includes the following operations:

- *trajectory(Geometry g, Time t)*: Returns the trajectory of a spatial object g over a given timeframe t. The operand t can be given as a null value, indicating that we wish to calculate the trajectory of g regardless of time.

- *distance(Geometry g1, Geometry g2, Time t)*: Returns the distance between spatial objects g1 and g2 for all intervals of temporal intersections within the time interval t.

5.7 Research Question 2

Our second research question was (chapter 1.5): *How can we capture the spatial and temporal aspects of data when modeling spatio-temporal databases, and how do we implements these as data types with corresponding operations in a database system*?

We solved the issue of spatio-temporal modeling by extending the domain model of chapter 4.7 with constructs that enable it to capture the spatial, temporal and spatio-temporal aspects of geometrical objects that change. We also presented data models for supporting these aspects and how they can be implemented, for example by extending existing database systems, regardless of vendor. This is in accordance with our desire to propose general solutions which can be implemented independently of specific database systems.

6 Object-Relational Implementation

Database systems are used to effectively store and manage large amounts of information. The database can be viewed upon as a model of reality in the sense that it represents a selected set which is deemed important enough to be represented in digital form. Before we can construct the database, we must identify the selected set that we wish to represent and determine how we wish to represent it.

In chapter 4 we created a domain model with the intention of using it for indoor location based services. In chapter 5 we extended the model by adding temporal properties to the geometrical objects, making them spatio-temporal. The model thus describes objects, their properties and relationships within the intended domain and can be viewed as our selected set. It serves as the conceptual model for any spatio-temporal application, including database systems.

We begin this chapter by mapping the classes and relationships in our conceptual model from chapter 5.4 to relations used in the object-relational model. We also present in detail the timestamping approach we have chosen to capture the temporal aspects defined so far. Finally, we use the object-relational model to implement a spatio-temporal database and discuss the issue of spatio-temporal data management for indoor location aware systems.

6.1 Mapping from Object-Oriented to Object-Relational

There are several fundamental differences between the object-oriented and object-relational model. The most obvious difference is how data is structured. Although the object-relational model incorporates many of the object-oriented aspects, it still structures data in tables, i.e. relations. A relation R is written as $R(A_1{:}D_1,\ A_2{:}D_2,\ \ldots,\ A_n{:}D_n)$ where A_n is an attribute and D_n is its respective value domain. The domains can be standard types like integers, reals, booleans or strings, or more complex types like points, lines or regions.

Because of this difference, we must map the classes in our object-oriented conceptual model to relations in an object-relational model before we can implement

it. If our work is to have any realistic practical potential, the mapping process should be carried out in such a way to ensure that we capture relevant elements, concepts and associations from object-oriented to object-relational.

Guidelines for mapping from object-oriented to purely relational are given by Sparks [58]. He defines a series of twelve steps that should be followed to ensure no loss of information. Since we are not mapping to relational, but object-relational, we are not forced to follow the complete set of steps. This is because issues such as inheritance and complex data types are supported by the object-relational model and can thus be implemented as modeled (no mapping is required). We adapt the remaining steps proposed by Sparks to our needs. The following issues must then be considered in the mapping process:

Identify Persistent Classes

We begin by identifying persistent classes in our model. Persistent classes are classes that exists even when they are not the current subject of a programmer's or computer's attention. These classes are permanently stored on a persistent storage medium, such as a hard disk, and are not lost if for example the power is lost. We assume that each persistent class maps to one relation (table). Although this is a big assumption, Sparks argues that it works in most cases:

> *The logical extension of this is that a single object (or instance of a class) maps to a single table row.*
>
> *[58], pp. 16.*

Map Class Attributes

The next step involves mapping attributes to columns. Each class is given a suitable primary key (a unique ID). Other attributes are mapped to table columns. The object-relational model allows us to map complex data types such as geometrical types to single columns. We also have the possibility to define our own data types.

Map Relationships (Associations, Aggregations and Compositions)

Once we have mapped classes to tables and attributes to columns, we turn our attention to associations, aggregations and compositions. Associations are mapped with primary/foreign key pairs. Aggregations and compositions (i.e. strong aggregations) are dealt with in the same way. There are however a few important points to consider.

In cases where aggregation describes a *many-to-many* relationship, a separate primary/foreign key table is required. If aggregation shows exclusive ownership of one object over another, the table corresponding to the owned object could have an extra column containing the id (primary key value) of its owner. This column could be set to NULL if the object is currently not owned by anyone.

Strong aggregation, i.e. composition, implies that an object is composed of separate parts. The separate parts depend on the existence of the object as a whole. If the object ceases to exist, so do its parts. This requires strict integrity constraints which demand the use of foreign keys with the constraint that on deletion of the object, the part is deleted to.

6.2 The Database Model

Figure 24 shows the database model once the mapping process from object-oriented to object-relational is complete. The model is created using the UML database modeling syntax. We wish to keep the model as simple as possible for illustration purposes. Because of this, no attributes, constraints or procedures are shown at this point. This model is merely meant to show the necessary tables. Beside the mapping steps, considerations have been taken to reduce redundancy and secure integrity. Situations where these considerations cause special changes are explained in the following models, which show the different tables and associations (including cardinality) in detail.

Composition, such as a 2-dimensional spaces belonging to a 3-dimensional space is mapped to a primary/foreign key table showing the whole-part relationship. Associations of the kind *one-to-many* and *many-to-many* are also mapped to

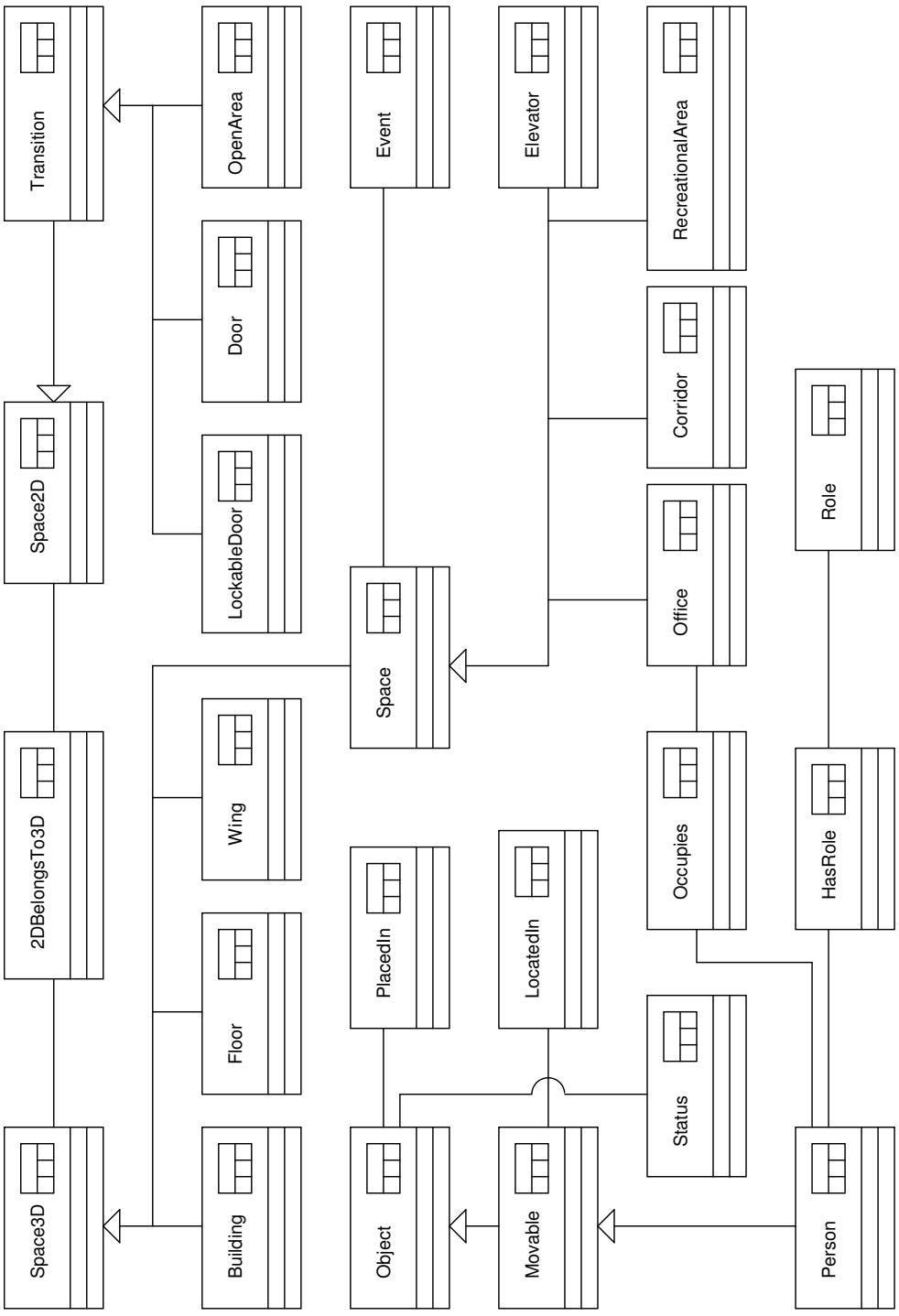

Figure 24: Main database model, without attributes, constraints and procedures

primary/foreign key tables. This is for example the case with the relationship between persons and offices, where an office can be occupied by several persons.

Complex data types such as geometrical ones are simply mapped to column attributes of the corresponding type. Temporal types such as bitemporal or transactional time can either be created as user defined types, or mapped to columns containing standard temporal attributes such as timestamp and date. Since transactional and bitemporal time both have start and end times that change, implementing them as user defined temporal types means creating composites which are difficult to maintain and update. Instead of this, we can use standard temporal types such as TIMESTAMP and DATE[20]. Which type to use in which case depends on the granularity of time we wish to model. TIMESTAMP allows for smaller intervals, e.g. seconds, which makes it ideal for modeling continuous change. Discrete change can be modeled using both types. Existence time is not explicitly implemented, since the lifespan of objects can be derived from their transactional or bitemporal properties (described in chapter 5.5.3).

Inheritance is modeled in a similar manner as in the domain model, with an arrow pointing from the child to the parent table. The child table inherits all the attributes of its parent.

Figure 25 shows the details of 2- and 3-dimensional spaces. A 3-dimensional space (building, floor, wing or space) is stored as a record in its respective table, inheriting all symbolic and geometric properties from "3DSpace". The same is true for 2-dimensional constructs, such as doors and open area transitions. These inherit their symbolic and geometric properties from "2DSpace". The table "2DBelongsTo3D" shows which 2-dimensional constructs belong to which 3-dimensional spaces. An example of this is doors belonging to an office. We could have used the same approach to model 3-dimensional spaces containing others, such as buildings containing floors, wings and spaces. However, since we can use the geometric information of spaces to find set/subset relationships, we believe its unnecessary to include an extra table for this in the database.

Geometric information can also be used to determine which 3-dimensional space

[20]We chose this approach after feedback on the difficulties of composite type maintenance from Tom Lane, one of the many contributing developers of the PostgreSQL database system.

