

Hierarchical Structures in Map Series

by

Sabine Timpf

A thesis submitted in partial fulfillment of the requirements
of the degree of
Doctor of Technical Sciences

submitted at the
Technical University Vienna
Faculty of Science and Technology

January 1998

Advisory Committee:

Univ.-Prof. Dr. Andrew Frank

Department of Geoinformation,
Technical University Vienna

Univ.-Prof. Dr. Markus Stumptner

Database and Expert Systems Group,
Technical University Vienna

Vienna, January 1998

Abstract

Among the different operations in Geographic Information Systems (GIS), changing scale is crucial for the handling and analysis of multi-scale data. The term multi-scale is used to indicate that there are data of more than one level of detail in the database. Different levels of detail are needed when dealing with spatial data. Current scaling functions blow up and shrink the objects to display or change to a radically different representation of the display.

In this research we investigate how to link map objects at different levels of detail that represent the same real-world entity. We observe the behavior of map objects at four levels of detail in eight case studies. We apply an observation method derived from the method of abduction. It relies on careful and controlled observation of phenomena, similar to the method of reverse engineering in computer science. We define four types of map objects to be observed:

- a trans-hydro network, that is a combination of the street network, the railroad network, and the hydrographic network,
- containers, that represent the regions between the links of the trans-hydro network,
- areas, that represent the land-use areas and are contained in the containers, and
- map elements, that are contained in the areas, representing buildings, symbols, and dead-end streets and railroads.

We propose a hierarchical data structure that supports and describes the behavior of map objects over four levels of detail. Hierarchical data structures are best represented by trees or forests. The data structure of the map cube model is a combination of three different trees, representing three different types of hierarchies.

Hierarchies are defined according to the functions associated with the edges of their trees: aggregate, generalize, and filter. The aggregation hierarchy is built by aggregating objects. The generalization hierarchy defines how classes relate to more generic super classes. The filter hierarchy filters objects according to a filter criterion.

The *map cube model* can be imagined as a three dimensional composition. Each map is a level in a three-dimensional cube, the vertical axis represents the level of detail.

The hierarchical data structure for the map series is a combination of the trans-hydro network, a tree of containers including the information on areas and elements, an area graph, and an element graph. The tree of containers stores which container includes which areas and elements. It does not store how these areas and elements are related to areas and elements on a lower or higher level of detail. This information is encoded in the area graph or element graph, respectively. We formalize the map cube model in a functional programming language with an object-oriented paradigm.

In our model the trans-hydro network is represented by a filter hierarchy, the filter criterion is the relative importance of the link in the network. The container tree is an aggregation hierarchy - the more abstract the representation, the more containers are aggregated. The area graph is a collection of area trees; each tree represents a generalization hierarchy. The element graph is a collection of trees; each tree is a constructed by several functions. Those functions are 'omit', 'typify', 'merge', and 'continue'. The first three functions correspond to the cartographic functions with the same name. Thus, the model incorporates all three types of hierarchies.

Hierarchical Structures in Map Series

“Scale is an important consideration in nearly all geographical studies. Careful selection of appropriate scales in space and time is required to formulate and answer meaningful questions. Because trends discernible at one scale are often invisible at another, geographers commonly use a range of scales to ensure that the conclusions drawn from a study are well matched to the economic, social, and physical processes known to underlie observed patterns.”

John C. Hudson: Scale in Space and Time, p.296 (Hudson, 1992).

Acknowledgments

As in any greater endeavor, there are many people that helped and provided support. In particular, I want to express my profound thanks to my advisor Andrew U. Frank for his guidance and patience in the last four years. He kept me from straying too far from the topic I chose without restricting my interest in other areas. He also provided an interesting and enthusiastic research environment, where working on a thesis is a joy. I am grateful to my second advisor, Markus Stumptner, for his patience and effort.

Werner Kuhn is a wonderful colleague, friend, and my greatest moral support during those years. He always found time for discussion and helped me to gain insights into cognitive science. He was there when help was most needed. I thank him for his valuable advice and for having an open ear, especially to distressed voices.

My colleagues at the department have been incredibly understanding and put up with me being grumpy during some of the harder days. During my time at the department they enhanced my learning experience through discussion and quiet encouragement. I thank them for their kindness, their understanding, and their positive criticism. Especially, I am grateful to Thomas Bittner for stimulating discussions about interesting papers. Adrijana Car is a great friend and companion on the way to a finished thesis. I thank her for the marvelous time we had laughing. Peter Haunold held up morale and helped with earlier drafts of the thesis. I acknowledge his help with stubborn computers and his criticism ‘à la Barry’. Edith Unterweger has encouraged me time and time again. I thank her for pounding my back once in a while.

Among the colleagues and friends outside the department, I thank Katrin Dyballa for the moral and philosophical support she has given me. I am grateful to Marion Czeranka, who commented on earlier versions of the thesis. Thomas Wilke spurred the will to survive this period with his sarcastic comments. Helge and Roger Seeck provided solace during the dark moments. Arno Prokes helped me to see things from a different perspective every once in a while. Credits go also to Doug Flewelling for encouragement from afar and to Max Egenhofer for heartening comments about the topic and the research environment.

I also thank my mother, Liselotte Timpf, who has become a friend during those years. It was wonderful to take the step from daughter to friend. Marion Timpf's emails were always encouraging and lighted the days with laughter. Thanks to Mutti, Marion, and Karl-Heinz for their understanding and kindness.

When I least expected it, I found a true and loving friend. Werner Porod provided soothing comments, great cooking, and help at the last moment. I am deeply grateful for his caring, for his understanding, and especially for his patience.

Table of Contents

Chapter 1 - Introduction	1
1.1 Motivation and Problem Statement.....	1
1.2 Hypothesis and Goal.....	4
1.3 Approach.....	5
1.4 Expected Results	7
1.5 Contribution and Intended Audience.....	7
1.6 Organization of this Thesis	8
Chapter 2 - Background and Previous Work	11
2.1 Multiple Representations.....	11
2.2 Cartographic Generalization.....	13
2.3 User Interfaces and Computer Graphics.....	15
2.4 Hierarchical Data Structures.....	16
Chapter 3 - Methods of Investigation and Formal Tools	19
3.1 Observation Method	19
3.2 Graph Theory.....	21
3.3 Partition and Refinement.....	24
3.4 Formalization with a functional language	25
3.4.1 Model and Language requirements	25
3.4.2 The functional programming language Gofer.....	27
3.5 Summary.....	29
Chapter 4 - Hierarchies	31
4.1 Hierarchies in Cognitive Science.....	32
4.2 Hierarchies in Systems Theory	33
4.3 A Definition of Hierarchy	34
4.4 Types of Hierarchies.....	35
4.4.1 Aggregating	36
4.4.2 Generalizing.....	36
4.4.3 Filtering.....	37

4.5 Combining Hierarchies	38
4.6 Summary	38
Chapter 5 - Map Model and Map Cube Model	39
5.1 The Map Model.....	39
5.1.1 Ontology of Map Elements	40
5.1.2 Behavior of Graphic Objects.....	42
5.1.3 A Formal Map Model.....	44
5.1.4 Remarks.....	45
5.2 The Map Cube Model	45
5.2.1 Trans-hydro Graph.....	46
5.2.2 Container Graph.....	47
5.2.3 Area Graph.....	48
5.2.4 Element Graph	49
5.2.5 The Cube Model	49
5.3 Summary	51
Chapter 6 - Observation Plan	53
6.1 Map series.....	53
6.2 Comparing Maps	54
6.2.1 Rules for Comparison of Map Objects.....	55
6.2.2 Remarks to the Observation Process	56
6.3 A Strategy for Matching Map Objects.....	56
6.3.1 Building a Container Graph.....	57
6.3.2 Building an Area Graph.....	58
6.3.3 Building an Element Graph	59
6.5 Summary	60
Chapter 7 - Case Studies	61
7.1 Overview of Study areas.....	61
7.2 Study area Berneburg I.....	64
7.3 Study area Berneburg II/III.....	66
7.4 Study area Berneburg IV	68

7.5 Study area Alsleben I.....	69
7.6 Study area Alsleben II	71
7.7 Study area Alsleben III.....	73
7.8 Study area Murten I	75
7.9 Study area Murten II.....	77
7.10 Summary.....	79
Chapter 8 - Analysis and Formalization	79
8.1 The Issue of Consistency.....	79
8.2 Trans-Hydro Network and Trans-Hydro Graph.....	81
8.3 Containers and Container Graph.....	83
8.4 Areas and Area Graph.....	84
8.5 Elements and Element Graph	86
8.6 The Map Cube Model.....	87
8.6.1 Relations between Graphs.....	88
8.6.2 Recommended Data Structure	89
8.7 Summary.....	90
Chapter 9 - Results and Future Work	91
9.1 Summary.....	91
9.2 Results and Conclusions.....	94
9.3 Discussion	95
9.4 Future Work	96
Appendix - Gofer Code.....	99
A.1 Generalities.....	99
A.2 Generic Graph	100
A.3 Map Object	102
A.4 Trans-hydro Network	103
A.5 Rose Tree.....	104
A.6 Map and Map Series.....	105
A.7 Example	106
References	113

List of Figures

Fig. 1.1:	Historical atlas - Zoom.....	1
Fig. 1.2:	Linking map objects in a map series.....	3
Fig. 1.3:	Levels in a map series.....	5
Fig. 1.4:	Map Cube with two levels.....	7
Fig. 3.1:	Reduction (a) and deduction (b) in a logical system.....	20
Fig. 3.2:	Acyclic Graph.....	23
Fig. 3.3:	Several types of trees.....	23
Fig. 3.4:	Progress for modeling of complex systems.....	25
Fig. 4.1:	Partial ordering: tree.....	34
Fig. 4.2:	Levels in a hierarchy.....	34
Fig. 4.3:	Visualization of a generalization hierarchy.....	37
Fig. 4.4:	Filter hierarchy.....	37
Fig. 5.1:	Map objects.....	39
Fig. 5.2:	Trans-hydro network creates partitions of space at three levels of detail...	39
Fig. 5.3:	Model of a map.....	44
Fig. 5.4:	Objects depicted with increasing graphical detail.....	46
Fig. 5.5:	Trans-hydro Graph.....	46
Fig. 5.6:	Container Graph.....	47
Fig. 5.7:	Area Graph.....	48
Fig. 5.8:	Element Graph.....	49
Fig. 5.9:	Map Cube with two levels.....	50
Fig. 5.10:	Components of a digital map series.....	50
Fig. 6.1:	Map clips at different levels of detail shown at same nominal scale.....	53
Fig. 6.2:	Relations between maps at different levels of detail.....	54
Fig. 6.3:	Representation of example case study at three levels of detail.....	57
Fig. 6.4:	Networks for spatial subdivision and Container Tree.....	57
Fig. 6.5:	Containers and Areas.....	58
Fig. 6.6:	Area Tree.....	59
Fig. 6.7:	Element Graph.....	59
Fig. 7.1:	Berneburg/Saale.....	62

Fig. 7.2:	Alsleben/Saale.....	62
Fig. 7.3:	Murten.....	63
Fig. 7.4:	Study area Berneburg I at different levels of detail.....	64
Fig. 7.5:	Container Tree (a) and Area Tree (b) of Berneburg I.....	64
Fig. 7.6:	Element Graph of Berneburg I.....	65
Fig. 7.7:	Study area Berneburg II/III at several levels of detail.....	66
Fig. 7.8:	Container Graph and Area Graph of Berneburg II/III.....	66
Fig. 7.9:	Element Graph of Berneburg II/III.....	67
Fig. 7.10:	Berneburg IV at different levels of detail.....	68
Fig. 7.11:	Container Tree (a), Area Tree (b) and Element Graph (c).....	68
Fig. 7.12:	Study area Alsleben I at different levels of detail.....	69
Fig. 7.13:	Container Graph of Alsleben I.....	69
Fig. 7.14:	Area Graph of Alsleben I.....	70
Fig. 7.15:	Element Graph of Alsleben I.....	70
Fig. 7.16:	Alsleben II at different levels of detail.....	71
Fig. 7.17:	Container Tree of Alsleben II.....	71
Fig. 7.18:	Area Tree of Alsleben II.....	72
Fig. 7.19:	Element Graph of Alsleben II.....	72
Fig. 7.20:	Alsleben III at different levels of detail.....	73
Fig. 7.21:	Container Tree (a) and Area Graph (b) of Alsleben III.....	73
Fig. 7.22:	Element Graph of Alsleben III.....	74
Fig. 7.23:	Study area Murten I at different levels of detail.....	75
Fig. 7.24:	Container Tree (a) and Area Tree (b) of Murten I.....	75
Fig. 7.25:	Element Graph of Murten I.....	75
Fig. 7.26:	Study Area Murten II at different levels of detail.....	76
Fig. 7.27:	Container Tree of Murten II.....	77
Fig. A.1:	Map series Berneburg I.....	106
Fig. A.2:	Graph representation of the study area Berneburg I.....	106
Fig. A.3:	Containers over four levels of detail.....	108
Fig. A.4:	Areas over four levels of detail.....	108