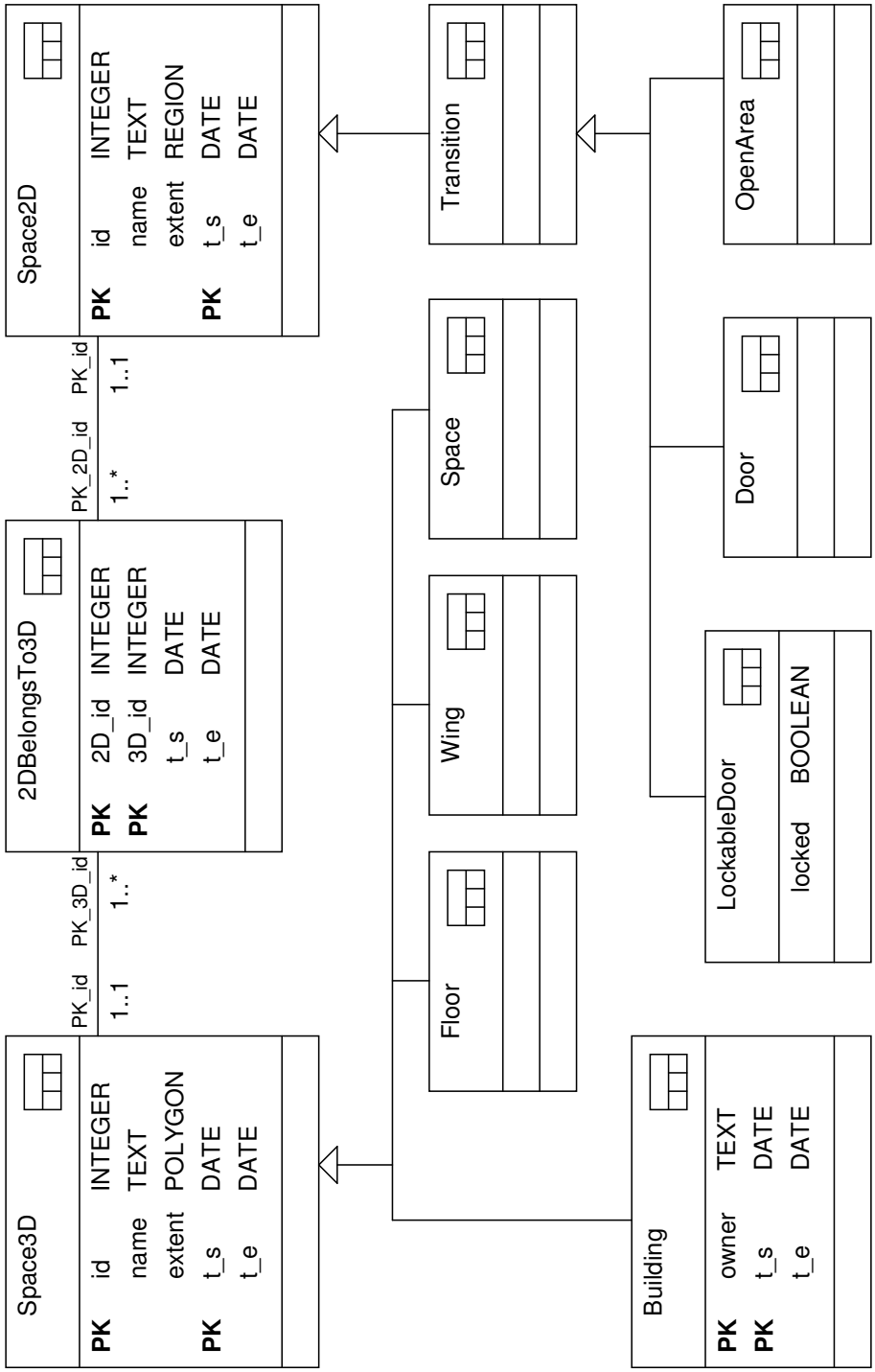

Figure 25: Database model of 2- and 3-dimensional spaces

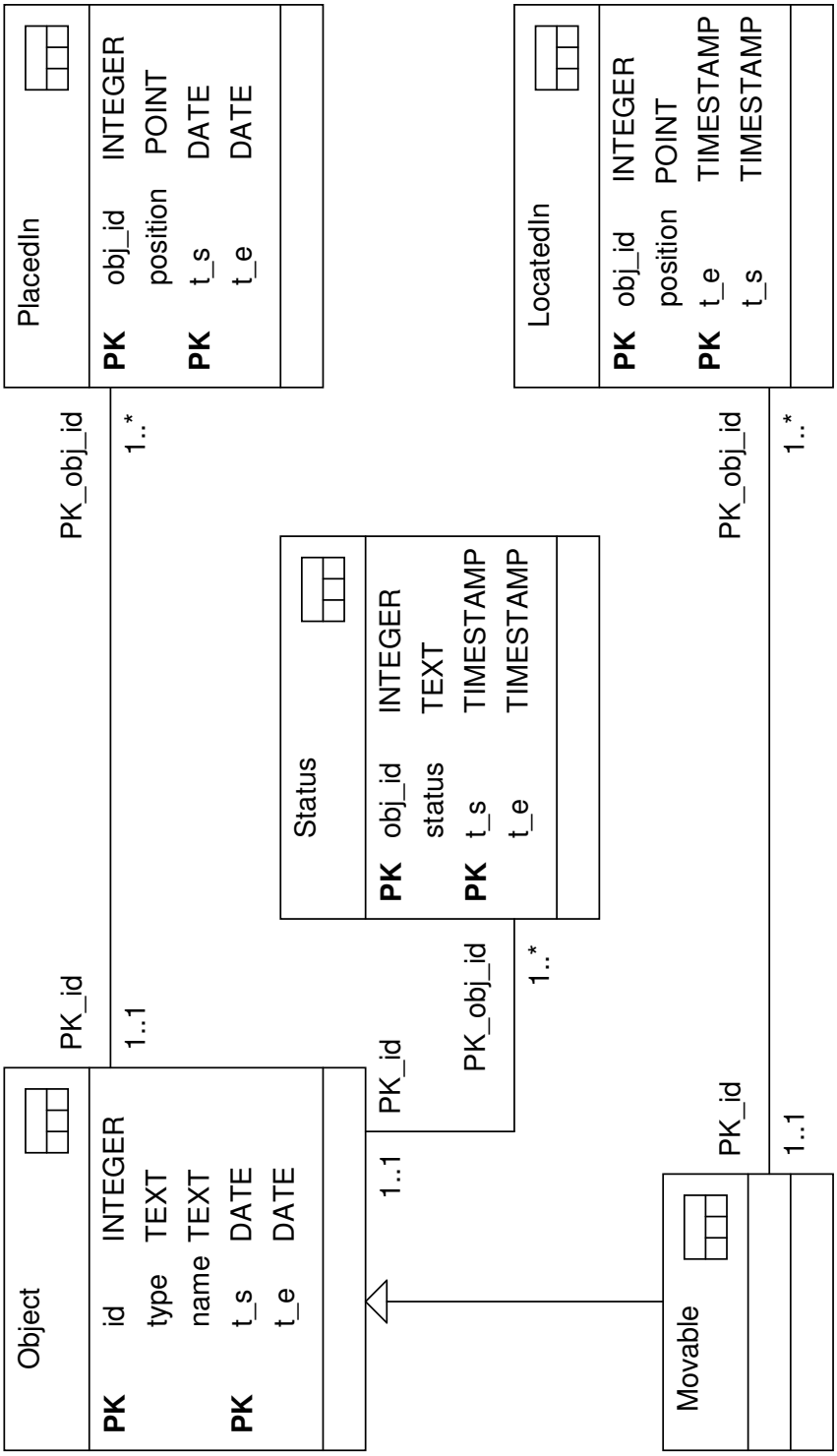

Figure 26: Database model of objects in space

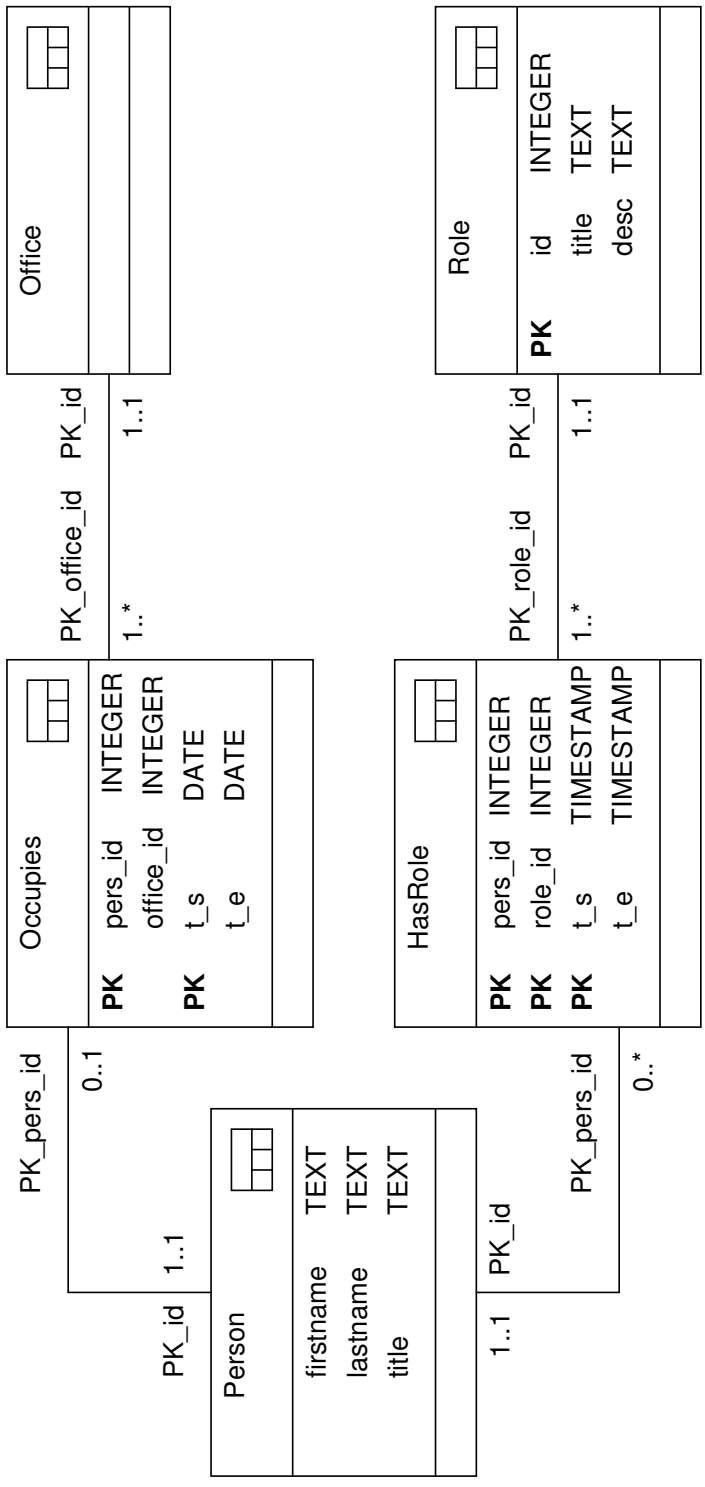

Figure 27: Database model of person occupying office and having roles

a 2-dimensional one belongs to. This raises the question: why then do we need the table "2DBelongsTo3D"? The answer is simple: we have already decided that a door or transition of any sort can belong to only one 3-dimensional space (chapter 4). Since transitions lie between two 3-dimensional spaces, using geometric information to determine which one they belong to will return two spaces. For example, a door between office "o02" and corridor "c01" belongs to office "o02". Using geometric information will return "o02" and "c01", indicating that the door belongs to both spaces.

Figure 26 shows how information regarding discretely and continuously moving objects is stored in the database ("Object" and "Movable", respectively). Since we are dealing with two different types of movement, we introduce a separate table for continuously moving objects. Positions with respect to time are thus given by the tables "LocatedIn" for continuous movement and "PlacedIn" for discrete movement.

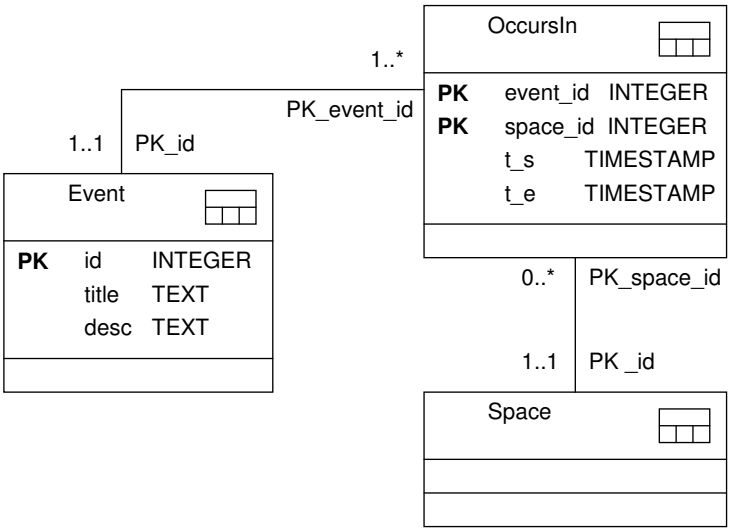

Figure 28: Database model of event occurring in space

The database model of persons occupying offices and having different roles is given in figure 27. The table "Person" has no identifier or any temporal attributes explicitly associated with its other attributes (firstname, lastname and title). Because it inherits the properties of "Object" via the class "Movable", it already has an identifier and the necessary temporal attributes.

The final database model, seen in figure 28, shows events occurring in spaces. Events are the only objects in our domain which are explicitly modeled with existence time. This is because we assume that events are planned in advanced and their start and end time are therefore known. The time and location of events is given in the table "OccursIn".

6.3 Other Implementation Issues

Now that we have created a database model based on a spatio-temporaly extended domain model, we are ready to implement it. There are however two major points that we have to consider before continuing. The first concerns the limitations of available extensions for spatial data management, while the second deals with the use of global versus local coordinates for positioning.

6.3.1 2- or 3-Dimensional?

In our models so far, we have considered locations as 3-dimensional spaces with the z-coordinate as the height of the location. Only transitions between these spaces have been regarded in 2-dimensions. Unfortunately, the spatial extensions available only support 2-dimensional space, meaning that they cannot be used to fully implement the database as modeled.

For indoor positioning systems, this problem can be solved by assuming that every space has the same height (i.e. the same z-coordinate) as the floor in which it is contained. Different floors can be explicitly numbered so that we know which floor we are on, and we can further assume that objects within spaces move in two dimensions only: x and y. This is a common assumption, since movable objects within buildings, such as persons, seldom move vertically (e.g. fly or crawl up walls). Based on these assumptions, it is not necessary to implement the table "Space3D". All locations and transitions are given as entries in the table "Space2D".

This particular solution can easily be integrated with the Ekahau Positioning System described in chapter 2.3.3, since Ekahau returns the position of a tracked object in the form of x, y and floor.

This also forces us to rethink our solution regarding the elevator. Since we no longer have a z-coordinate that can be used to determine the position of elevators, we must explicitly store this information using the numbering scheme of the floors. Elevators no longer have a spatio-temporal property showing their position relative to floors, but have a thematic-temporal relationship of the type integer with the class "Floor" that shows which floor the elevator is in at any given time.

Figure 29 shows the domain model with the necessary changes to cope with 2-dimensional instead of 3-dimensional geometric data. Only classes that require change are depicted. Other classes remain the same, as do their corresponding tables in the database model. The location of objects within spaces are no longer given as 3-dimensional coordinates, but as 2-dimensional coordinates and the floor number which the space is in.

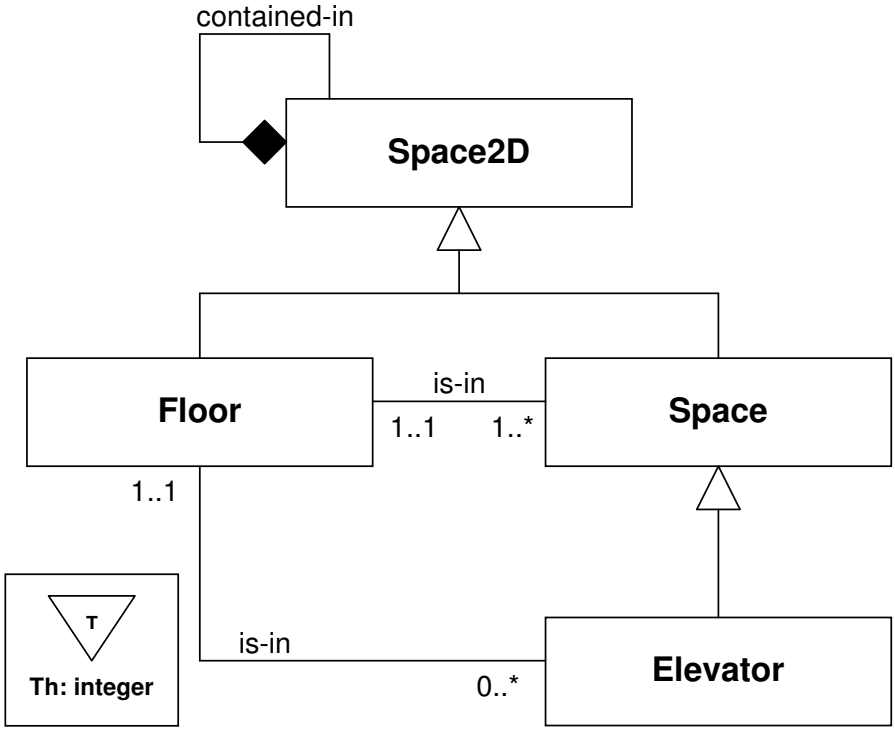

Figure 29: Domain model for 2-dimensional data

The corresponding database model is shown in figure 30. The attribute "floor_nr" is added to the table "LocatedIn". This is necessary in order to capture the

floor number in which the location data was registered. Two new tables are also added: “SpaceFloor” and “ElevatorFloor”. The table “SpaceFloor” might seem unnecessary, since a space can be thought of as existing on one floor only. However, this is not always the case. An example of this are different wings a building can consist of. Since wings overlap several floors, this particular table is needed to avoid redundancy in the database. Finally, “ElevatorFloor” contains data on the current and previous positions of the elevators.

6.3.2 Global or Local Coordinates?

The database model shows floors and wings defined as regions. It also shows rooms (corridors, offices etc) and open spaces defined as regions within floors and wings. Since each location in the model is defined as a region, a decision has to made about how to position objects within the building. Should local coordinates of each subspace be used, or global ones of the superspace? There are advantages and disadvantages to both approaches.

Local coordinate systems require additional information and therefore extra storage capacity in the database when storing data about positions. This is due to the fact that we now have to store the id of the location along with positioning information about objects. The id is used to determine in which location the position was registered. Without explicitly storing the id of a location with the coordinates, it is impossible to know in which location the coordinates were registered.

Global coordinate systems do not require the use of location ids when storing positioning data. They do however rely on a more extensive use of spatial operations to determine which location a specific set of coordinates belong to. This requires additional database and application resources.

In our implementation, we use global coordinates. Although the use of both approaches can be argued, we choose this particular approach because we wish to demonstrate the use of spatial operations.

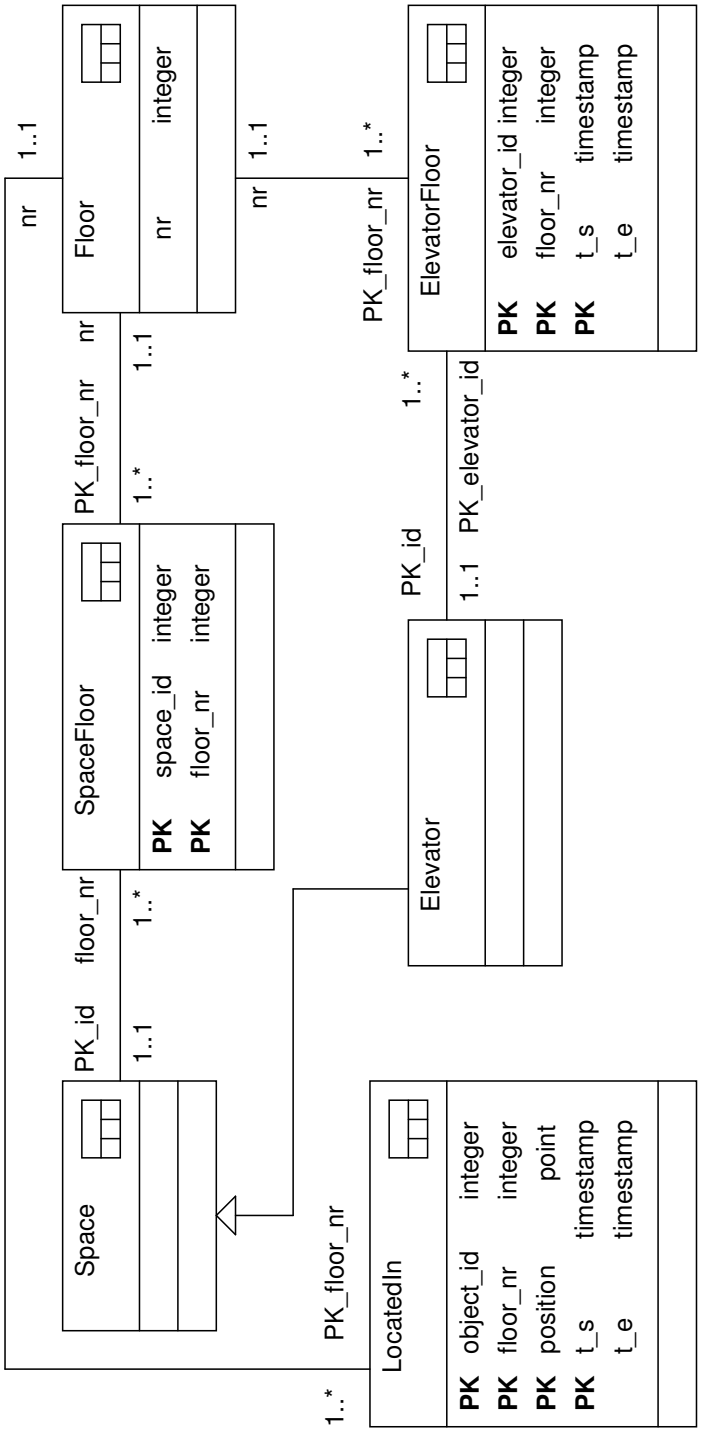

Figure 30: Changes to database model for 2-dimensional data

6.4 PostgreSQL Database and User Interface

Implementation of the database is done on the *PostgreSQL* [90] object-relational database system version 8.0.0beta. It includes the necessary spatial extensions based on the OpenGIS geometry model and is well suited for our use. In the following, we introduce and explain key parts of the code generated from the database model. We also discuss maintenance issues (i.e. updates) related to the database and present example queries, a few of which are executed through a user interface written in Java[21]. The complete source code for the database and user interface, along with necessary test data can be found on the CD in the appendix.