List of Tables

Table 3.1: Definition of a partition	24
Table 3.2: Example for a tree of integers	28
Table 3.3: Gofer code for a rose tree	29
Table 5.1: Graphical element changes	43
Table 7.1: Information on map series Berneburg and Alsleben.....	61
Table 7.2: Information on map series Murten	61
Table 8.1: Gofer code for the classes <i>TranS</i> and <i>TraS</i>	81
Table 8.2: Filtering a list of edges.....	82
Table 8.3: Gofer code of a Container	83
Table 8.4: Example for a container tree in Gofer.....	84
Table 8.5: Gofer code of an Area	84
Table 8.6: Gofer code of an Element.....	86
Table 8.7: Gofer code for a map database.....	88
Table 9.1: Map Elements.....	92
Table 9.2: Types of hierarchies and Graphs.....	94

Chapter 1 - Introduction

The research topic addressed in this work is part of the area of multiple representations in Geographic Information Systems (GIS). The term multiple representations in GIS denotes that more than one representation of a geographic entity is included in the database of the information system. In our first section, the motivating example shows that less detailed as well as more detailed representations of data are desirable in information systems dealing with geographic data (section 1.1).

In this research, we use a series of topographic maps to investigate what kind of data structures are suitable for geographic databases with multiple representations. We hypothesize that the data structure is a hierarchical structure and that there are several types of object hierarchies superimposed in a map series (section 1.2).

The goal of this research is to observe the structures and transformations of objects in maps over several levels of detail. Our approach (section 1.3) is to observe carefully and document formally the structures we find in map series. In the final sections of this chapter we present the results to be expected, the intended audience of this research, and an overview on the content of the subsequent chapters.

1.1 Motivation

Imagine, someone just bought a digital historical atlas. They are interested in the political situation of northern Italy at the end of the 18th century.



Fig. 1.1: Historical atlas - Zoom (Centennia, 1992-1995)

They click on the zoom button to see Italy (Fig.1.1). And then they zoom again to see the area around Venice. They are interested to know the name of the little state just south of Mantua and zoom again. Unfortunately, the program's zooming range is at its end and the name cannot be read. There is no means to enlarge the display more than it already is inside the program.

At the heart of this problem lies the fact that most map databases have only data at a single scale to display. However, humans use information at multiple levels of detail. Especially in geography, a range of scales is used to make sure that the conclusions drawn are correct and reached efficiently. A function to draw cartographic sketches quickly and at arbitrary scales is needed, ranging from overview screens to detailed views (Herot *et al.*, 1980)

Geographic Information Systems (GIS) are used for decision making and, therefore, need data at more than one scale in the database. For example, to navigate in a city, a scale range of 1:10,000 to 1:30,000 is suitable. Maps in this scale range provide detailed information. For inter-city navigation a range from 1:50,000 to 1:200,000 is necessary. For inter-country travels maps of the scales from 1:300,000 to 1:1,000,000 are used. Traveling by car from Rome (Italy) to London (UK) requires more than one map: an overview map to fix the general direction, a medium scale map to find the daily stops, and a detailed map to make the connection between the faster road network and the city street network. Thus, different levels of detail are needed when dealing with spatial data (Minsky, 1985).

A map is a collection of graphical objects that are either symbols or approximate representations (in shape and location) of observable entities in the real world. Some map objects are combinations of these two, e.g. an area which includes forest symbols. The approximation of real-world entities can be by shape or by prototypes of shapes of classes. A prototypical shape for a house is a rectangle, for a street is a line with a certain width. We use map objects from topographic maps at scale 1:10,000 to scale 1:500,000. Maps that depict the same area and are arranged in an order (by scale or by level of detail) are called a *map series*.

Cartographers have created overview maps from detailed maps for centuries. The process of cartographic generalization is a combination of skill and art. It would be an elegant solution to store the most detailed data in the database and deduce data at other levels of detail automatically. Attempts to automate the process of cartographic generalization have been successful in special cases, but a fully automated process is not available (Buttenfield and McMaster, 1991; Mueller, Lagrange, and Weibel, 1995).

A less elegant, but feasible solution is a multi-level database that contains several levels of increasingly detailed data. The problem with this approach is that objects (e.g., the little state south of Mantua) exist on every level in the database but cannot be automatically linked to each other. The link is crucial to determine which objects on separate levels represent the same real-world object.

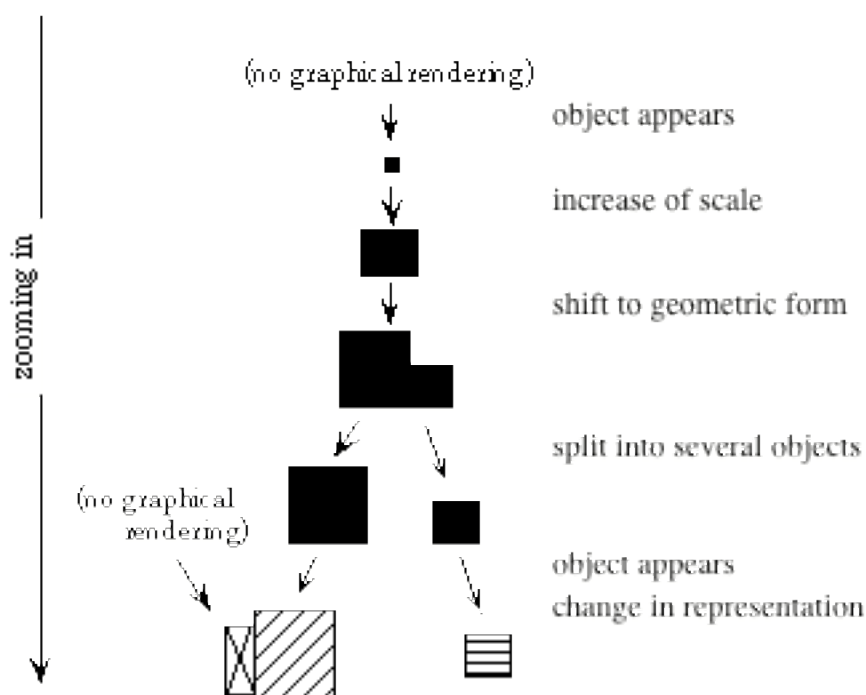


Fig. 1.2: Linking objects in a map series

Linking objects at several *levels of detail* in a database requires a hierarchical data structure. For example, one object at a high level is represented by more than one object at a lower level. Linking these objects results in a tree structure, which is a special type of hierarchy (Fig. 1.2). Hierarchies are used by everyone in everyday life, but the phenomenon is not very well researched in the field of databases for cartographic data. A

hierarchical data structure for multi-level data is a prerequisite in the context of databases for multiple representations.

We aim at supporting functions such as an intelligent zoom (Bjørke and Aasgaard, 1990; Frank and Timpf, 1994) with specially adapted data structures. These data structures must provide data at several levels of detail and enable links between objects at one level of detail to objects at another level of detail. The data we support is geographic data. In this research we investigate in detail cartographic data from analog map series.

1.2 Hypothesis and Goal

Our hypothesis is that in map series several types of object-hierarchies are superimposed. These different hierarchies can be separated and a multi-level data structure can be constructed and formalized. This process includes formalizing different types of hierarchies and describing their interaction. Hierarchies are distinguished by the function that leads from one level of a hierarchy to the next level.

Our first goal is to investigate how objects at several levels of detail can be linked to each other. The idea is to assign to each map object those map objects at the levels below and above that represent the same real-world entity. We then describe how the object at the lower level is transformed into the object at the higher level. The sum of transformations leading from a low level object to an abstract object is called behavior of that object. We say that the object lives; in this notation the levels of detail denote time (Timpf and Frank, 1997a).

We observe how map objects behave over four levels of detail. For a human this seems like a straightforward task. A computer, however, needs additional assistance in identifying map objects as being related to each other. Therefore, the second goal is to derive a data structure that can support and describe the behavior of map objects. We derive a hierarchical data structure from map series that is suited for the representation of map data in a database. The result of this study is a formal model of a map series.

This work does not consider the representations of continuous surfaces for multiple levels of resolution, i.e. isolines or triangulations. The topic is studied in detail by several

research groups (Weibel and Heller, 1991; Floriani and Puppo, 1992; Gold, 1992).

Therefore, we leave out the visualization of isolines in topographic maps.

1.3 Approach

Our approach towards a formal model of map series is similar to methods in psychology and the social sciences: A theory is tested with an experiment and the results are used to derive a formal model of the theory. In this research, we first observe map objects over four levels of detail (Fig. 1.3) and describe their structure and behavior informally. Then, we formalize the different parts of the structure and combine them into a formal model of a map series.

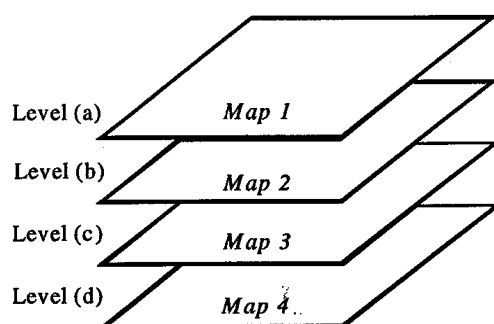


Fig. 1.3: Levels in a map series

The objects we are studying are neither purely graphical objects nor purely geographic entities. They are objects resulting from the observation of cartographic maps plus some semantic information we have from our common-sense knowledge of geographic space. E.g., a house block is an object resulting from the cartographic representation, which is in fact not one object but several ones merged into one meaningful abstracted object. Knowing that house blocks are merged from smaller entities we also realize that the reasoning process on house blocks is restricted.

Every observation is theory dependent (Chalmers, 1994). So we first outline our theory on the structure of maps and map series. Then we observe carefully and systematically how objects change over four levels of detail, bringing the maps from the map series to the same scale. An object is represented in several maps and its representation changes with a change in level of detail. For example, when zooming in, objects are enlarged, they split, and more and more objects appear (see Fig. 1.2). When

zooming out, objects disappear, collapse, or are merged into larger objects. We describe the behavior of map objects informally either in figures or in functions. This method is called abduction, which is a reductive method similar to induction. In computer science an application of the method of abduction is also known as reverse engineering.

It is necessary to make a distinction between the operations on the database and the behavior of the graphic objects. All operations in the database entail a change of the graphic object, but not all changes of the graphic objects are reflected in the database operations. Those changes that are not reflected in the database are purely graphical operations. They depend on the space of the screen and on the relations between the graphic objects.

For detailed observation three study regions have been selected: Murten in Switzerland, and Alsleben and Berneburg in Germany. The German maps use the same symbolization scheme for all map scales and this fact simplifies comparison across the levels. The Swiss maps have only slightly different symbolization schemes, that do not prevent comparison of map objects. The reason to use topographic map series is that in map series the knowledge of deriving multiple less detailed representations is already encoded. Our approach is also suitable for the knowledge acquisition for formal cartographic knowledge.

The second step is to abstract from the behavior of objects the underlying principles of changing levels in the hierarchy. We expect several different principles to be uncovered, depending on the type of hierarchy used. We define hierarchies as abstract structures of object organization. Three types of hierarchies are identified: a filter hierarchy to divide the map space into regions, an aggregation hierarchy to express relationships between those regions, and a generalization hierarchy to express relations between classes of objects and their superclasses. We formalize the structure of a map and of a map series using the functional programming language Gofer with an object oriented paradigm.

1.4 Expected Results

The main results of this research are the formal models for a map and for a map series, called the map cube model (Fig. 1.4). This formal model incorporates a hierarchical data structure that supports and describes the behavior of map objects.