6.4.1 Tables

The following CREATE-statements demonstrate how some of the key tables used for storing information on geometrical locations and transitions are created. The main table is "Space2D", from which all other locations and transitions inherit their geometric properties. Since PostgreSQL done not have a general instatiable type "Geometry" (unlike MySQL, Oracle or DB2), we use the type "Polygon" to represent the geometric extent of locations and transitions.

```
CREATE TABLE Space2D (
   id INTEGER NOT NULL,
   name TEXT NOT NULL,
   extent POLYGON NOT NULL,
   t_s TIMESTAMP NOT NULL,
   t_e TIMESTAMP,
   PRIMARY KEY(id, t_s)
);
```

Examples of tables representing locations and transitions are given below ("Building" and "Transition", respectively). The tables seem odd, since they don't

[21]Not all the queries presented in this chapter are implemented in the application. The application is meant to be a simple demonstration of how data can be retrieved from the database using a graphical user interface (GUI).

have any columns of their own. They inherit necessary columns from the table "Space2D". The table "Owner" holds data on the different owners a building might have had over time.

```
CREATE TABLE Building (
   PRIMARY KEY(id)
) INHERITS (Space2D);

CREATE TABLE Transition (
   PRIMARY KEY(id)
) INHERITS (Space2D);

CREATE TABLE Owner (
   building_id INTEGER NOT NULL,
   owner TEXT NOT NULL,
   t_s DATE NOT NULL,
   t_e DATE DEFAULT NULL,
   PRIMARY KEY(building_id, owner, t_s),
   FOREIGN KEY(building_id) REFERENCES Building(id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

The table "Person" inherits the properties of "MovableObject", which includes an identifier and status. As with all movable objects, a persons position is stored in the table "LocatedIn". As we see from the table declaration, it is necessary to explicitly store the floor number of a person's location. This is due to the fact that the geometric point describing the location is in two dimensions (see chapter 6.3.1 for details).

```
CREATE TABLE Person (
   firstname TEXT NOT NULL,
   lastname TEXT NOT NULL,
   title TEXT NOT NULL,
```

```
    t_s DATE NOT NULL,
    t_e DATE DEFAULT NULL,
    PRIMARY KEY(id)
) INHERITS (MovableObject);

CREATE TABLE LocatedIn (
    id INTEGER NOT NULL,
    floor_nr INTEGER NOT NULL,
    position POINT NOT NULL,
    t_s TIMESTAMP NOT NULL,
    t_e TIMESTAMP DEFAULT NULL,
    PRIMARY KEY(id, t_s),
    FOREIGN KEY(id) REFERENCES Person(id)
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

Roles that persons might have at different times can be found in the tables "HasRole" and "Role". Using the data in these tables we can find out who was where at what time and with what role.

```
CREATE TABLE Role (
    id INTEGER PRIMARY KEY,
    name TEXT NOT NULL,
    tasks TEXT NOT NULL
);

CREATE TABLE HasRole (
    person_id INTEGER NOT NULL,
    role_id INTEGER NOT NULL,
    t_s TIMESTAMP NOT NULL,
    t_e TIMESTAMP DEFAULT NULL,
    PRIMARY KEY(person_id, role_id),
    FOREIGN KEY(person_id) REFERENCES Person(id)
```

ON DELETE CASCADE ON UPDATE CASCADE,
FOREiGN KEY(role_id) **REFERENCES** Role(id)
ON DELETE CASCADE ON UPDATE CASCADE
);

6.4.2 Functions

Two examples of user-defined functions are presented below. These are implementations of operations which we deemed as relevant for indoor location based services, presented in chapter 5.6.3.

The spatial function "within" returns a boolean value (true or false) on whether or not a point is within a polygon (e.g. whether or not a person is within a room).

CREATE FUNCTION within(**POINT**, **POLYGON**)
RETURNS BOOLEAN AS
'**SELECT** $1 @ $2;'
LANGUAGE SQL;

The temporal function "overlap" return a boolean value indicating whether or not two timeframes overlap. The start and end times of the two timeframes to be checked for overlap are given as timestamps.

CREATE FUNCTION overlap(**TIMESTAMP**, **TIMESTAMP**, **TIMESTAMP**, **TIMESTAMP**)
RETURNS BOOLEAN AS
'**SELECT** ($1 < $4 **AND** $2 > $3) **OR** ($1 < $4 **AND** $2 **IS NULL**);'
LANGUAGE SQL;

6.4.3 Updates

A location aware application periodically updates the database with the positions of continuously changing[22] objects (i.e. objects with bitemporal time).

[22]Remember: moving is still regarded as a form of change.

Positions of discretely changing objects (i.e. objects with transactional time) are updated only when they actually change. These operations are related to time and are perhaps the biggest issue concerning maintenance of the database, that is: *how do spatio-temporal databases cope with changes in their location models*? Several different changes may occur:

- Spaces and transitions (e.g. rooms, offices, floors, wings, doors) may be added or removed.
- Spaces and transitions may change their symbolic names and geometrical extent.
- Spaces (e.g. buildings and offices) may change owner or occupant.
- Objects (movable and non-movable) and events may be added or removed.

New spaces, transitions or objects must be added manually by the application administrator or someone with the proper privileges. The same is true for when elements change their symbolic name, geometric extent, owner or occupant. The location aware application reflects these changes in the database by updating necessary relations. To retain temporal and spatio-temporal validity, triggers are used to update relevant temporal properties. The following example of a space changing its symbolic name illustrates this.

The location aware application is told that the office known as "o02" is to be renamed "rc02". Since we wish to track the history of all objects in our database, we cannot simply update the name attribute and discard its previous value. The previous value is kept, and the t_e attribute corresponding to it is set to the time of the change. The t_s of the new name and t_e of the previous name are now the same (current_date), indicating that the space in question has a new symbolic name from the given time. The application thus changes the symbolic name in the location model, updates the temporal validity of the previous name in the database, and inserts the new name:

UPDATE Space2D **SET** t_e = current_date **WHERE** id = office_id;
INSERT INTO Space2D **VALUES**(office_id, 'rc02', extent, current_date);

Here, the attribute *current_date* is given by the database system, while the attributes *office_id* and *extent* are provided by the application. As we see from the example, both the update and insert must be performed explicitly. This is not necessary using temporal query languages. In such cases, we merely update the database with the new name, and the proper temporal values and inserts are added implicitly [62]. But as mentioned, there are no temporal database systems available with provided support for spatial data. We must therefore perform all updates and inserts explicitly.

As another example, imagine if office "o02" is to be divided into two new offices "o02a" and "o02b", each with its own geometric extent. This means that the original office ceases to exist, and two new ones must be added. Although "o02" no longer is a part of the location model, it must still be represented in the database. This is again due to the fact that we wish to keep historical data for future querying. We therefore set the t_e attribute of the office in relation "Space2D" to the time of its termination, indicating that the name and extent no longer are valid. We then insert the two new offices in the location model and database:

UPDATE Space2D **SET** t_e = current_date
  WHERE name = "o02" **AND** t_e **IS NULL**;

INSERT INTO Space2D **VALUES**
    (office_id_a, 'o02a', extent_a, current_date);
INSERT INTO Space2D **VALUES**
    (office_id_b, 'o02b', extent_b, current_date);

The same approach is taken for all objects that update temporal attributes. We begin by updating the temporal attribute of the outdated version, before inserting a new, temporaly valid object. Using this approach, we keep track of object history in the database.

6.4.4 Queries

This sections contains example queries that demonstrate the functionality of the database. The queries are written based on the scenarios described in chapter

5.2. They are meant to demonstrate the practical use of such databases. The queries consider the status and roles of different objects/persons as well as their positions. All the queries have been tested to ensure that they compile, run and return the proper result, based on test data for which we already know what results they will return.

The section also contains screenshots of a simple application used to retrieve information from the database. The application is written in Java and allows users to query the database about present and historical information. In cases where execution of a query requires certain parameters, users can enter or change these via the interface. We wish to emphasize that this application in *not* a location aware system. It is merely a demonstration of how a graphical interface towards the database can be created.

Query 1: Where am I (assuming that "current_position" and "floor_nr" are given by the location aware application)?

SELECT S.name
FROM FROM Space2D S, SpaceFloor SF
WHERE S.t_e **IS NULL**
  AND S.id = SF.space_id **AND** SF.floor_nr = floor_nr
  AND within(current_position, S.extent);

This is a fairly simple query. It is purely spatial and requires the use of one spatial operation: **within**. Since a space may have changed its geometry and names over time, we select the latest by checking that the temporal attribute t_e is null.

Query 2: Where is "Behreng Mirzaei"?

SELECT SN.name
FROM Space2D S, Person P, LocatedIn L, SpaceFloor SF
WHERE P.firstname = 'Behreng' **AND** P.lastname = 'Mirzaei'
  AND P.t_e **IS NULL** P.id = L.id
  AND L.t_e **IS NULL AND** L.floor_nr = SF.floor_nr

AND SF.space_id = S.space_id **AND** S.t_e **IS NULL**
AND within(L.position, S.extent);

This query retrieves the latest position of "Behreng Mirzaei" by checking the t_e attribute for null in the relation "LocatedIn". The tuple that satisfies this criteria is used with the spatial operation **within** to determine where this position is. The geometry and name of the location are selected as in query 1, that is, by selecting the latest entry. Figure 31 shows the query executed by the application.

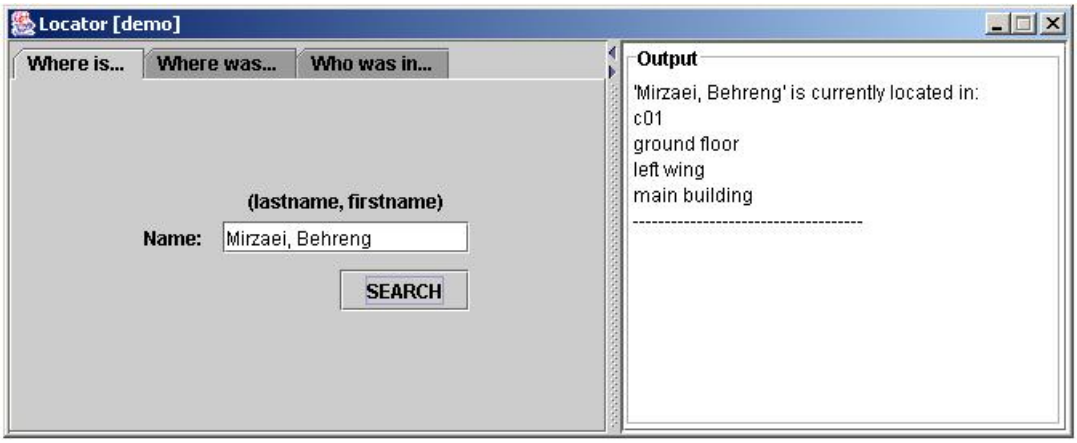

Figure 31: Searching for the current position of a particular person

The two first queries are usually not executed in the database. The queries ask for information regarding the present position of objects, which the location aware application is capable of answering. We have included these queries to demonstrate that the database can answer queries regarding the present. It is in cases where we are interested in the past that the information in the database becomes relevant. The following queries illustrate this.

Query 3: Who was in office "o03" on November 20. 2004?

SELECT DISTINCT ON (P.firstname, P.lastname)
P.firstname||' '||P.lastname **AS** was_in_o03
FROM Person P, LocatedIn L, Space2D S, Office O, SpaceFloor SF
WHERE P.t_e **IS NULL AND** P.id = L.id

AND overlap(L.t_s, L.t_e, '2004-11-19 23:59', '2004-11-20 23:59')
AND L.floor_nr = SF.floor_nr **AND** SF.space_id = S.id
AND S.t_e **IS NULL AND** S.name = 'o03'
AND within(L.position, S.extent);

In this query, the positions of all persons for the date in question are retrieved by first using the temporal operation **overlap** to ensure the selection of values within the given timeframe and then the spatial operation **within** to determine which one of these were inside office "o03". We also use the standard SQL operation **DISTINCT** because a person might have been inside office "o03" on several different occasions during the given date. A screenshot of the application executing this query is given in figure 32.

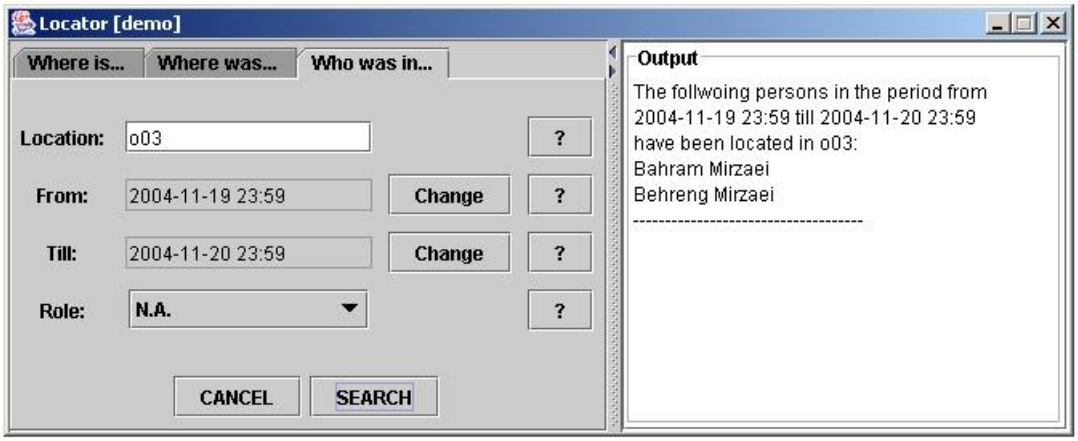

Figure 32: Searching for names of persons in a room at a given time

Query 4: What were the locations of employees who were project managers between November 20. 2004 and November 24. 2004?

SELECT DISTINCT ON (P.firstname, P.lastname, S.name)
P.firstname||' '||P.lastname **AS** name, S.name **AS** location
FROM Space2D S, Person P, LocatedIn L, HasRole HR, Role R
WHERE P.t_e **IS NULL AND** P.id = HR.person_id
AND HR.role_id = R.id **AND** R.name = 'project manager'
AND overlap(HR.t_s, HR.t_e, '2004-11-19 23:59', '2004-11-24 23:59')

AND overlap(L.t_s, L.t_e, '2004-11-19 23:59', '2004-11-24 23:59')
AND P.id = L.id **AND within**(L.position, S.extent)
ORDER BY P.firstname, P.lastname;

Here, we make sure that only values which lie within the desired timeframe are selected by checking the temporal attributes of positions and roles. We do this by using the temporal operation **overlap** twice: once to determine the names of those who had the role of "project manager" during the desired timeframe (including those who received the role after November 20. 2004 and kept it longer than November 24. 2004), and again to determine positions within the same timeframe. Locations of the selected project managers are determined by the spatial operation **within**. The clause **ORDER BY** orders the result of the query so that it becomes easier to read. Figure 33 shows a screenshot of the application executing this query.

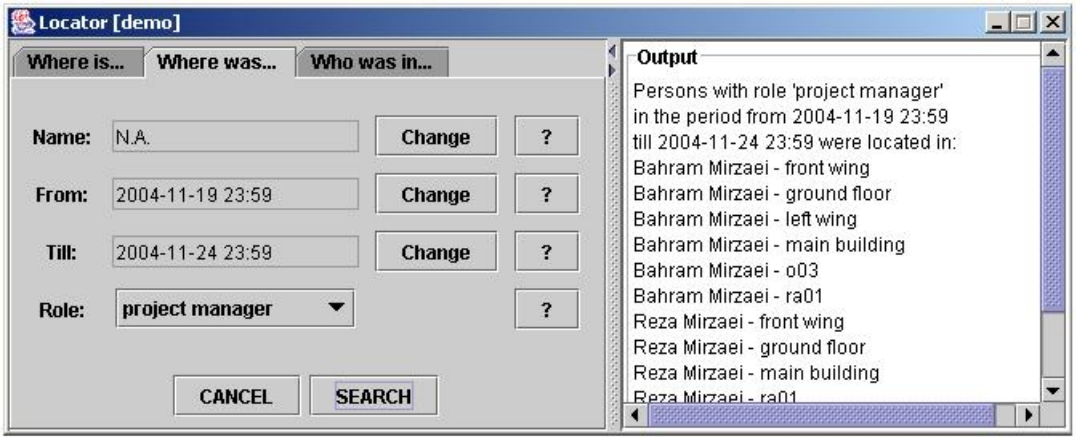

Figure 33: Searching for previous positions of persons related to their roles

Query 5: Did any project co-workers attend staff meetings on November 20. 2004?

SELECT DISTINCT ON (P.firstname, P.lastname)
P.firstname||' '||P.lastname **AS** attended_staff_meeting
FROM Person P, HasRole HR, Role R, LocatedIn L, Event E,
OccursIn O, Space2D S

WHERE P.t_e **IS NULL AND** P.id = HR.person_id
AND HR.role_id = R.id **AND** R.name = 'project co-worker'
AND overlap(HR.t_s, HR.t_e, '2004-11-19 23:59', '2004-11-20 23:59')
AND L.id = P.id **AND** E.title = 'staff meeting' AND E.id = O.event_id
AND overlap(O.t_s, O.t_e, '2004-11-19 23:59', '2004-11-20 23:59')
AND O.space_id = S.id **AND within**(L.position, S.extent);

Once again we use the temporal operation **overlap** twice: first to select names of employees who were project co-workers during the desired timeframe, then to select identifiers for locations in which staff meetings occurred within the same timeframe. Finally, we use the spatial operation **within** to determine which one of the project co-workers were in locations where staff meetings were held.

Query 5: Which project managers met on November 20. 2004, and where did they meet?

This is the most complicated query we have tested so far, because it requires us to make a few assumptions about what we deem as "meeting". Needles to say, the database possesses no artificial intelligence in order to "understand" what it means for two or more individuals to meet. We have to somehow define a meeting with conditions that have to be true for the database to recognize such an event. We propose the following conditions: First of all, for a meeting to take place, the involved individuals must be on the same floor and in the same location at the same time. Secondly, the duration of this time must be longer than a given minimum, and finally, the distance between the individuals must be less than a given maximum. Simply being in the same room at the same time does not qualify as a meeting, since people can be in the same corridor, room or open area without exchanging a single word.

Since this query requires a great number of clauses, we simplify it by first creating a view which contains the identifiers of individuals who were project managers during the desired timeframe and had a distance of less than the given maximum to each other at the same time. The view also contains the start and end times for when the distance was less than the given maximum. We use the view to select the names of those who were close enough to each other with a

duration longer than the given minimum, along with the name of the location they were in.

CREATE VIEW Temp **AS**
SELECT P1.id **AS** P1_id, P2.id **AS** P2_id,
  MIN(L1.t_s) **AS** L1_min, **MAX**(L1.t_e) **AS** L1_max,
  MIN(L2.t_s) **AS** L2_min, **MAX**(L2.t_e) **AS** L2_max,
FROM Person P1, Person P2, HasRole H1, HasRole H2,
  Role R, LocatedIn L1, LocatedIn L2
WHERE P1.id <> P2.id
  AND P1.t_e **IS NULL AND** P2.t_e **IS NULL**
  AND overlap(H1.t_s, H1.t_e, '2004-11-19 23:59', '2004-11-20 23:59')
  AND overlap(H2.t_s, H2.t_e, '2004-11-19 23:59', '2004-11-20 23:59')
  AND P1.id = H1.person_id **AND** P2.id = H2.person_id
  AND H1.role_id = R.id **AND** H2.role_id = R.id
  AND R.name = 'project manager' **AND** P1.id = L1.id AND P2.id = L2.id
  AND L1.floor_nr = L2.floor_nr
  AND overlap(L1.t_s, L1.t_e, L2.t_s, L2.t_e)
  AND distance(L1.position, L2.position) < given_maximum
GROUP BY P1.id, P2.id;

SELECT DISTINCT ON (P1.firstname, P1.lastname)
  P1.firstname||' '||P1.lastname||' met '||P2.firstname||' '||P2.lastname
  AS met_on_nov_20_2004, S.name **AS** met_in
FROM Temp T, Person P1, Person P2, LocatedIn L, Space S
WHERE P1.id <> P2.id
  AND P1.id = T.P1_id **AND** P2.id = T.P2_id
  AND duration((**SELECT MIN**(L1_min) **FROM** Temp),
      (**SELECT MAX**(L1_max) **FROM** Temp)) > given_minimum
  AND duration((**SELECT MIN**(L2_min) **FROM** Temp),
      (**SELECT MAX**(L2_max) **FROM** Temp)) > given_minimum
  AND within(L.position, S.extent);

There is of course some degree of uncertainty regarding the assumptions we have made. Two people might for example be standing next to each other several minutes without interacting in any way. The query therefore really answers whether or not individuals *could* have met. This is the best result we can achieve without introducing some form of artificial intelligence or explicitly storing information on random meetings.

6.5 Research Question 3

Our third and final research question was (chapter 1.5): *Based on spatial, temporal and spatio-temporal data, how can we construct spatio-temporal databases that are able to reflect changes in their location models*?

We have answered this question in this chapter. We began by creating a database model based on the conceptual model that was used as a basis for the location model. This ensured that entities, events and relationships which are important in the location model are also present in the database model. Changes in the location model, e.g. new spaces, transitions, objects and events being added, or spaces changing extent or name can be handled by the database in a manner that does not conflict with the integrity of the location model.

7 Reflections and Evaluation

This chapter is dedicated to evaluation of, and reflection over the scientific work conducted in this thesis. We begin by presenting how the work was conducted in practice, including mistakes, misunderstandings and lessons learned. We continue by presenting how our selected research approach was applied to the topics. This is followed by evaluation of our research, including a discussion on the validity of our proposed answers to the research questions (see chapter 1). We then discuss how our research approach might have affected the results, and whether or not other approaches might have yielded different ones.

7.1 Scientific Work

Conducting the scientific work for this thesis was indeed a cumbersome task. Although scientific papers discussing the issues of location modeling and spatio-temporal databases were easy to find, examples and studies of implementations were less common. Because of the lack of implementation examples and guidance, problems and misunderstandings occurred during our work which in several occasions lead to major changes.