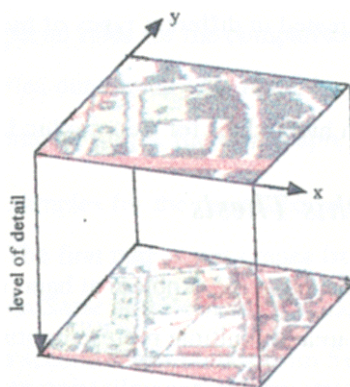


Figure 1.4: Map cube with two levels

The hierarchical data structure is either a tree or a forest. Additionally, we define constraints between data and constraints on the database. One of these constraints is for example, that the relationships between objects in map series need to be consistent, e.g., at scale1 object A is contained by object B, at scale2, object A must not be outside object B. The formal model is a multi-level data structure and thus builds a database with multiple representations.

We also define several types of hierarchies, their respective properties, and their methods to change levels. Hierarchies are defined according to the function that builds them. We identify what limitations these hierarchies present to the representation of spatial knowledge and consequently for which structures and tasks these hierarchies can be used. This is important for the design of databases for multiple representations.

1.5 Contribution and Intended Audience

The contribution of this thesis to spatial information science is a formal map model and a formal map series model. The suitability of these models is investigated in case studies of map series. We also contribute the definition of different types of hierarchies with their properties, their mechanisms and their application to a multi-scale database.

The intended audiences of this research are

- designers of spatial databases and Geographic Information Systems,
- researchers interested in multi-scale databases, especially cartographers,
- researchers from computer graphics interested in geographic applications,
- cognitive scientists interested in different types of hierarchies used to depict the environment, and
- researchers interested in hierarchies for Geographic Information Theory.

1.6 Organization of this Thesis

In the next chapter the background to the problem at hand is reviewed. We review what has already been done in the area of multiple representations, especially multi-scale representations. The area of cartographic generalization gives a background for the case studies. We then shortly review the pertinent literature in computer graphics and user interfaces. The last area of interest in the second chapter are hierarchical data structures.

We introduce our methods in chapter three. First we give the philosophical background to careful and systematic observation. The method is similar to the technique of reverse engineering. It is used here in the sense of observing exactly what happens and defining this formally. Graph theory is needed to specify the different types of hierarchies and the structure of the map series. We define the concepts of partition and refinement for hierarchical structures. GOFER provides the functional programming environment we need to implement the model of a map series.

Chapter four investigates on a theoretical level the phenomenon ‘hierarchy’. We first discuss why hierarchies are important for the modeling of multi-scale databases. We define a hierarchy mathematically and discuss three different types of hierarchies. Finally, we examine how these hierarchies can be combined.

In chapter five, we develop a theory on the structure of a map series. First we present a model of a map. We give an ontology of map elements and assign them to the objects of the model. A formal model of the map model is given in Gofer (section 5.1).

Then we derive a model for a series of maps, connecting several of the map models with the help of tree structures. This model is called the map cube model (section 5.2).

In chapter six, we describe in detail how we carry out the observations in the case studies. The first section explains the impact of the level of detail on map series and what this means for our case studies. The second section details how to compare map objects in general and lists special rules for the comparison of maps at different levels of detail. Section three specifies our strategy for matching map objects and gives an example how to build the container graph, the area graph, and the element graph.

Chapter seven shows examples for the construction of graphs. Our examples are drawn from two map series. The first map series comes from former East Germany, including the areas of Alsleben/Saale and Berneburg/Saale. Berneburg and Alsleben are small cities on the Saale. The second series comes from Switzerland, including the area of Murten. We present examples for the construction of the container graph, area graph, and element graph. Most study areas have irregularities, that made it impossible to build forests or trees as data structures.

Although there were irregularities in the map data, it was possible to build the graph structures as described in chapter five. The irregularities in the data resulted in deviations from the tree or forest structure in the graphs. The reason there were irregularities is that the map data is not consistent over the four levels of detail. We discuss the issue of consistency in section 8.1. Each of the components of the map cube model is analyzed and formalized in section 8.2 through 8.5. In section 8.6, we formalize the map cube model, describe the relations between the graphs in the model, and discuss the data structure resulting from our model. Section 8.7 summarizes this chapter.

Chapter nine first summarizes what we have done in this research. The results and the conclusions of our investigations are stated and limitations are discussed. Finally, we present our ideas on how to continue research on this topic. The appendix lists the complete code in Gofer of the map model and the map cube model. It also presents the example we used to check our model.

Chapter 2 - Background and Previous Work

The aim of this thesis is to define a hierarchical data structure for data at multiple levels of detail. One special application of this data structure is to facilitate an intelligent zooming function (Frank, 1982; Frank and Timpf, 1994; Liebermann, 1994; Timpf and Frank, 1995a). This chapter presents the background to data structures at multiple levels of detail, which are a special type of multiple representations. We discuss relevant aspects of multiple representations in section 2.1 and the background from cartography in section 2.2. Contributions from the field of computer graphics are discussed in section 2.3. Previous work in the area of hierarchical data structures is presented in section 2.4. The background to hierarchies in general is discussed in chapter 4.

2.1 Multiple Representations

The field of multiple representations is an important research topic in Geographic Information Systems (Buttenfield and Delotto, 1989). Humans naturally use in their minds several representations, even conflicting ones, of the world around them (Tversky, 1993). Multiple representations result from seeing the world from different angles, coming from different cultures (Frank, Campari, and Formentini, 1992; Campari and Frank, 1995) and backgrounds, being in various fields of study, or domains of expertise. For example, a biologist sees each living tree as a host to a multitude of small creatures, whereas the owner of the tree sees its value as wood once chopped down, or the ecologically minded politician sees this tree as a political asset to be preserved. Geographic Information Systems call for new modeling and reasoning methods to accommodate several perspectives.

A geographic information system serves many users. Thus, it seems only natural to store more than one representation of real-world objects in the information system. Logically there are two approaches to provide the user with several representations:

The first approach is maintaining a single database from which any other representation can be derived. This is only possible if the transformation from the stored database object to the desired output object can be accurately described,

formalized and implemented (Bruegger and Frank, 1989). However, this is rarely the case. The manual cartographic generalization is an analogue method based on this approach (c.f. section 2.2).

The second approach is storing multiple representations from a single real-world object in the database. This leads to the problem which representations to pick and how to store them appropriately in the database so that they can be accessed for the correct task. There are solutions from computer graphics that are based on this approach (c.f. section 2.3).

If the representations can only vary in their scale, as is the case with cartographic data, then there is a third approach. Multiple representations are stored at certain 'key scales' in the database. Those data sets can be enlarged or reduced in scale, but only within the range of scales permissible for the data set. The 'key scales' are dependent on the tasks to be performed with the data. This approach relies on data structures that are hierarchical, that means, that the objects are ordered depending on their scale. Hierarchical data structures are discussed in section 2.4.

The idea of a multi-scale (or as it is sometimes called scaleless) database is not new. Guptill (Guptill, 1989) speculated on a seamless and scaleless database, where users would be able to see the amount of information requested unhampered by map sheet boundaries. He also opted for an ability to transition from one level of detail to another appropriate to the scale of the task to be performed.

Another issue discussed in this context is the problem of maintaining consistency among the objects in a database with multiple representations. Users of a GIS expect that the answers to their queries do not change with a different representation of the data. Strategies to evaluate inconsistencies are the objective of research in that area (Egenhofer, Clementini, and DiFelice, 1994; Oosterom, 1997). This is especially interesting for the task of updating a database with multiple representation (Kilpelainen, 1994).

The problem of linking databases at different levels of detail becomes one of schema integration if the conceptual schemata of the databases differ (see for example,

(Devoegele, Parent, and Spaccapietra, 1997)). Links between datasets at different levels of detail can be represented by scale-transition relationships if the database schemata are the same and if the datasets are consistent (Devoegele and Raynal, 1995; Devoegele, Trevisan, and Raynal, 1996). This is an interesting research topic for the representation of cartographic knowledge.

Existing tools in GIS may be easy to adapt to multiple representations if we better understand the hierarchical natures of multi-level data and changing levels of resolution. This is important to make Geographic Information Systems more usable and more adaptable to the user's needs. It is necessary to provide multiple views on the world in a GIS - the users come from many different fields and have many different views.

2.2 Cartographic Generalization

Cartographic Generalization is the process of selecting and simplifying the representation of detail of a source map appropriate to the scale and the purpose of a target map. This graphic process corresponds to the fundamental human activity of abstracting and reducing complexity.

In manual cartographic generalization a source map is subjected to a number of operations that yield the target map at the target scale. These operations are (Hake, 1975):

- simplification,
- exaggeration,
- displacement,
- merging or amalgamation,
- selection or omission,
- classification or typification, including symbolization, and
- enhancement.

Most operators are only vaguely defined, so that authors may use different definitions for the same term or use different terms for the same definition (Rieger and Coulson, 1993).

Efforts to achieve automated cartographic generalization were successful for specific aspects (Staufenbiel, 1973; Freeman, 1991; Powitz, 1993; Barrault, 1995), but no complete solutions are known (Brassel and Weibel, 1988; Bittenfield and McMaster, 1991), nor are there any expected within the immediate future. Mueller et al. give a comprehensive account of the current research trends (Mueller, Lagrange, and Weibel, 1995).

In order to render a complex and holistic process such as cartographic generalization amenable to automation, conceptual frameworks need to be developed. Brassel and Weibel (Brassel and Weibel, 1988) proposed a conceptual framework that identifies the major steps of the manual generalization process and transposes these concepts into the digital realm. The source database is first subjected to a process termed *structure recognition*. This step aims at the identification of objects and aggregate objects, their spatial and semantic relations and their relative importance. The second step is to define the relevant generalization processes in the *process recognition* phase. This involves the identification of the types of data modifications and the parameters controlling these procedures. Next is the *process modeling*, which compiles rules and procedures from a process library. Digital generalization takes place with the *process execution*, where the rules and procedures are applied to the source database in order to create the generalized target database. The last process, the *data display* converts the target database into a fully symbolized target map.

One approach for structuring cartographic data is to consider cartography as a language with its own syntax and vocabulary (Robinson and Petchenik, 1976; Youngmann, 1977; Palmer and Frank, 1988). The objects are combined from a graphical vocabulary, which provides the atoms for graphical communication (Bertin, 1967; Schlichtmann, 1985; Mackinlay, 1986; Head, 1991).

The model by Brassel and Weibel (1988) was extended by McMaster and Shea (McMaster and Shea, 1992). This model decomposes the generalization process into three operational areas: a consideration of the philosophical objectives *why* to generalize, a cartometric evaluation of *when* to generalize, and the selection of appropriate spatial and attribute transformations which provide techniques on *how* to generalize. The area

on when to generalize is equivalent in scope to the structure and process recognition in the model of Brassel and Weibel (1988). In the area of how to generalize a list of twelve operators is proposed, similar to the list of operators given by Hake (1975). Ten of these operators perform spatial transformations and two operators are for attribute transformations.

Generalization operators have been defined and implemented (Lee, 1992; Bundy, Jones, and Furse, 1994), but a human operator is still needed for guidance each time a new map is derived. This insight has led to a new approach called *amplified intelligence* (Weibel, 1991). This approach builds on the integration of algorithmic and knowledge-based techniques. Algorithmic techniques serve the purpose of implementing tasks for which sufficiently accurate objectives can be defined. Knowledge-based methods are used to encode expert knowledge into the system. Knowledge-based methods still suffer from the lack of formalized cartographic knowledge and the problems encountered acquiring it (Weibel, Keller, and Reichenbacher, 1995).

2.3 User Interfaces and Computer Graphics

In computer graphics the notion of level of detail is well known. Especially in the rendering processes of virtual worlds it is necessary to derive simpler representations of objects to minimize rendering time. The process is called polygonal simplification (Erickson, 1996) and can be compared to the approaches of geometric generalization of terrain representations (Weibel and Heller, 1991; Misund, 1996).

However, this approach is not applicable to cartographic data (with the exception of terrain modeling), because only single objects are simplified. In maps, single objects interact with each other and a change in one object entails a change in the surrounding objects. The second objection to the use of polygonal simplification is that in cartography each symbol and line on the map has a very special meaning, whereas in computer graphics lines may be expendable. An extension of polygonal simplification is the constraint polygonal simplification. Indeed, models and implementations of this type of simplification have been proposed in a cartographic context (Bundy, 1994).

“One of the best ways to comprehend a large complicated information structure is to form multiple simpler structures each highlighting different aspects of the original structure” (p.336 (Mukherjea, Foley, and Hudson, 1995)). Mukherjea et al. use this method to visualize complex hypermedia networks with the help of multiple hierarchical views. Each of the views is independent of the others and each gives a different view on the information structure.

Displays should be more adaptable to users needs and their tasks instead of creating and presenting a static view of the data (Lindholm and Sarjakoski, 1992; Lindholm and Sarjakoski, 1994). Tools for handling multi-scale data in displays are even scarcer than multi-level data structures (Timpf and Devogele, 1997). Among these tools, changing scale or ‘zooming’ is crucial for the handling and analysis of multi-level data. Zooming in the technical sense of the word means 'seeing more or less detail'. Current zooming functions blow up and shrink the objects to display. This means, that no additional or less information is given on the topic of interest. It may also be that the system changes to a radically different representation when a certain scale range for a single representation has passed. This behavior of the system leads to stress on the user - the user loses the orientation, the focus, or even the observed object when the representation changes abruptly.

In user interfaces, the context and focus zoom (Robertson and Mackinlay, 1993) is well researched. Similar methods have been applied to geographical data (Liebermann, 1994; Stone, Fishkin, and Bier, 1994; Lokuge and Ishizaki, 1995), but the existing systems only display data at multiple levels. The connection between the levels is purely visual and cannot be used to analyze space, as is the prerequisite for Geographic Information Systems.

2.4 Hierarchical Data Structures

Different levels of detail are needed when dealing with spatial data. Consequently, appropriate data structures and tools are required for a multi-level database. Multi-scale or multi-level data structures already exist on a low level, e.g., quadtrees (Samet, 1989) or hierarchical triangulated networks (Floriani and Puppo, 1992; Dutton, 1997), and

topological structures (Bruegger and Frank, 1989). Those structures show aspects of hierarchically organized data but only for one type of hierarchies, namely for aggregation hierarchies.

Research in the area of different types of hierarchies and their combination is needed to explain the complex and varying structures of hierarchical spatial data. It is necessary to separate the processes that are innate to changing levels of detail (e.g., switch to a higher level of detail) from their application to a special function (e.g., zooming in). This situation can be compared to the relationship between functions and higher order functions in mathematical category theory (Asperti and Longo, 1991; Walters, 1991): higher order functions explain what is similar in a set of functions. The essence of the behavior of the functions is taken and abstractly described in a higher-order function (Frank, 1997b).

Ballard applied the hierarchical structure to lines in strip trees (Ballard, 1981). Plazanet also structures only the lines hierarchically (Plazanet, 1995). The pyramid structures used in image processing (Rosenfeld *et al.*, 1982) apply the hierarchical structure to image pixels.

Van Oosterom (Oosterom, 1991) proposed to use an R-tree for storage of less detailed objects. The R-tree is a data structures for spatial access (Guttman, 1984). For this structure van Oosterom proposed an 'importance' characteristic. The spatial allocation of the objects is assured by encoding a minimal bounding rectangle. Our approach is broader, in that we model the underlying conceptual organization of cartographic objects in a map series and are not yet concerned with implementation methods.

Chapter 3 - Methods of Investigation and Formal Tools

This chapter explains the methods we use in this work to accomplish our goal. Our goal is to detect and formalize the structure of elements in map series. This structure is hierarchical, because there are several levels of map elements ordered by detail. It is necessary for the formalization to understand the processes that are going on when changing the level of detail.

The first step is to observe systematically the behavior of map elements over a range of map scales. The next step is to describe this behavior informally either in figures or in functions. A part of this method is also known as reverse engineering. Our method of observation is described in section one of this chapter.

The next step is to extract information about the structure of elements in a map series. We define the identified structures with the help of graph theory. Graphs are very well suited to describe hierarchical structures. The part of graph theory we need is introduced in the second section. We also introduce the concept of partition and refinement of a partition in section three.

As a last step towards formalizing the data structure of map series we implement the structures and their operations in a functional programming language. The programming environment and the principles of functional programming are explained in the last section of this chapter.

3.1 Observation Method

In this work, we observe systematically the behavior of map elements over several map scales. However, an objective and detached observation of phenomena is not possible, since each observer favors certain aspects over others and thus cannot be truly objective (Chalmers, 1994). The theory in the observer's mind guides (sometimes unconsciously) the setup of the experiment and the process of observing. In other words, observation is theory dependent.

The method of observing a system and detecting the rules of the system from its components and its behavior is called abduction (Kakas, Kowalski, and Toni, 1993). Abduction is similar to the method of induction. Both are reductive methods and are in contrast to the deductive method (Tarski, 1995). In natural sciences reductive methods are used to attain a deductive system (see Fig. 3.1b). Our goal is to arrive at a deductive system that can be formalized.

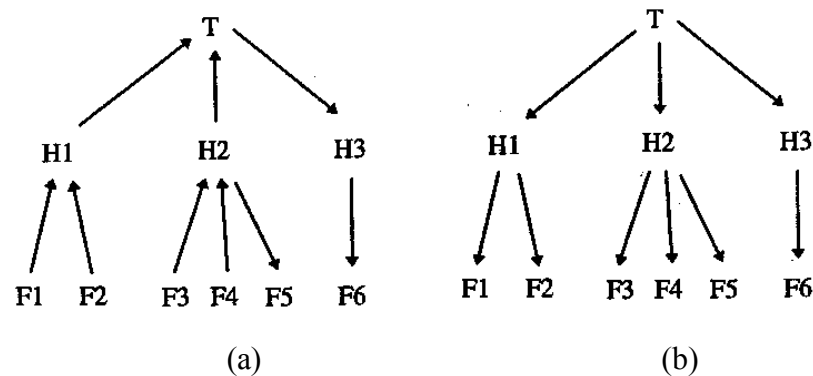


Fig. 3.1: Reduction (a) and deduction (b) in a logical system

Figure 3.1a shows the flow of reasoning about the system while observing, whereas Figure 3.1b shows the logical deduction of facts once observation has been completed. F denotes the facts that can be observed, H denotes the hypotheses valid in the system, and T denotes the underlying theory. The figure demonstrates that the theory plays a central role in the method of reduction.

Reverse engineering (Rekoff, 1985) is a method similar to abduction. It is the process of analyzing a system in order to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction (Marciniak, 1994). Rekoff defines Reverse Engineering as "the act of creating a set of specifications for a piece of hardware by someone other than the original designers, primarily based upon analyzing and dimensioning a specimen or collection of specimens" (Rekoff, 1985). Originally a method to analyze a competitor's hardware, today, reverse engineering is used to create software specifications (Hall, 1992; Lano and Haughton, 1994; Hoschek and Dankwort, 1996), and to maintain existing software (Bowen, Breuer, and Lano, 1993).

As such, reverse engineering is a technique for the acquisition of knowledge: deducing rules by observing the results of these rules. Weibel et al. (1995) mentioned this technique among others as necessary for the acquisition of cartographic knowledge for automated map generalization.

The process of reverse engineering according to Rekoff (1985) is a sequence of five steps. First, assimilate existing data. Second, identify the elements of the system. Third, disassemble the item under consideration, then analyze and test its functions. Finally, complete a documentation of the system. When disassembling the complete piece, smaller subparts may appear that are processed in the same manner as the whole. This process is repeated until the smallest disassembled part cannot be further subdivided.

Our goal is to observe the processes going on when map elements change their level of detail. The system under consideration is a series of maps containing map elements. We first assimilate existing data using literature on hierarchical structures and cartographic generalization. Next, we identify the elements under consideration in the theory detailed in chapter five. Instead of disassembling the item under consideration we promote a theory of how its parts fit together. With the help of graph theory (see next section) we model the structures of the map elements over several levels of detail, i.e., the hierarchies of map elements. We define types of hierarchies according to the functions between map elements on different levels of detail.

Next, we devise experiments to demonstrate that our theory is valid in practice. The observation plan describes in detail how we study the cases taken from two different map series. The analysis describes functions between elements and between levels of detail. We specify a formal model to describe the structure of map elements in a functional programming language (see section 3.3). The formal model and the theory are the results of our investigation.

3.2 Graph Theory

Order relations and hierarchies are an especially important structure in this thesis. The mathematical tools to model order relations are directed graphs. Trees, a special type of graphs, serve as natural representation for any type of hierarchically organized data. The

following fundamental definitions of graphs and trees are based on (Wilson and Watkins, 1990; Laurini and Thompson, 1991; Piff, 1991; Gueting, 1994). We only give those definitions that have a bearing on our investigations. We define graph, directed graph, forest, and tree.

A *graph* is a set of points (vertices) in a mathematical space, which are interconnected by a set of lines (edges). The graph is said to be finite if the number of vertices is finite and if the number of edges is finite. An edge is specified by the two vertices, called its *end-points*, that it connects. The edge is *incident* with the vertex if the vertex is an end-point of the edge. Two vertices are said to be *adjacent* to another if they have a common edge. Two edges are *adjacent* if they have a common vertex. The *degree of a vertex* is the number of edges incident with the vertex.

A *simple graph* is a graph that contains no self-loops or parallel edges. A *multi-graph* is a graph that contains parallel edges but no self-loops. A *path* from one vertex to another is a sequence of alternating vertices and edges so that every edge is incident with the last vertex and the next vertex. In a simple graph, the path can more simply be specified by the sequence of vertices. Two vertices are *connected*, if there is a path from one to the other.

A graph that has directions assigned to its edges is called a *directed graph* or *digraph*. In a diagram the direction of each edge is represented by an arrow. In the description of *directed edges* the order of its end-points is significant. The edge is assumed to be directed from the first vertex to the second vertex (e.g., $e_1 = (v_2, v_1)$). The graph may or may not contain the directed edge $e = (v_1, v_2)$. The directed edge is said to be *incident from vertex 2 to vertex 1*. The number of edges incident from a vertex is called the *out-degree* of the vertex. Correspondingly, the number of edges incident to a vertex is called the *in-degree* of the vertex.

Every digraph has an underlying undirected simple graph that can be obtained by deleting the directions of the edges. A path in a simple undirected graph is a sequence of vertices and edges. In a digraph there are two possibilities to define a path: If the edges connecting the vertices are incident from the first vertex to the following vertex, the path is said to be *directed*. Otherwise it is an *undirected* path. Two vertices v_1 and v_2 are *strongly*

connected if there is a directed path from v_1 to v_2 *and* a directed path from v_2 to v_1 . If v_1 and v_2 are connected in the corresponding undirected graph, then v_1 and v_2 are *weakly connected*.

An *acyclic graph* is a graph that contains no cycles or, to express this fact with the help of vertices, in an acyclic graph there is at most one path between any two vertices of the graph (Fig. 3.2).

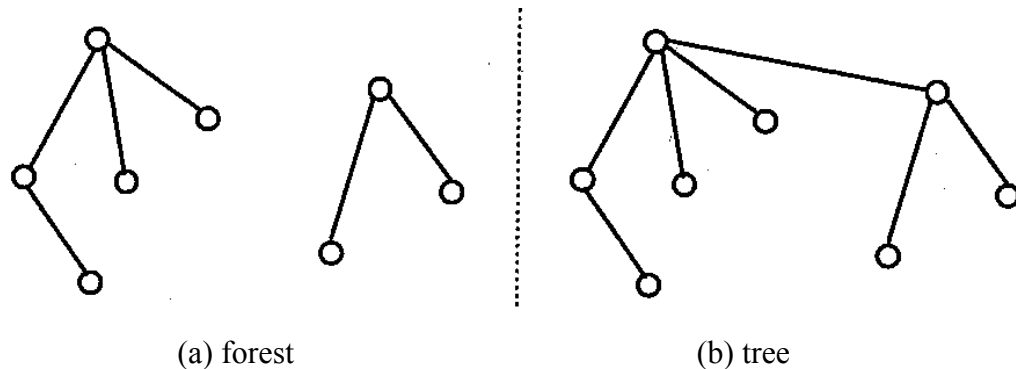


Fig. 3.2: Acyclic Graph

An acyclic graph may be non-connected (see Fig. 3.2a), it is then called a forest. If it is connected it is called a tree (Fig. 3.2b). A graph is a tree if and only if for each pair of vertices in the graph there exists a unique path in the graph joining those two vertices. This definition is equivalent to the following: A connected graph is a tree if and only if the number of edges is equal to the number of vertices minus one (Perl, 1981).