7.1.1 Problems, Misunderstandings and Changes

During the course of our work, problems and misunderstandings that arose were discussed with our supervisors, and their recommendations were followed. These recommendations often resulted in changes to specific issues in modeling and implementation. A few recommendations however, caused redefinition of the research questions. We now present these aspects in detail.

Location Modeling

We decided at an early stage to use a hybrid approach for location modeling. The decision was made based upon the fact that we wanted to represent symbolic as well as geometric information in our location model. Our first location model was created using a *tree* structure. However, we soon discovered that a tree could

not model a building according to all possible relationships between its different constructs. Because of this, we redefined our location model as a lattice. We also defined the principles of the lattice using mathematical set theory, but we soon discovered that our mathematical discussion was to complicated and indeed unnecessary, since it only validated what mathematicians already have. We therefore decided to not include the chapter on mathematical validation.

Creating the location model itself also posed a challenge. We began by creating a hybrid location model using the proposed syntax by Dürr and Rothermel [16]. This syntax however could not model locations well enough for range-queries. It also lead to extremely complicated models when the building to be modeled consisted of many floors, wings and rooms. To cope with these shortcoming, we proposed our own syntax and created a location model based on it.

Spatial and Spatio-Temporal Databases

To begin with, we focused on purely spatial databases. We started by presenting work conducted in that area, including a lengthy discussion on the underlying geometrical basis that is required in order to implement geometrical data models in computer systems. The reason for this was that computer number systems cannot cope with the infinite properties of Euclidean geometry and are thus obsolete for coordinate based geometry [18]. Although this is to some extent relevant in our work, it does not have any direct influence on how we model and construct databases for geometrical data. The discussion was therefore removed from the thesis.

The second change regarded the incorporation of temporal aspects to geometrical data. After a few attempts to model changing geometrical object based on purely spatial data types, we discovered that time is an important factor and must be dealt with in some manner. This is because the position and extent of changing geometrical objects are time dependent, i.e. they change with respect to time. We therefore decided to use spatio-temporal database systems instead of just spatial ones. Unfortunately, this decision was made late in the process, which meant that large sections of chapter 5 had to be rewritten.

Research Questions

Because of the mentioned changes to the work conducted, we were forced to redefine the second and third research questions (chapter 1.5). The changes were made so that the questions now focused on spatio-temporal data management, a opposed to the just spatial. The first research question, regarding location modeling, was not redefined during the work despite changes to the models.

Implementation

Once we had created a location model and modeled a spatio-temporal database, we attempted to implement it for testing. Our first implementation efforts were based on an object-relational approach. This proved to be more difficult than anticipated, because the available spatial extensions did not have all the features required. Because of this, we had to define several functions and data types ourselves. During this part of our work we were fortunate enough to receive help from several members of the PostgreSQL community through e-mail correspondence.

Besides function and data type implementation issues, there was one serious problem that had to be dealt with: none of the currently available database system supported 3-dimensional geometric data. This meant in effect that it was impossible to implement our database model based on existing standards. We were however able to solve this problem by explicitly numbering floors and assuming that objects only move in two spatial dimensions (longitude and latitude, not altitude).

When time came to demonstrate the practical use of the database, it became obvious that issues related to events occurring in spaces and persons having different roles had been neglected. The concept of persons having roles was not even modeled in the first versions of the domain or database models, which meant that these had to be redone. This was done after discussions with our supervisors and necessary tables based on the new models were added to the database. Furthermore, additional queries were written to test both issues.

Finally, the implemented database has yet to be tested properly with an indoor positioning system. This was the initial intention, but by the time the

implementation was completed, a positioning system was not yet installed. The database has been manually filled with data and tested, but its true potential can only be discovered once it's properly run with a location aware system.

7.1.2 Applied Research Approach

The scientific work in this thesis was conducted following the steps defined in design research (chapter 3.2, figure 2). The steps were applied to two different topics: location and spatio-temporal database modeling for indoor location aware services.

In the case of location modeling, we began by identifying the problem as the lack of location modeling techniques that are suited for nearest-neighbor as well as range queries (step 1). We then proposed a solution to the problem based on an extended hybrid location model (step 2) and used our own proposal to develop a new modeling technique (step 3).

The same steps were applied to the next topic. We identified the problem as a lack of constructs for conceptual modeling of spatio-temporal databases for indoor positioning systems (step 1). As a solution to this problem (step 2), we proposed an extended version of an already existing modeling language (UML). We followed our proposal and created a conceptual model that defines objects, events, actions and relationships that occur in indoor environments (step 3).

The steps were followed with multiple iterations in both cases. We had to repeat steps 1 through 3 several times due to the problems and misunderstandings mentioned previously.

The above discussion only covers the three initial steps of design research. The question that has yet to be answered is: What about steps 4 and 5, i.e. evaluation and conclusion? We answer this question in the next section.

7.2 Evaluation of Research

In order to evaluate our research, we must take several different issues into consideration. First of all, we must evaluate the data gathered and determine

how it was analyzed and used. We must also evaluate the results achieved to prove their validity. The entire process of evaluation must be performed with the selected research approach in mind. This is a very important point to remember, since the approach itself could in many ways have (and most likely has) affected the research.

7.2.1 Evaluation of Data

The data gathered and used in this thesis is based on observation of three different concepts: (1) The architectural design of buildings to determine a set of general constructs which can be used to describe how buildings are constructed and the relationship between these constructs (location modeling), (2) events, actions and changes associated with these constructs, including actions occurring in them, such as objects moving and events taking place (location-awareness), and (3) spatial, temporal and spatio-temporal properties of (1) and (2).

Evaluating this means asking whether or not the data is relevant and valid according to the topics we wish to research. The answer to this question is closely associated with the type of result we aim at producing, namely conceptual models. This in turn, is associated with the selected research approach, demonstrating how it directly affects the research. Since our chosen approach presumed conceptual models, we knew from the beginning that we needed to gather data which could be used to create such models:

(1) The first concept, regarding location modeling, produced a set of data containing different classes and associations used to conceptually model buildings. Since the basis for the model is object-oriented design, it can be extended to include other classes which other developers might yield as important. The data was in part validated by reference to previous work done in this field, and part by creating a location model of a given building. The location model was tested for validity by performing general queries to check if it is capable of answering these. Data was recollected and the model redefined until it could answer the necessary queries.

(2) The second concept, concerning location-awareness, was also based in parts on previous work in the field of location aware systems. We gathered the most general concepts and incorporated these into our data set. This included concepts such as events and status of objects. As a further concept, we extended the data set to include different roles persons might have. We deemed this as an important construct, since the location of persons is often associated with the roles they have.

(3) Once the fundamental data had been gathered, we added spatial, temporal and spatio-temporal properties to it. We did this by observing particular properties of the different types of data in order to determine specific spatial, temporal or spatio-temporal aspects. We found for example that all buildings have a geometric extent of some sort and that this can change over time, thus making it spatio-temporal. The conceptual modeling technique used to illustrate these aspects was found in previous research.

The data acquired in (2) and (3) were validated by creating a prototype database based on a conceptual model derived form them. The database was queried on issues we deemed as significant for indoor location aware applications. As with (1), the data in (2) and (3) was recollected and models based on them redefined until the database could answer the queries satisfactory.

7.2.2 Validation of Results

According to Tichy [61], validation of research results can (and should be) be done by experiments. But, as he argues further, experiments used to validate scientific work in computer science are less common relative to other sciences. The reasons for this are several fallacies that dominate scientific work in this field. These are, among other things, based on the wrongful impressions that experiments cost to much, slow the progress and that enough experiments are already being conducted. Another fallacy, which is probably the reason why researchers in computer science conduct few proper experiments, is:

Demonstrations will suffice.

[61], p. 35

In our work, we present examples of usage, where we simply demonstrate the modeling of a simple scenarios using our proposals for location and database modeling. Although we also argue how the same models would be difficult to construct without our proposed extensions, the demonstrations merely illustrate the potential of the result, they do not validate it.

A proper experiment based on different scenarios, different data types and actual implementations into functional applications would be helpful in validating or rejecting the result. Such an experiment would show if the proposed modeling technique is applicable to all scenarios containing geometrical objects that change. It would also show if models based on this technique can be mapped into useful applications, which essentially is the goal of the research.

Our research relies to a large extent on previous work based on constructive and design research, or as referred to by March and Smith, *design science*. We thus continue the work in the same tradition by constructing a model "that serves human purpose" [47] p. 253. However, since we do not properly evaluate our work by conducting experiments, we cannot say that we fully follow the steps of design research[23]. Steps 4 (measurement and evaluation of the results) and 5 (conclusion based on the 4. step) are not dealt with in a satisfactory manner. It is therefore difficult to conclude that the result is valid for all scenarios regarding changing geometrical objects. The fact that only one scenario is used to demonstrate the use of the models also makes it hard to evaluate the generalizability of our results.

Although this discussion implies that it is difficult to evaluate the research approach since we cannot tell whether it has contributed to a valid result or

[23]Design science consists of similar steps, but as pointed out by March and Smith [47] p. 254, the basic activities are *build* and *evaluate*. In this thesis, the evaluation step is poorly covered.

not, it is clear that the selected approach has yielded results of the expected (and wanted) type: conceptual models.

7.3 Alternative Approaches

It is quite clear that we use a positivistic constructive research approach in this thesis (chapter 3.2). We chose this approach because we knew from the beginning that the intended solution would be conceptual models based on the observation of geometrical constructs. There are however other research approaches which might have lead to different results. What would the outcome have been have we adopted a different point of view?

An Interpretive Approach

We believe that an interpretive approach to the research questions would include human perception of changing geometrical objects. Although research data is collected empirically, the general human perception of it is not mapped. It might be argued that the data is influenced by how we ourselves perceive it. We have however been as humanly objective as possible, meaning we have attempted to collect data on the basic elements which buildings consist of along with events and objects inside those building without distortion of our personal feelings or interpretation.

We also believe that an interpretive approach would consider the practical use of spatio-temporal applications more closely. In our work, we use an office management and a hospital monitoring system as examples, but there is no discussion on the general practicality of spatio-temporal applications. According to us, an interpretive stance would seek to discover further scenarios where people find such applications useful.

All this is not to say that an interpretive point of view could not yield conceptual models as result, it might just as well do so. It is however highly probable that models based on this approach would turn out dissimilar to the ones we have proposed. The reason for this is of course human awareness of its surroundings.

Since humans experience events differently, their descriptions of them are also different. Thus, human perception of how geometrical objects change might not

be the same as capabilities and actions such as *intersection*, *containment*, and *trajectory*, deemed significant by us. The same could be true for the temporal aspects presented in this paper. Humans might value other temporal qualities than valid and transaction time, or use other terms to describe these phenomena.

Collecting such data would require tedious work, based on surveys and interviews to uncover what the research objects mean to humans, i.e. which capabilities humans view as important. The data would then have to be sorted, interpreted, and a model based on it created. An interpretive stance could reveal other spatio-temporal constructs which would have to be embedded into the model and the result, although it could still be designed as a conceptual model, would be different.

Quantitative Data Acquisition

It is obvious that the research conducted in this thesis uses no quantitative means what so ever. All data is gathered and presented in a qualitative manner, describing the properties and relationships of geometrical objects and using graphical symbols to depict these.

As an alternative to this, we could have used quantifiable data to create mathematical models describing the research objects. However, such an approach would not fulfill the purpose of the thesis.