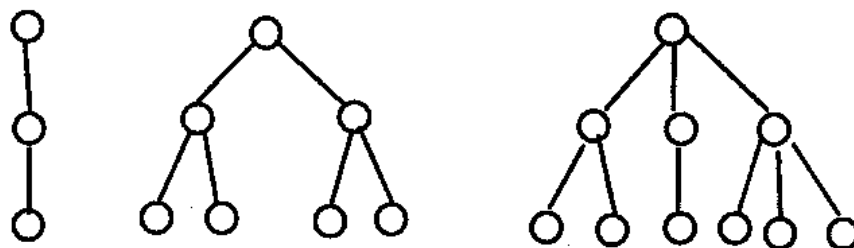


Fig. 3.3: Several types of trees

In figure 3.3, the first and second of these trees are binary trees: they have at most two subtrees. The left tree is also called a minimal graph or path graph. The right tree is a general tree. A general tree can be recursively represented as a node with a list of its subtrees: $\text{Tree} = \text{Node} [\text{Tree}]$. A binary tree (in the middle) is usually represented as a node and its left and right subtrees: $\text{Btree} = \text{Node} (\text{LeftTree}, \text{RightTree})$.

3.3 Partition and Refinement

Partitions have been identified as a central spatial concept to organize our perception and understanding of space (Frank, 1990). We use the notion of partition in the context of graphs for the representation of spatial objects. It is also possible to partition attribute values as in the work on categorical coverages (Volta and Egenhofer, 1993; Frank, 1997a). The newest approach to a rigorous definition of a partition of space is given in (Erwig and Schneider, 1997).

Let $P = \{A_i\}$ be a set of nonempty subsets of A . The set P is called a partition of A if

- (a) $A_i \text{ overlap } A_j = 0$, whenever $i \neq j$
- (b) Sum of the $A_i = A$.

Table 3.1: Definition of a partition

A spatial partition is a division of space, where the borders of the regions do not cross and the sum of all regions is the complete space. That is, the regions are jointly exhaustive and pairwise disjoint. This is the same concept as the tiling of the bathroom. Bathroom tiles are usually regularly shaped, whereas regions have irregular shapes. Table 3.1 gives a mathematical definition of a partition.

A sequence of partitions over a several levels of detail results in a hierarchy of partitions (Frank, 1983). Graphs that build hierarchical partitions have some requirements:

- the edges of a partition do not cross
- the edges of all partitions do not cross
- the edges of one partition are a part of a cycle
- the edges of a higher hierarchical partition are also the edges of a lower hierarchical partition

In a hierarchy of partition, the partition on level n is a refinement of the partition on level $n-1$, which is a refinement of the partition on level $n-2$, etc. A partition $P' = \{A_i'\}$ is a refinement of partition $P = \{A_j\}$ if every A_i' is a subset of some A_j . This property ensures that topological relationships of containment and neighborhood are consistent over all levels of detail.

3.4 Formalization with a functional language

The data structure we propose is a conceptual model of a map series. The data structure must be implemented to be applicable to a Geographic Information System. We could derive the implementation directly from the conceptual model. With increasing complexity and size of the conceptual model, the step from concept to implementation needs to be controlled through a formal model (Fig. 3.4).



Fig. 3.4: Progress for modeling of complex systems

A formal model captures the semantics and the behavior of the objects and the operations in the model (Kuhn, 1995). It is independent of any implementation issues such as brand of computer, specific programming language, run time considerations, or efficiency of the data structure. A formal model is created by describing the conceptual model with a formal language.

In this section we present a formal language and its properties. The definition of the conceptual model in a formal language should be consistent with the conceptual model. It should also be complete in the sense that each property of the conceptual model is also stated in the formal model. It should allow the designer to examine whether the original ideas and models are correctly represented and whether they correspond to the objectives and the expectations of the model (Ehrich, Gogolla, and Lipeck, 1989). We show that the functional language Gofer has these properties.

3.4.1 Model and Language requirements

There are different ways to describe the world around us. One method is speaking, i.e. using a natural language. Natural languages are not suited as formal languages (Mark, 1993; Kuhn, 1994), because their words often have more than one meaning. In a formal language, words and symbols have only one predetermined meaning (Bochenski, 1954).

In a second approach, the formal model could be a purely mathematical model. In mathematics, symbols have precisely one predetermined meaning. The semantics of

mathematics is known and the formal model can easily be transformed into a program. Using standard mathematics works as long as the world that is described is not too complex. Then, the formulae become very complex, hard to handle and to understand. According to the law of parsimony or Occam's razor the simplest model should be used, that offers the same power of explanation than the more complicated models (Bochenski, 1954).

Third, a simple model describes the world through its properties. Languages used to describe properties and static situations are first-order languages. First order predicate calculus is used in relational databases to describe the properties of their entities. However, defining objects through properties has problems rooted in human cognition (Rosch, 1973; Rosch, 1978b; Lakoff, 1987b; Lakoff, 1987a). For example, a table is an object that has a flat surface and four legs. This is the prototype of a table. But a block of wood can also be used as a table and will be recognized as one (by a human). A better solution is to define a table through its behavior: its affords that things can be put on top of it and stay there (Gibson, 1979; Kuhn and Blumenthal, 1996). But first order languages cannot express behavior and changes of behavior (Kuhn, 1995).

The fourth method is to describe objects by their behavior. Modeling behavior is best done with an algebraic approach. Describing changes of behavior and dynamic situations demands higher-order functions (Kuhn, 1995; Frank, 1997b). Higher order functions allow to distinguish between the operations on the data and the operations on the data structure. Modeling behavior can be realized with, e.g., algebraic specifications (Guttag and Horning, 1978; Liskov and Guttag, 1986; Timpf *et al.*, 1992). They are based on a universal algebra (Birkhoff, 1945; Birkhoff and Lipson, 1970). However, algebraic specifications are hard to read and check, and they are not executable.

In summary, we are looking for a formal language that is algebraic, allows higher-order functions, and is executable. The next section shows that the functional programming environment Gofer fulfills these requirements.

3.4.2 The functional programming language GOFER

Gofer (Jones, 1991; Thiemann, 1994) is a functional programming language from the ML family, similar to Haskell (Hudak *et al.*, 1992). It is very close to algebraic specifications and therefore beneficial to formal modeling, theorem proving and fast prototyping. Gofer has a class system that allows multiple type parameters. Each class is an algebra, similar to the sorts (Liskov and Guttag, 1986) or traits (Guttag, Horowitz, and Musser, 1978) of algebraic specifications. Gofer allows multiple inheritance in the framework of ‘parametric polymorphism’ (Cardelli and Wegner, 1985). The language is strictly without side effects and code is therefore referentially transparent. An introduction to the language and how it is used can be found in (Frank, 1997c). We will just highlight a few interesting features of Gofer in this section.

Gofer is a strongly typed language, the Hindley/Milner type system is used and extended (Jones, 1995). It assigns a type to every object in the language. The type inference mechanism checks the consistency of the types in any expression automatically. This is similar to checking the units (meters, centimeters) when calculating in physics. The type checking system makes Gofer more powerful than imperative languages. If the types are correct, the result is much more likely to be correct than if the types were not correct. The type inference also prevents incomprehensible run-time errors due to wrong types (Bunemann and Nikhil, 1984).

Gofer is a functional programming language. Every expression is a function, constants are 0-ary functions. Functions can have functions as parameters. Those functions are higher-order functions. Some programming languages have means to allow functions and even functions as parameters (e.g., Pascal, and C). But this functionality is not fully embedded in the structure of the language and therefore often not cleanly implemented. Specifically they cannot deal with the complex types of functions in type checking.

Gofer’s evaluation mechanism is lazy, this means each expression is only evaluated if it is needed. Therefore, it is possible to express knowledge about objects that are infinite or cannot be computed. It must be ensured that the infinite or non-computable expressions are never evaluated.

Gofer is computationally complete. Important for programs are also correctness and termination. Type checking helps to ensure correctness but does not imply it. Termination is critical, because it might be necessary to operate with infinite sets.

Gofer is a class-based language. A class has a name, basic constructors, basic observers and axioms. We show the class based structure with an example of a general tree. A general tree can be constructed with a node and a list of subtrees (see section 3.2). This data structure is also called a Rose Tree. Table 3.2 shows an example for a tree of integers. In this tree, *t1* is the root node and *t4*, *t5*, and *t6* are the leaves of the tree. The depth and the width of the tree are both three. *T1* is defined as a node with name *n1* and a list of two subtrees, namely *t2* and *t3*. The node *n1* is an Integer. For this reason, the structure is a tree of integers. In line 1 of table 3.1, the *T* is called the type constructor. The type checking system recognizes that after a *T* always a type *a* and a list of type *Tree a* must follow. In a sense the *T* constructs that particular data type. The type *Int* is introduced for the type parameter *a* in the first line of table 3.2. The correct type for *t1* is then: `Tree Int = T Int [Tree Int]`.

```

data Tree a = T a [Tree a]
t1,t2,t3,t4,t5,t6 :: Tree
Int
t1 = T n1 [t2,t3]
t2 = T n2 [t4,t5]
t3 = T n3 [t6]
t4 = T n4 []
t5 = T n5 []
t6 = T n6 []

```

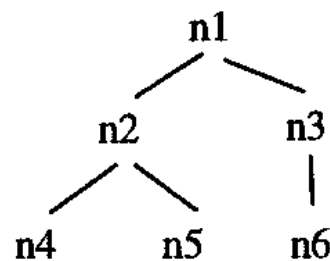


Table 3.2: Example for a tree of integers

The corresponding class of operations to the data type tree is the class *TreeS* (Table 3.3, line 2). *TreeS* has two parameters: one for the data type *Tree* itself and one for the type parameter of *Tree*. The symbol combination ‘*t a*’ always means a particular tree of type *a*, where tree is constructed as defined in line 1. The expression before the implication means that equality must be defined for the type *a*. The expression is called context.

The basic constructor of the class *TreeS* is *createTree* (line3). This function takes a type parameter and returns a tree. How this is done for this particular representation can be seen in line 7: the variable *n1* (of type *a*) is required and a tree is constructed with the

type constructor T , the variable $n1$ as node, and an empty list of subtrees. The resulting symbol combination ‘ $T\ n1\ []$ ’ is of type *Tree a*.

The basic observer *node* takes a tree and returns an object of type a (line 4). Line 8 shows, that a node of an object constructed as ‘ $T\ n1\ _$ ’ (the underbar signifies a wildcard) is always the variable $n1$ of type a . The basic observer *subtrees* takes a tree and returns a list of subtrees (line 5). Line 10 shows that this function always returns the second part of the construction ‘ $T\ _ \ ts$ ’, which is in fact the list of trees as defined in line 1.

```

1  data Tree a = T a [Tree a]
2  class (Eq a) => TreeS t a where
3      createTree :: a -> t a
4      node      :: t a -> a           --root node of the tree
5      subtrees  :: t a -> [t a]     --the immediate children
7  instance (Eq (Tree a)) => TreeS Tree a where
8      createTree n1 = T n1 []
9      node (T n1 _) = n1
10     subtrees (T _ ts) = ts
12 instance (Eq a, Eq [Tree a]) => Eq (Tree a) where
13     (==) (T n1 t1) (T n2 t2) = ((n1==n2) && (t1==t2))

```

Table 3.3: Gofer code for a rose tree

Lines 7 through 10 are called the instantiation of the class *TreeS* for the data type *Tree a*. The context stipulates that the equality for *Tree a* is defined. We define the equality for *Tree a* in lines 12 and 13. The equality operator belongs to a class *Eq* - a class of all things that can be compared for equality. In line 13 the equality operator is defined for the data type *Tree a*: two trees are equal if their nodes are equal and if their subtrees are equal. This definition demands that the equality for nodes and for lists of trees is defined. This is exactly what the context stipulates.