Mathematically based artifacts, such as mathematical models are *not* conceptual models. In informatics, mathematical models are used to analyze, test, simulate or control systems, i.e. they are used to "explain behavior and improve performance" [47], p. 259. Conceptual models on the other hand, are used to describe a system using symbols and descriptive language before it is implemented. Such semantic models are easier to understand since they "better correspond to the end user's conceptualization" [47] p. 259. Based on this, we found that a quantitative approach would indeed not have answered our research questions, since they aimed at creating conceptual models.

The quantitative approach is not restricted to mathematical models. We are of the opinion that it could have been used together with surveys and questionnaires to find the dominant capabilities that humans associate with moving

geometrical objects. The most prominent capabilities could then have been used in the development of our conceptual models. This could lead to potentially different models than the ones presented by us.

Deeper Insight or General Description?

We have used the constructive approach because our purpose is to create conceptuals model representing locations and objects with spatial and temporal properties within those locations. We assume that other approaches could have been used to either give deeper insight into the research topic, or to provide general descriptions of it.

Deeper insight could have been given using the ideographic approach [14]. We presume that the we then would have had to observe a geometrical object in its natural environment over some time to describe its behavior. This could for example be the observation of a typhoon from the time it is created until is disappears, or the development of the population density in a geographic area. However, the description of these phenomena would not be useful in creating a conceptual model for describing spatio-temporal constructs. Deeper insight is not what we seek to give. We therefore deem an ideographic approach as unsuitable for answering the research questions.

In order to give a general description of the research topic, we could have used the nomothetic approach. This would mean an empirical study of the research topic with the idea to provide general laws/theories. Although it can be said that results based on this approach lie within the scope we set, i.e. to give a general description of location aware applications, the outcome would most likely not be a conceptual model. We thus believe that also the nomothetic approach is unsuitable for answering the research questions.

7.4 Final Reflections

During our work, we had to perform several unexpected and at times large changes in order to cope with problems and misunderstandings. This was an eye opener for us, particularly since we expected to conduct the scientific work according to the initial research questions and expectations. We were prepared

for some drawbacks, and experienced enough to know that problems will occur and that changes must be done to overcome these, but the extent in which they occurred was at times very discouraging. However, in the aftermath of our "struggle", we realize that it was indeed the problems and misunderstandings that were the real source of scientific research. It was through them that we managed to improve our work and learn. The experiences gained and lessons learned from rewriting, remodeling, redesigning and reimplementing have left us with a vast array of knowledge regarding modeling and construction of spatio-temporal databases for indoor location aware systems.

The theoretical discussion on research methodology was also a great learning experience. It opened a new world of research approaches and methods to us, which we were unfamiliar with until then. It also showed us how our selected approach in part affected our work, and how other approaches would have attempted to solve our research questions. We learned that although several of the other approaches could have resulted in conceptual models, it is very unlikely that these would be the same as the ones presented in this thesis.

8 Conclusion

We have discussed two related topics in this thesis: the design of location models and spatio-temporal databases for indoor positioning systems. We conclude our work with a short recapitulation of the main issues presented and a justification of the limitations, followed by suggestions for further research and our final thoughts.

8.1 Location Modeling

Different aspects of location modeling were discussed, among them symbolic, geometric and hybrid location models. We found that only the hybrid form is suited for our purposes, that is to accurately position objects within a domain and relay this information to users in an understandable way.

As a modeling technique, we proposed a solution using basic constructs of UML with added extensions. We created a domain model containing objects and relationships between them which we deem as relevant for indoor location aware systems, including the geometric extent of locations and transitions.

8.2 Spatio-Temporal Databases

We defined spatio-temporal databases as databases which manage data on geometrical objects that change (movement is also considered as a form of change). Although research in this area has received much attention in the past decade, no functional spatio-temporal systems are available. Purely spatial database systems however are. We therefore proposed a solution based on timestamping spatial data, thus capturing its temporal aspects and in effect making it spatio-temporal.

In order to create a conceptual model of such a database for indoor location aware applications, we used our own domain model for location modeling and proposed extensions to it in order to capture specific spatial, temporal and spatio-temporal aspects. We used the conceptual model to create a prototype database, demonstrating the concepts proposed by us.

8.3 Limitations

The limitations of our work were presented in chapter 1.6. These included *querying mechanisms*, *indexing techniques*, *storage capacity/management*, *positioning technology* and *legal and ethical issues*. These were deemed as beyond our scope of research because our main objective was based on a *conceptual* approach. Our results were thus meant to be valid regardless of these areas. Although these limitations exclude important aspects of location aware systems from our work, they have no direct influence on our results. The models we have proposed will be the same regardless of how positioning data is physically stored, how fast it is queried, with what technology it is gathered and who has legal access to it.

8.4 Further Work

Our research covers only a small part of a vast area regarding databases for location aware systems, as seen by the limitations. Needless to say, further research in those areas is required in order to expand our knowledge of such databases and create more effective and flexible solutions. Below, we present other areas which we consider to be more closely related to our work, and propose ideas for future research in those fields:

- **Approximate queries**: Incoming data to an indoor location aware application will typically be in the form of data streams, e.g. through sensors placed in the building. In most cases the total amount of the data gathered over time will be very large and require rapid updates. Imagine for example if we were to store the individual positions of 100 persons once every minute over a period of five years. Physically storing such large amounts of data is possible, but difficult. Even if we were to somehow store all of it, no indexing technique currently available would be efficient due to its size, and exact query processing would be expensive. Approximate query processing focuses on approximate summerized information about objects that satisfy a set of spatio-temporal predicates and thus

reduces the processing costs put on the database. An example of an approximate query can be to calculate the number of persons in a room at a given time, as opposed to exact information about those persons (e.g. names, statuses, roles).

- **Uncertainty**: All positioning systems are faced with the inherent challenge of uncertainty. The question is, simply put: how accurate is the positioning information received from the sensors? If a person is tracked to a particular position, how certain can we be that this person actually is at that position? Another interesting and perhaps more complicated issue regarding uncertainty is the question of tracking who we think we are tracking. Since most location aware applications communicate with their surroundings through sensors and not face/voice recognition, how can we be certain that the person we think is carrying a particular sensor actually is carrying it?

- **Privacy/legal issues**: Location aware applications used to track the position of people have a great potential for abuse. Management and use of personal data is most often bound by legislative restrictions. Using such information without the explicit consent of the parties involved can lead to legal consequences. However, it is not only the legal aspects that are important to consider, but also the moral and ethical points of view. Questions that should be asked are: Is information about a persons position in public buildings or a private building not owned by that person deemed as personal data on the same level as for example social security numbers? Who should have access to such information, and what limitations should be set for its use?

8.5 Final Remarks

This thesis has been more than just a scientific paper. It has been a learning experience in the field of research. It has taught us how such work should be conducted, from the initial planning and draft, till the final evaluation. We have learned how to gather, organize and use scientific data effectively, both

regarding related research, and on the research subject itself.

Another important lesson learned during the past year is that knowledge is found in many different places, not just in the library, through supervisors or on the internet. Other students, scholars at other institutes of higher learning and developers of particular technologies used in our prototype have contributed with knowledge, ideas, suggestions and answers which we ourselves would never have thought of.

Although our work merely touches a small fraction of spatio-temporal database research, we believe that this is how research and development is driven forward, where scientists, through their work, contribute with one of the missing pieces of an endless puzzle. Keeping this in mind, we have tried to provide research material that is not too trivial in terms of theoretical contribution, and at the same time not too large in terms of practical implementation and testing that is required to validate the result.

Finally, during the course of our work, we have realized that *patience* indeed is a virtue.

Appendix

The following CD contains the complete source code for the prototype database, along with test data and example queries. An example application is also included, also with its source code. To install the database system, insert testdata and run queries and the application, please read the manual on the CD ("index.html").

References

[1] Abel, David; Ooi, Beng Chin (eds.) (1993): *Proceedings of the 3rd International Symposium on Large Spatial Databases*, Singapore, LNCS 692, Springer

[2] Abraham, Tamas; Roddick, John (1999): "Survey of Spatio-Temporal Databases", *GeoInformatica*, Vol. 3, No. 1, p. 61-69

[3] Atkinson, Malcolm; Bancilhon, François; DeWitt, David; Dittrich, Klaus; Maier, David; Zdonik, David (1989): "The Object-Oriented DatabaseSystem Manifesto", *Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD)*, p. 223-240

[4] Bancilhon, François (1992): "The O_2 Object-Oriented Database System", *Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data*, p. 7

[5] Bauer, Martin; Becker, Christian; Rothermel Kurt (2001): "Location Models from the Prespective of Context-Aware Applications and Mobile Ad Hoc Networks", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[6] Beigl, Michael; Decker, Christian; Zimmer, Tobias (2002): "A Location Model for Communicating and Processing of Context", *Personal and Ubiquitous Computing*, Vol. 6, No. 5-6, p. 341-357

[7] Becker, Christian; Dürr, Frank (2005): "On Location Models for Ubiquitous Computing", *Personal and Ubiquitous Computing*, Vol. 9, No. 1, p. 20-31

[8] Beynon-Davies, Paul (2004): *Database Systems*, New York, USA: Palgrave Macmillan, p. 87-155, 381-425 and 512-523

[9] Böhlen, Michael; Busatto, Renato; Jensen, Christian (1998): "Point- Versus Interval-based Temporal Data Models", *Proceedings of the 14th International Conference on Data Engineering (ICDE)*, Orlando, p. 192-200

[10] Brumitt, Barry; Shafer, Steven (2001): "Topological World Modeling Using Semantic Spaces", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[11] Buchmann, Alejandro; Günther, Oliver; Smith, Terrence; Wang, Yuan-F. (eds.) (1989): *Proceedings of the First International Symposium on Large Spatial Databases*, Santa Barbara, LNCS 409, Springer

[12] Chen, Peter (1976): "The Entity-Relationship Model: Toward a Unified View of Data", *ACM Transactions on Database Systems*, Vol. 1, No. 1, p. 9-36

[13] Codd, Edgar Frank "Ted" (1970): "A Relational Model of Data for Large Shared Data Banks", *Communications of the ACM*, Vol. 13, No. 6, p. 377-387

[14] Cornford, Tony; Smithson, Steve (1996): *Project Research in Information Systems - A Student's Guide*, New York, USA: Palgrave

[15] Domnitcheva, Svetlana (2001): "Location Modeling: State of the Art and Challenges", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[16] Dürr, Frank; Rothermel, Kurt (2003): "On a Location Model for Fine-Grained Geocast", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2003, Technical Report

[17] Egenhofer, Max (1994): "Spatial SQL: A Query and Presentation Language", *IEEE Transactions on Knowledge and Engineering*, Vol. 6, No. 1, p. 86-95

[18] Egenhofer, Max; Frank, Andrew; Jackson Jeffrey (1989): "A Topological Data Model for Spatial Databases", *Design and Implementation of Large Spatial Databases*, First Symposium SSD '89, p. 271-286

[19] Egenhofer, Max; Herring, John (eds.) (1995): *Proceedings of the 4th International Symposium on Large Spatial Databases*, Portland, LNCS 951, Springer

[20] Erwig, Martin; Güting, Ralf Hartmut; Schneider, Markus; Vazirgiannis, Michalis (1998): "Abstract and Discrete Modeling of Spatio-Temporal Data Types", *Proceedings of the 6th ACM International Symposium on Advances in Geographic Information Systems*, Whashington, p. 131-136