3.5 Summary

In this chapter we introduced the methods we use to study objects in map series and to formalize the underlying structures. Since every observation is theory dependent, we first need to outline our theory on the structure of maps and map series (see chapter 5). On the basis of the theory we observe the objects in map series (see chapters 6 and 7). This method is called abduction, which is a reductive method similar to induction.

In computer science an application of the method of abduction is also known as reverse engineering. Reverse Engineering is the process of analyzing a subject system in order to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction. We analyze the system of map series with the goal to build a formal model of this system at a higher level of abstraction.

The mathematics for hierarchical structures is found in graph theory. We gave a short introduction to graph theory as we use it in this thesis. We explain in detail the directed graph and the tree, because they are especially suited to describe hierarchical structures.

Finally, we formalize the structures and operations with the functional programming language Gofer. The formalization with Gofer has some advantages over previous approaches. Gofer can be compared to algebraic specifications that are executable. It supports higher order functions, that we need for the hierarchical structures. The type inference system insures that the results of the formalization are type-consistent.

Chapter 4 - Hierarchies

“One of the best ways to comprehend a large complicated information structure is to form multiple simpler structures each highlighting different aspects of the original structure.”

(Mukherjea, Foley, and Hudson, 1995)

This chapter investigates on a theoretical level the phenomenon ‘hierarchy’. We first discuss why hierarchies are important for the modeling of multi-scale databases. We define a hierarchy mathematically and discuss three different types of hierarchies. Finally we examine how these hierarchies can be combined.

Current Geographic Information Systems (GIS) lack the structures, tools, and operations to handle multiple representations and especially representations with multiple levels of detail (Timpf and Devogele, 1997). One reason for this is that very little is known about using hierarchies for the description of ordered levels of detail. Hierarchies appear in many different spatial contexts, for example, road networks (Car, 1996), political subdivisions, land-use classes (Volta and Egenhofer, 1993), water sheds (Fisher, 1996). But hierarchies appear also in non-spatial situations such as organizational hierarchies. It is the intrinsic nature of hierarchies that has not yet received much attention and this lies at the basis of the problems with hierarchies in GIS.

Hierarchies have many advantages usable for GIS. They provably reduce processing time (Pattee, 1973; Car, 1996). They increase the stability of any system (Pattee, 1973). They break down the task into manageable portions (Timpf *et al.*, 1992), enhancing the potential for parallel processing. Inferences can be refined depending on the type and level of query (Fotheringham, 1992; Mainguenaud and Simatic, 1992; Glasgow, 1995; Rigaux and Scholl, 1995; Frank, 1996; Timpf and Frank, 1997b).

Hierarchies and the principles that build them are interesting as knowledge acquisition methods (Weibel, Keller, and Reichenbacher, 1995). We use hierarchies and their principles as data representation methods. The ordering variable in all these hierarchies is scale, i.e., the level of detail of the map elements.

Our work is based on the assumption that many individual object-systems with simple behavior form a whole object-system with complex behavior. For example, a map

series is a complex object-system composed from many maps. A map in its turn is also composed of many elements. It is also assumed that the organization of the individual objects is hierarchical; the complex object being on top of the hierarchy. This assumption can be found in diverse fields such as systems theory (Koestler, 1967; Mesarovic, Macko, and Takahara, 1970; Pattee, 1973), ecology (Ahl and Allen, 1996), biological organisms (Braitenberg, 1984), software specification (Borgida, Mylopoulos, and Wong, 1984; Brodie, Mylopoulos, and Schmidt, 1984), landscape planning (Alexander, Ishikawa, and Silverstein, 1977), and reverse engineering (Rekoff, 1985).

Literature on a theory of hierarchies is scarce. There are two areas in hierarchy research that contribute to the questions examined in this thesis. In cognitive science, hierarchies are examined as a way how humans represent the world. In system theory the observation is made that most biological systems are hierarchically structured and this structure is applied to non-biological systems.

4.1 Hierarchies in Cognitive Science

Hierarchies are fundamental to human cognition (Langacker, 1987) and have been studied in cognitive science (Hirtle and Jonides, 1985; Freksa, 1991; Fotheringham, 1992; Timpf *et al.*, 1992; Kuipers, 1993; Car, 1996). Humans arrange information hierarchically and use hierarchical methods for reasoning (Stevens and Coupe, 1978; Hirtle and Jonides, 1985; Hirtle, 1995). '*Hierarchization*' is one of the major conceptual mechanism to model the world. Other mechanisms to express structures of the world are, for example, using prototypes (Rosch, 1978a) or relations (Egenhofer and Franzosa, 1991; Mark and Egenhofer, 1992). The idea is to deduce knowledge at the highest (coarsest) level of detail in order to reduce the amount of facts taken into consideration. Too much detail in thinking means long and inefficient reasoning.

Humans construct hierarchies by abstracting information, that means by building ordered classes of information. For example, when seeing a tree, one recognizes that the brown long trunk in combination with the green leaves fall into the object category 'tree'. In this example, the spatial arrangement of the entity from the class 'leaves' and the entity from the class 'trunk' suggests that the aggregate entity belongs to the class 'tree'. The

corresponding hierarchy is an aggregation hierarchy. Hierarchies of entities in the geographic world result in multiple representations within a spatial database.

What makes the conceptual task of modeling hierarchies so difficult is the fact that humans are able to switch between different types of hierarchies (attributes, classes, instances, tasks etc.). Humans do not even notice that they are using different types of hierarchies simultaneously. We assume that this fact is the biggest impediment to the investigation of representations with multiple levels of detail.

4.2 Hierarchies in Systems Theory

In systems theory the observation is made that most biological systems are hierarchically structured and this structure is applied to non-biological systems (Ahl and Allen, 1996). This approach was especially taken in the description of complex dynamic systems (Mesarovic, Macko, and Takahara, 1970; Pattee, 1973).

Simon (Simon, 1981) states that complex systems are usually hierarchically structured. There are several advantages to a hierarchical structure. The system has a number of interrelated subsystems that are themselves hierarchically structured. Each subsystem is a stable intermediate form and can function without 'help' from the complex structure. All forms in evolution are stable intermediate forms.

Koestler (Koestler, 1967) explains the principle with the parable of the watchmakers. One watchmaker assembles a watch from scratch with all single parts on the table. The other watchmaker assembles smaller subparts of the watch and then connects these subparts. The second watchmaker is much faster in assembling a watch. This is possible in all systems that are nearly decomposable, which means that the relations inside a subsystem are stronger than the interrelations between subsystems.

Mesarovic (Mesarovic, Macko, and Takahara, 1970) introduces three types of hierarchical systems distinguished by the types of levels: the level of abstraction, the level of decision complexity and the organizational level. The level of abstraction in a hierarchical system expresses the fact that some objects in the system may be more abstract than others, and the level imposes an order on the objects. The level of decision complexity creates a hierarchy of decision layers, meaning that decisions in a system are broken down

to decisions of subsystems. The original decision is made with a certain margin for error, because not all subsystems may be able to decide. The organizational level creates an organizational hierarchies as in a human formal hierarchy.

4.3 A Definition of Hierarchy

A hierarchy is an ordered structure. Order can be established between individuals or between classes of individuals. The ordering function may be any function defining a partial order:

$$v_1 \leq v_2 \text{ iff } \exists f : v_2 = f \{v_i\}; v_1 \in \{v_i\}$$

A vertex v_1 in a graph is on a lower level than v_2 if and only if there exists a function f , such that v_2 can be calculated by applying f to a set of v_i of which v_1 is a member. The function f must be reflexive, antisymmetric and transitive (Gill, 1976).

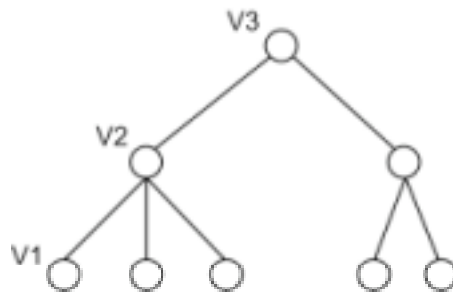


Fig. 4.1: Partial ordering: tree

The partial ordering can be depicted as a tree with the vertices denoting the individuals of the domain and the edges representing the ordering function between individuals (Fig. 4.1).

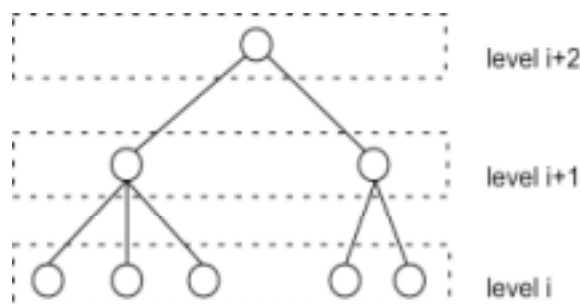


Fig. 4.2: Levels in a hierarchy

The notion of levels is introduced through the idea that vertices at the same depth of the tree belong to the same level of the hierarchy. Thus, there are as many levels in the

hierarchy as the tree is deep. The highest level is the most abstract level of the hierarchy, the lowest level is the most detailed level (Fig. 4.2).

A hierarchy is intricately linked to the idea of levels. Individuals on the same level share a common property, e.g. individuals have the same power in the company, or map objects have the same level of detail. The order of the levels is always total, although the order between individuals in the hierarchy is only partial.

Hierarchies can be distinguished by the way they are constructed. The distinction lies in the ordering function between individuals or classes of individuals. The most common function to build a hierarchy is the aggregation function. Classes of individuals are aggregated because they share a common attribute. In the next section we investigate several distinct types of hierarchies.

A hierarchy is an ordered structure with an total order between its levels, but with a partial order between individuals or classes of individuals. The structure has sub-structures that are also hierarchies. The function that establishes the order must be reflexive, antisymmetric, and transitive.

4.4 Types of Hierarchies

Hierarchies are formed through abstraction, i.e., through factoring out commonalities in the description of several concepts into the description of a more general concept (Borgida, Mylopoulos, and Wong, 1984). In software development three abstraction mechanisms have been identified:

- aggregation/decomposition
- generalization/specialization
- classification/instantiation.

The abstraction mechanism of classification is a prerequisite for all other abstraction mechanisms. It is necessary to classify objects before applying operations like aggregate, generalize, or filter to them.

4.4.1 Aggregating

We call the type of hierarchy that is constructed by aggregating sets of individuals to single individuals *aggregation hierarchy*. Other names like nested hierarchy (Ahl and Allen, 1996) or container hierarchy (Timpf and Frank, 1997a) are also common. The possibility of aggregation depends solely on a certain attribute of the individuals. For example, containers can be aggregated to form larger containers if they are spatially adjacent. The aggregation hierarchy is the most common type of hierarchy.

$$v_1 \leq v_2 \text{ iff } \exists f_{\text{agg}} \text{ such that } v_2 = f_{\text{agg}} \{v_i\}; v_1 \in \{v_i\}.$$

The aggregation function maps a set of individuals to a single individual, or it maps a set of classes of individuals to a single class of individuals. The end of the aggregation is reached if there is only one individual or class of individuals left as argument of the function. The type of the individuals is the same throughout the hierarchy. Only individuals of the same type can be aggregated. Fig. 4.1 shows a structure that represents an aggregation hierarchy.

4.4.2 Generalizing

The *generalization hierarchy* defines classes as more generic the higher the level of the class in the hierarchy. E.g., houses and industrial buildings all belong to the generic class ‘building’, or local road, city streets, and highways all belong to the generic class ‘roads’. This type of hierarchy has also been called classification hierarchy by Molenaar (1993).

The generalization hierarchy relates a class to a superclass. V1, W1, and Z1 denote classes in figure 4.3. Generalization is defined as forming a new concept by leaving out properties to an existing concept. The inverse function of specialization forms a new concept by adding properties to an existing concept (Parsons and Wand, 1997). This hierarchy is a non nested hierarchy in contrast to the aggregation hierarchy, which is a nested hierarchy.

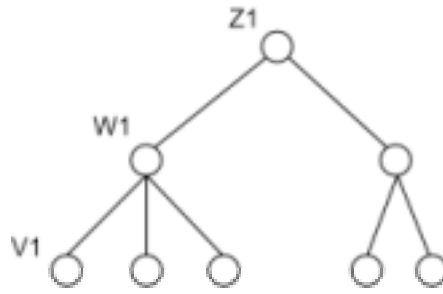


Fig. 4.3: Visualization of a generalization hierarchy

$$V_1 \leq W_1 \text{ iff } \exists f_{\text{gen}} \text{ such that } W_1 = f_{\text{gen}} V_1.$$

The rules to determine which class belongs to which generic class are predetermined. The generalization function states explicitly which classes generalize to which generic classes. This can be visualized as a hierarchy of classes with the most generic classes at the top level. Fig. 4.3 shows a generalization hierarchy with a single most generic class Z1 at the top level. Individuals change their type when generalized, e.g., individuals of class V1 become individuals of class W1 after generalization.

4.4.3 Filtering

The *filter hierarchy* applies a filter function to a set of individuals on one level and generates a subset of these individuals on a higher level. The individuals at the higher level are also represented at the lowest level. Individuals pass the filter at one or more levels of detail. Individuals that do not pass the filter disappear from the representation (Fig. 4.4). This is the most striking difference between the two other hierarchies and this hierarchy. The class and the type of the individuals stay the same.

$$s_1 \leq s_2 \text{ iff } \exists f_{\text{fil}} \text{ such that } s_2 = f_{\text{fil}} (s_1).$$

Two sets of individuals s_1 and s_2 are partially ordered if there exists a function f_{fil} such that the set s_2 can be filtered from the set s_1 ; s_2 is a subset of s_1 .

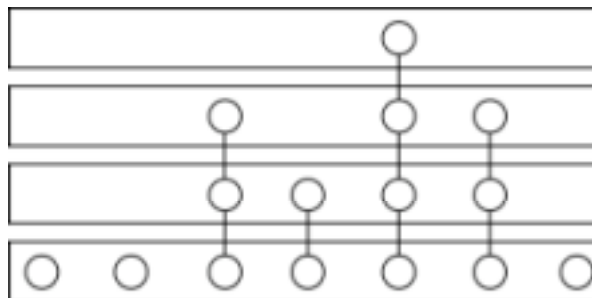


Fig. 4.4: Filter hierarchy

4.5 Combining Hierarchies

The three types of hierarchies mentioned have a relationship. A filter hierarchy can be deduced from a generalization hierarchy. The filter hierarchy is the mechanism that creates the aggregation hierarchy.

A generalization hierarchy describes the relationship between classes and their generic super-classes. For example, classes c_1 , c_2 , c_3 are at the lowest level. c_1 and c_2 have common super-class b_1 , c_3 has a super-class b_2 . Classes b_1 and b_2 have a common super-class a_1 . If an individual belongs to a certain class c_1 , then it is represented at the level of that class. The filter function is 'individual belongs to super-class'. If an individual i_1 and another individual i_2 belong to classes c_1 and c_2 respectively, then they can be aggregated to form a new individual i_3 that belongs to class b_1 .

4.6 Summary

In this chapter, we gave reasons why hierarchies are necessary for multi-scale databases. The main reasons are that hierarchies reduce processing time, increase the stability of an information system, and break down the tasks into manageable portions.

We distinguished hierarchies by their functions. We looked in detail at three functions producing three different types of hierarchies: aggregate, generalize, and filter. The aggregation hierarchy is built by aggregating objects. The generalization hierarchy defines how classes relate to more generic super classes. The filter hierarchy filters objects according to a criterion.

In our work we assume that many individuals with simple behavior can be combined to a larger individual with complex behavior. The three types of hierarchies can be combined to build a complex structure. In this work we aim at finding the elements of a map series. The elements should be easily describable and a mechanism for combination of the hierarchies must be found.

Chapter 5 - Map Model and Map Cube Model

In this chapter, we develop a theory on the structure of a map series. First we present a model of a map. We give an ontology of map elements and assign them to the objects of the model. A formal model of the map model is given in Gofer (section 5.1). Then we derive a model for a series of maps, connecting several of the map models with the help of tree structures. This model is called the map cube model (section 5.2). The formalization of the map cube model is described in chapter 8.

5.1 The Map Model

In our model, we distinguish four classes of entities in a map (Fig. 5.1), called map objects:

- trans-hydro network
- containers
- areas
- map elements.

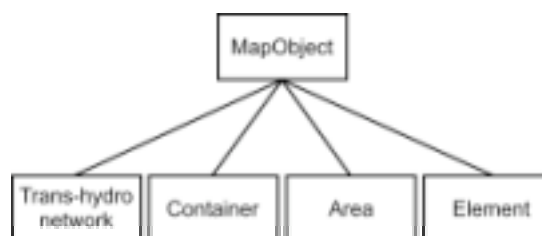


Fig. 5.1: Map objects

The trans-hydro network is a merge of the networks of transportation and hydrology. From a bird's perspective, those lines that are black and broad give a first subdivision of space. Lagrange (Lagrange and Ruas, 1994) and Bannert (Bannert, 1996) have proposed a division of space with the help of the road network. This idea is taken up and extended here: We subdivide a map with the hydrographic network, the railway network, and the road network.

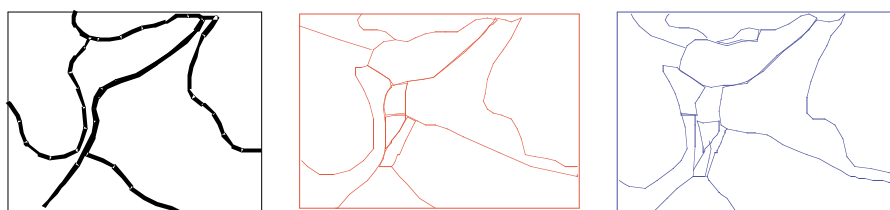


Fig. 5.2: Trans-hydro network creates partitions of space at three levels of detail

The *trans-hydro network* creates a partition of map space (Fig. 5.2). Buttenfield et al. (Buttenfield *et al.*, 1991) have expressed a similar idea (see also (Leitner, 1993; Leitner and Buttenfield, 1995)).

Containers are those areas between the lines that the trans-hydro network creates on the map. Containers completely partition map space, if we neglect the width of the lines of the trans-hydro network. The set of containers is the dual to the trans-hydro network within the space of the map they cover.

Areas are a refinement of the container partition and thus need to be contained within a container. In contrast to the containers, areas are bounded by fences, ditches, terrain peculiarities, land-use changes, and the like. Only the borders that coincide with container borders are part of the trans-hydro network. We use the term area in the sense of land-use area, denoting e.g., residential areas, garden areas, and cemeteries.

Elements are contained in the land-use areas. Elements are the smallest type of entity in a map. Map elements may be houses, church symbols, other symbols, labels, terrain features, etc. They thus are mostly alike to traditional map elements, with the exception of all linear structures. The next section assigns map elements to map objects in our model.

5.1.1 Ontology of Map Elements

We identify the following classes of map elements in our case studies, depending on the element's expected behavior over several levels of detail. The classification is based on geometric and semantic properties of the objects, similar to the classification by Kottik (Kottik, 1997). We select some classes for our model and give reasons why we omit other classes.

STREETS AND RAILWAYS

Streets and railways are included in our model as part of the transportation and hydrology network. We make a distinction between paths and dead-ends. Dead-ends are classified as map elements. Paths serve as borders of containers. The areas between paths are considered to be objects, e.g., building blocks, and in some cases the areas may have names like the "Quartiers" in Paris or the "Bezirke" in Vienna.

HYDROGRAPHIC OBJECTS

Hydrographic objects like rivers and streams are included in our model as part of the transportation and hydrology network. Lakes and ponds are classified as areas with landuse 'water' or as elements embedded in an area. Sometimes the distinction between lakes and waterways cannot be clearly drawn. For example, the river Rhine flows through the lake Constance. The automatic classification of such objects is deferred to future work.

LAND-USE AREAS

Land-use areas are included in our model. The land-use class is given by the color and eventually a symbol group. We assume that a map is completely classified by land-use. The borders of land-use areas may be fences, ditches, walls, and even buildings. The symbols for area features themselves (e.g., symbols for forest or grassland) are not included in our model. They do not structure space, but are purely graphical objects, similar to labels. A single tree symbol, however, must be included in the model.

BUILDINGS AND SINGLE SYMBOLS

We include buildings and single symbols (e.g., for pumps, churches, industrial buildings) in our model. Both types of objects are single objects embedded in land-use areas. Buildings are represented as red or black polygons. Single symbols give additional information on types of buildings (e.g., a church symbol). They may also represent point objects too small to be represented, but too important to be left out (e.g., a pump symbol).

TERRAIN FEATURES

Terrain features and terrain symbols are not included in our model. Terrain is a continuous phenomenon and it is difficult to ascribe behavior to terrain features (Buttenfield *et al.*, 1991). Terrain features do not partition space in the same sense as, for example, the trans-hydro network does. Exceptions from this rule are mountainous regions, which we do not consider here.

The cartographic generalization of terrain is a field of study, that has received much attention (Weibel and Heller, 1991). However, the inclusion of structural components of the terrain into our model should be considered in future work. Structural components are

those, which influence the cartographic generalization process. For example, two buildings on both sides of a ridge must not be merged.

LABELS

We exclude labels from the comparison and consequently from our model. Labels do not represent real-world entities on the map. They give supplementary information to the reader. The process of applying labeling to an otherwise finished map is a purely graphical process. There also exist algorithms which deal with this special aspect of map compilation as a post-processing step (Freeman and Ahn, 1987; Freeman, 1991).

OBJECTS APPEARING AT ONLY ONE SCALE

We exclude objects from consideration that appear only at one scale (e.g., power lines), although they can be fitted easily into the existing model. These objects do not partition space or the map partition they create is not useful in our model.

THREE-DIMENSIONAL OBJECTS

Three-dimensional objects (e.g., city gates) represent a problem, that could not be solved for maps and therefore cannot be solved for a multi-level database. One example is the case study Murten A, where a street runs through the old part of the city. This is not shown on the map because it passes through the old city gate and it was more important to draw the complete city walls than the continuing street. In our theory we disregard this problem, but it needs to be addressed in future work.

5.1.2 Behavior of Graphic Objects

Graphically, we identified nine changes of the representation of graphic elements in a map series for each direction of scale change (Timpf and Frank, 1995a; Timpf, 1997). Table 5.1 shows the changes for both directions of scale change: “zooming out” and “zooming in”. These changes do not indicate the cartographic operations on the elements. They are observations of the result, not of the process of cartographic generalization. For more information on the process itself, see (Buttenfield and McMaster, 1991; McMaster and Shea, 1992; Mueller, Lagrange, and Weibel, 1995).


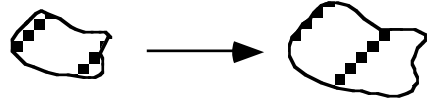





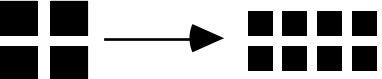
	Zooming out / zooming in	impact on data structure
1	no change / no change 	no
2	scale decreases / scale increases 	no
3	symbol changes / symbol changes 	no
4	detail decreases / detail increases 	no
5	change to less dimensions / more dimensions 	no
6	symbolization / shift to geometric form 	no
7	several elements merge / split into several objects 	yes
8	--- / typification 	yes
9	element vanishes / element appears	yes

Table 5.1: Graphical element changes

The changes can be divided into two groups. Changes in the first group are graphic changes, having no impact on the data structure. The second group of changes has an impact on the data structure. Those changes correspond to the four functions *merge*, *omit*, *typify*, and *continue*.

5.1.3 A Formal Map Model

The model (Fig. 5.3) captures the topological and structural aspects of a map, that is, the relationships of objects at one level of detail.