[21] Erwig, Martin; Güting, Ralf Hartmut; Schneider, Markus; Vazirgiannis, Michalis (1999): "Spatio-Temporal Data Types: An approach to Modeling and Querying Moving Objects in Databases" (Revised Version), *Geoinformatica*, Vol. 3, No. 3, 269, 296

[22] Erwig, Martin; Schneider, Markus (2002): "STQL - A Spatio-Temporal Query Language", in Ladner, Roy; Shaw, Kevin; Abdelguerfi, Mahdi (eds.) (2002): *Mining Spatio-Temporal Information Systems*, Boston, USA: Kluwer Academic Publishers, p. 105-126

[23] Etzion, Opher; Jajodia, Sushil; Sripada, Suryanarayana (eds.) (1998): *Temporal Databases: Research and Practice*, LNCS 1399, Springer

[24] Faria, Glaucia; Medeiros, Claudia; Nascimento, Mario (1998): "An Extensible Framework for Spatio-Temporal Database Applications", *TIMECENTER Technical Report TR-37*

[25] Funk, Harry; Miller, Christopher (2001): "Location Modeling for Ubiquitous Computing: Is This Any Better?", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[26] Gregersen, Heidi; Jensen, Christian (1998): "Conceptual Modeling of Time-varying Information", *TIMECENTER Technical Report TR-35*

[27] Günther, Oliver; Schek, Hans-Jörg (eds.) (1991): *Proceedings of the 2nd International Symposium on Large Spatial Databases*, Zurich, LNCS 525, Springer

[28] Güting, Ralf Hartmut (1994): "An Introduction to Spatial Databases", *Special Issue on Spatial Database Systems of the VLDB Journal*, Vol. 3, No. 4, p. 357-399

[29] Güting, Ralf Hartmut; Schneider, Markus (1995): "Realm-Based Spatial Data Types: The ROSE Algebra", *The VLDB Journal - The International Journal on Very Large Databases*, Vol. 4, No. 2, p. 243-286

[30] Güting, Ralf Hartmut; Papadias, Dimitris; Lochovsky, Fred (eds.) (1999): *Proceedings of the 6th International Symposium on Large Spatial Databases*, Hong Kong, LNCS 1651, Springer

[31] Güting, Ralf Hartmut; Böhlen, Michael; Erwig, Martin; Jensen, Christian; Lorentzos, Nikos; Schneider, Markus; Vazirgiannis Michalis (2000): "A Foundation for Representing and Querying Moving Objects", *ACM Transactions on Database Systems*, Vol. 25, No. 1, p.1-42

[32] Guttman, Antonin (1984): "R-Trees, A Dynamic Index Structure for Spatial Searching", *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, p. 47-57

[33] Hadzilacos, Thanasis; Manolopoulos, Yannis; Roddick, John; Theodoridis, Yannis (eds.) (2003): *Proceedings of the 8th International Symposium on Large Spatial and Temporal Databases*, Santorini Island, LNCS 2750, Springer

[34] Harter, Andy; Hopper, Andy (1994): "A Distributed Location System for the Active Office", *IEEE Network*, Vol. 8, No. 1, p. 62-70

[35] ISO (1987): "Database Language SQL", ISO/IEC 9075, Geneva, International Standards Organization

[36] ISO (1992): "Database Language SQL", ISO/IEC 9075, Geneva, International Standards Organization

[37] ISO (1999): "Database Language SQL - Part 2: Foundation", ISO/IEC 9075-2, Geneva, International Standards Organization

[38] ISO (1999): "Database Language SQL - Part 2: Persistent Stored Modules", ISO/IEC 9075-4, Geneva, International Standards Organization

[39] Jensen, Christian; Schneider, Markus; Seeger, Bernhard; Tsotras, Vassilis (eds.) (2001): *Proceedings of the 7th International Symposium on Large Spatial and Temporal Databases*, Redondo Beach, LNCS 2121, Springer

[40] Jiang, Changhao; Steenkiste, Peter (2001): "A Hybrid Location Model with Computable Location Identifier for Ubiquitous Computing", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[41] Kamil, Grajski; Kirk, Ellen (2003): "Towards a Mobile Multimedia Age - Location-Based Services: A Case Study", *Wireless Personall Communications*, Vol. 26, Kluwer Academic Publishers, p. 105-116

[42] Korkea-Aho, Mari; Tang Haitao (2001): "Experiences of Expressing Location Information for Applications in the Internet", *Workshop on Location Modeling for Ubiquitous Computing*, UBICOMP 2001, Technical Report

[43] Koubarakis, Manolis; Sellis, Timos et al (eds.) (2003): *Spatio-Temporal Databases, The CHOROCHRONOS Approach*, LNCS 2520, Springer

[44] Kuhn, Thomas (1996): *The Structure of Scientific Revolutions*, Chicago: University of Chicago Press

[45] Leonhardt, Uli (1998): *Supporting Location-Awareness in Open Distributed Systems*, PhD-thesis, Imperial College London, Department of Computing

[46] Leung, Hubert Ka Yau; Burcea, Ioana; Jacobsen, Hans-Amo (2003): "Modeling Location-Based Services with Subject Spaces", *Proceedings of the 2003 Conference of the Centre for Advanced Studies Conference on Collaborative Research*, Toronto, October 6.-9., p. 171-181

[47] March, Salvatore; Smith, Gerald (1995): "Design and natural Science research on information technology", *Decision Support Systems*, Vol. 15, No. 4, p. 251-266

[48] Naiburg, Eric; Maksimchuck, Robert (2001): *UML for Database Design*, Boston, USA: Addison-Wesley, p. 119-148

[49] Parent, Christine; Spaccapietra, Stefano; Zimànyi, Esteban (1999): "Spatio-Temporal Conceptual Models: Data Structures + Space + Time", *Proceedings of the 7th ACM International Symposium on Advances in Geographic Information Systems*, p. 26-33

[50] Rao, Bharat; Minakakis, Louis (2001): "Evolution of Mobile Location-Based Services", *Communications of the ACM*, Vol. 46, No. 12, p. 61-65

[51] Robson, Colin (2002): "Approaches to Social Research", in Robson, Colin: *Real World Research*, Oxford, UK: Blackwell

[52] Roddick, John; Patrick, Jon (1992): "Temporal Semantics in Information Systems - A Survey", *Information Systems*, Vol. 17, Nr. 3, p. 249-267

[53] Scholl, Michel; Voisard, Agnès (eds.) (1997): *Proceedings of the 5th International Symposium on Large Spatial Databases*, Berlin, LNCS 1262, Springer

[54] Sellis, Timos (1999): "CHOROCHRONOS - Research on Spatio-Temporal Database Systems" in Agouris, Peggy; Stefanidis, Anthony (eds.) (1999): *Integrated Spatial Databases: Digital Images and GIS*, Portland, LNCS 1737, Springer, p. 308-316

[55] Shashi, Gadia (1988): "A Homogenous Relational Model and Query Languages for Temporal Databases" *ACM Transactions on Database Systems*, Vol. 13, No. 4, p. 418-448

[56] Smith, Clint; Collins, Daniel (2002): *3G Wireless Networks*, New York, USA: McGraw-Hill, p. 59-80

[57] Snodgrass, Richard (ed) (1995): *The TSQL2 Temporal Query Language*, Boston, USA: Kluwer Academic Publishers

[58] Sparks, Geoffrey (2001): "Database Modelling in UML", *Methods & Tools*, Vol. 9, No. 1, p. 10-23

[59] Stonebraker, Michael (1996): *Object-Relational DBMSs: The Next Great Wave*, San Fransisco, USA: Morgan Kauffman

[60] Takeda, Hideaki; Veerkamp, Paul; Tomiyama, Tetsuo; Yoshikawam, Hiroyuki (1990): "Modeling Design Processes", *AI Magazine*, Vol 11, No. 4, p. 37-48

[61] Tichy, Walter (1998): "Should Computer Scientists Experiment More?", *IEEE Computer*, Vol. 31, No. 5, p. 32-40

[62] Torp, Kristian; Snodgrass, Richard; Jensen, Christian (1997): "Correct and Efficient Timestamping of Temporal Data", *TIMECENTER Technical Report TR-4*

[63] Tryfona, Nectaria; Jensen, Christian (1999): "Conceptual Data Modeling for Spatio-Temporal Applications", *Geoinformatica*, Vol. 3, No. 3, p. 245-268

[64] Tryfona, Nectaria; Jensen, Christian; Price, Rosanne (2003): "Conceptual Data Modeling for Spatio-Temporal Applications (Revised Version)", in [43], p. 79-117

[65] Want, Roy; Hopper, Andy; Falcão, Veronica; Gibbons, Jonathan (1992): "The Active Badge Location System", *ACM Transactions on Information Systems*, Vol. 10, No. 1, p. 91-102

[66] Worboys, Michael (1994): "A Unified Model for Spatial and Temporal Information", *Computer Journal*, Vol. 37, No. 1, p. 26-34

[67] http://www.esri.com/software/arcgis/arcinfo/
(last visited March 7, 2005)

[68] http://www.esri.com/software/arcview/
(last visited March 7, 2005)

[69] http://www.cs.aau.dk/ tigeradm/atsql.html
(last visited March 7, 2005)

[70] http://www-306.ibm.com/software/data/db2/udb/
(last visited March 7, 2005)

[71] http://www-306.ibm.com/software/data/spatial/
(last visited March 7, 2005)

[72] http://www.ekahau.com/
(last visited March 7, 2005)

[73] http://www.ekahau.com/pdf/EPE_3.0_datasheet.pdf
(last visited March 7, 2005)

[74] http://www.ekahau.com/products/engine/
(last visited March 7, 2005)

[75] http://www.gis.com/whatisgis/
(last visited March 7, 2005)

[76] http://www.gsmworld.com/
(last visited March 7, 2005)

[77] http://www.gpsworld.com/gpsworld/
(last visited March 7, 2005)

[78] http://java.sun.com/products/jdo/
(last visited March 7, 2005)

[79] http://www.mysql.com
(last visited March 7, 2005)

[80] "MySQL Reference Manual"
http://dev.mysql.com/doc/
(Last visited March 7, 2005)

[81] http://dev.mysql.com/doc/mysql/en/Spatial_extensions_in_MySQL.html
(last visited March 7, 2005)

[82] "Location Based Services: Considerations and Challanges",
http://www.northstream.se/download/LocationBasedServices.pdf
(last visited March 7, 2005)

[83] http://www.objectdb.com
(last visited March 7, 2005)

[84] http://www.objectdb.com/database/jdo/manual/odb-manual.pdf
(last visited March 7, 2005)

[85] http://www.opengeospatial.org/
(last visited March 7, 2005)

[86] "OpenGIS Simple Features Specification For SQL"
http://www.opengeospatial.org/docs/99-049.pdf
(last visited March 7, 2005)

[87] http://www.oracle.com/database/
(last visited March 7, 2005)

[88] http://www.oracle.com/technology/products/spatial/
(last visited March 7, 2005)

[89] http://postgis.refractions.net/
(last visited March 7, 2005)

[90] http://www.postgresql.org/
(last visited March 7, 2005)

[91] http://www.cs.auc.dk/TimeCenter/
(last visited March 7, 2005)

[92] http://www.uml.org/
(last visited March 7, 2005)

[93] http://www.w3.org/XML/
(last visited March 7, 2005)

Index