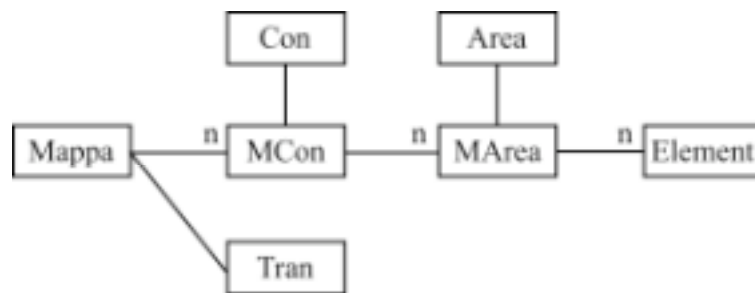


Fig. 5.3: Model of a map

A map (here called *Mappa* to avoid confusion with a reserved word in Gofer) is determined by its name, the trans-hydro network, and a list of containers (brackets designate a list of something):

```
Mappa = Name Tran [MCon]
```

The trans-hydro network, the container, the area, and the element are map objects (*Mob*):

```
Tran = Mob, Con = Mob, Area = Mob, Elem = Mob
```

A map-object has an identifier, a code (determining if it is a container, an area, or an element), a name, a level of detail, and a directed graph:

```
Mob = Mob ID Code Name Lod DGraph
```

A container (*MCon*) is a map object and has a list of areas:

```
MCon = Con [MArea]
```

An area (*MArea*) is a map object and has a list of elements:

```
MArea = Area [Elem]
```

These definitions in combination with the class and instances definitions shown in chapter 8 describe the structure of a map according to our model. The full Gofer-code is given in the appendix.

5.1.4 Remarks

The model does not take into account that, for example, cities are not surrounded by streets but represent a particular geographic entity that should be represented in our model. It usually is a problem to determine where a city ends and the only information is the administrative boundary of the city. This is the same for small villages in the countryside that are divided by a road. However, this boundary does not necessarily conform to the boundary of the built-up area.

The problem above is a consequence of the Janus effect (Koestler, 1967) that each entity in a hierarchy has: a city is a single entity if looked at from above and a collection of entities if looked at from below. Somewhere is a level of detail where the representation of parts changes to a representation of wholes. This depends on the size of the whole entity in relation to the level of detail and to the size available for the representation. If the available area is large, then even large cities are single entities. If the available area is small, then the city cannot be seen as a whole and only parts of it can be displayed.

However, the problem described here does not occur in our case studies, given the scale range and the samples chosen. One solution could be to design a second set of containers given by the administrative boundaries. This set of containers may be very hard to relate to the first set, to the areas, or even to the trans-hydro network. Casati and Varzi (1994) have proposed to represent and handle administrative boundaries as if they were shadows. This would result in a set of shadow-containers. The idea deserves investigation in future work.

5.2 *The Map Cube Model*

In this section, we introduce a method to describe the structural content of a topographic map series with the help of four different graphs: the trans-hydro graph, the container graph, the area graph, and the element graph. When building the graphs, we assign a vertex to each map object and an edge to each relation between map objects at different levels of detail. The resulting structure is a graph for each high level object. For example in fig. 5.4, each graphic representation signifies a vertex. The high level object at level (b) is represented at each level down to level (d). The idea is that every map object is represented

in a tree or graph, i.e., there are for every high level map object multiple low level objects, organized in decreasing levels of detail. This includes that an object may split into sub-objects. Map objects at the same depth of the tree belong to the same map.

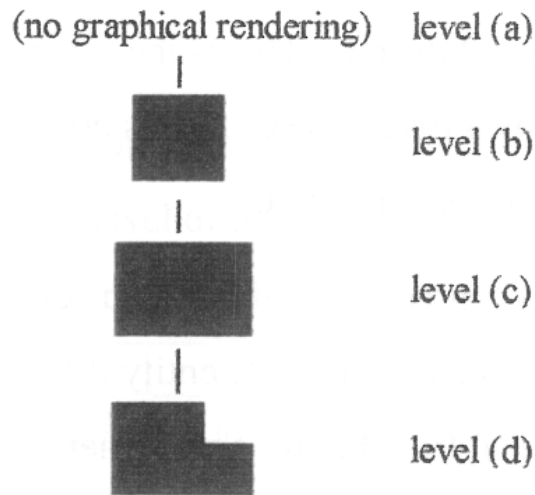


Fig. 5.4: Objects depicted with increasing graphical detail

At the end of this section we combine the four graphs into a single model, the map cube model. A cube as a symbol for a database has been proposed by (Frank, 1992) in the context of user interfaces. We will explore all three dimensions of the cube for our map series, using the cube as a convenient image for the spatio-temporal process of zooming a map.

5.2.1 Trans-hydro Graph

We first subdivide map space by the trans-hydro network. This gives us a partition of map space for each map. The change of the level of detail entails that there are less and less detailed transportation networks in each map as the data get more and more abstract.

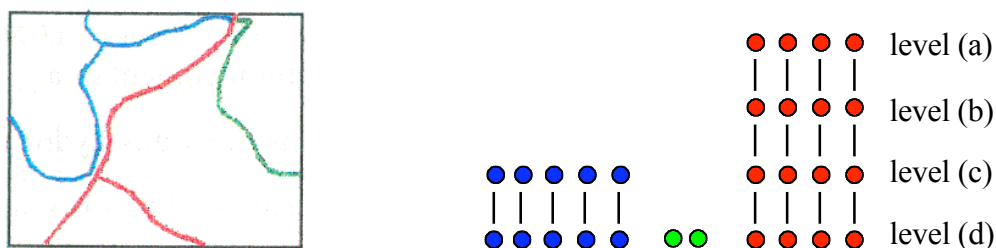


Fig. 5.5: Trans-hydro Graph

Fig. 5.5 shows on the left hand side a schematic representation of the map objects represented in the trans-hydro graph. On the right, each circle represents a segment of a street. There are no railroads or waterways in this example. The colors signify that the segments belong to the same street. Two segments are filtered from level (d) to level (c), another five segments are filtered from level (c) to level (b). The red circles stand for the boundaries of the space under consideration.

It is important to note, that the more abstract version of the trans-hydro network is a subset of the more detailed version. For this reason the trans-hydro graph has the structure of a filtering hierarchy (Fig. 5.5).

5.2.2 Container Graph

We see a consistent progression of the container partition towards a most detailed partition if we zoom into the data. The tree created by tracing the containers throughout the progression is called a container graph, or, more specific, a container tree (Fig. 5.5).

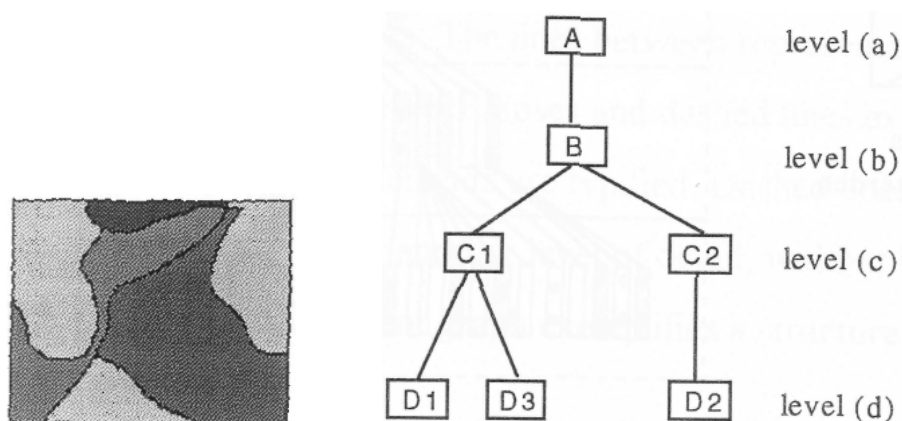


Fig. 5.6: Container Graph

Spaces between the edges of the network are called containers, because they contain information, e.g., on land use, buildings, or symbols. Containers are denoted as boxes with a unique letter in the container tree. Lines between boxes signify that the container at the higher level contains the one(s) at the lower level. Or conversely, the container at the lower level is part of the container at the higher level. The container tree thus has the structure of an aggregation hierarchy. We assume in our model that the ordering is consistent, i.e. that there are no cross-links in the container graph.

Fig. 5.6 shows only one part of a map. Each partition of a map produces at least one tree. The collection of all trees in a map is called the container graph. If we consider the map to be the top container, then the container graph will always have the structure of a tree.

5.2.3 Area Graph

The tree generated by progressing through the levels of detail, looking at the areas, is called area tree (Fig. 5.7). This tree also represents a subdivision of space, the spaces are smaller than in the container tree, and they subdivide a single container. The partition of areas is a refinement of the partition of containers. The collection of all area trees in a map is called area graph.

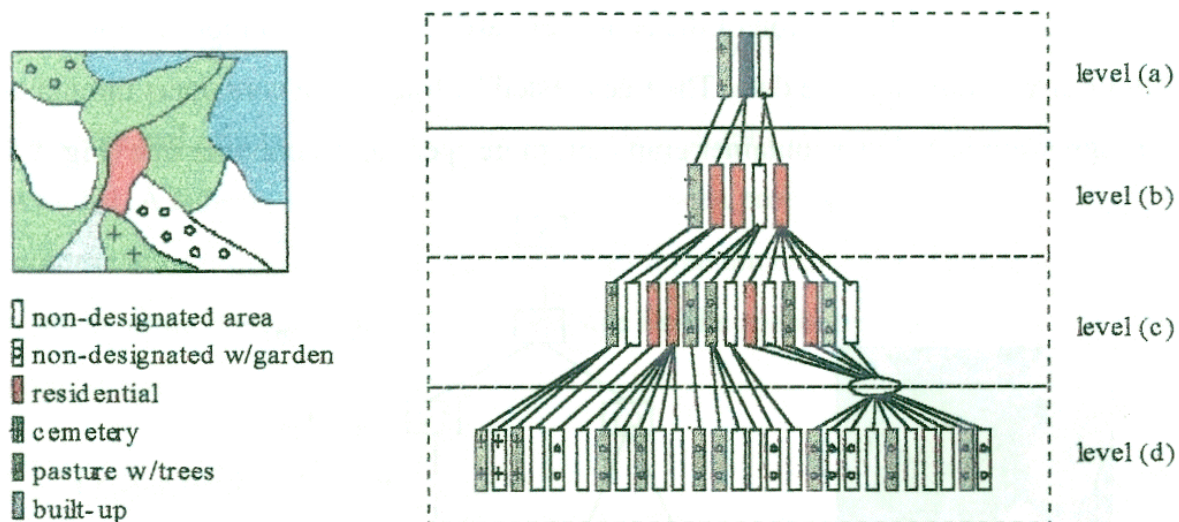


Fig. 5.7: Area Graph

Fig. 5.7 shows an area graph consisting of three trees. Each area is represented by a box in the color and with the symbols of the original area. Lines between boxes signify the relation contains/part_of as in the container graph. The area graph has the structure of a generalization hierarchy: land-use classes are generalized. The spatial representation of the areas is aggregated. The ellipsis in Fig. 5.7 signifies that corresponding areas could not be identified.

5.2.4 Element Graph

The elements contained in an area are either buildings, symbols, or dead-ends. The graph created when progressing through the levels of details and linking these elements is called element graph (Fig. 5.8).

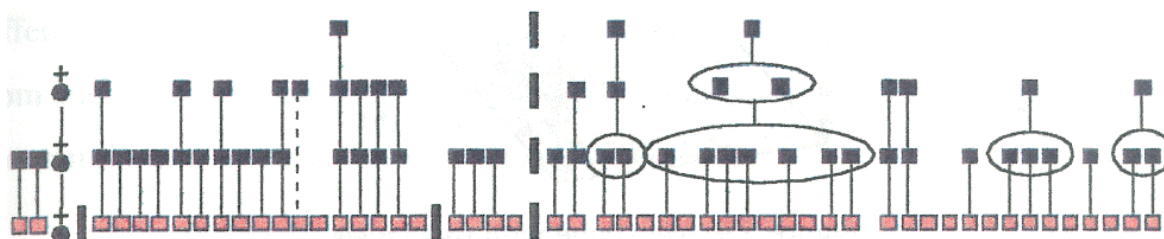


Fig. 5.8: Element Graph

Fig. 5.8 shows the element graph for one container. It is important to notice that elements always form a graph. The graph is composed of many trees, having their root nodes at different levels. Buildings are represented as small boxes in the color of the original building in the map. Symbols are represented as symbols and dead-ends are represented as black upright rectangles. The lines between representations signify that the element represented in the next level. Ellipses and dashed lines express a special type of relation. Ellipses signify that the elements are typified. Dashed lines signify that the element disappears, but re-appears at another level of detail, without being shown at the intervening levels of detail. The element graph exemplifies a structure that combines different hierarchies.

5.2.5 The Cube Model

One can imagine the structure of a digital map series as a three dimensional composition. Each map is a horizontal cut through a three-dimensional cube, the vertical dimension represents the level of detail, the other dimensions represent the x and y axes of a map (Figure 5.9).

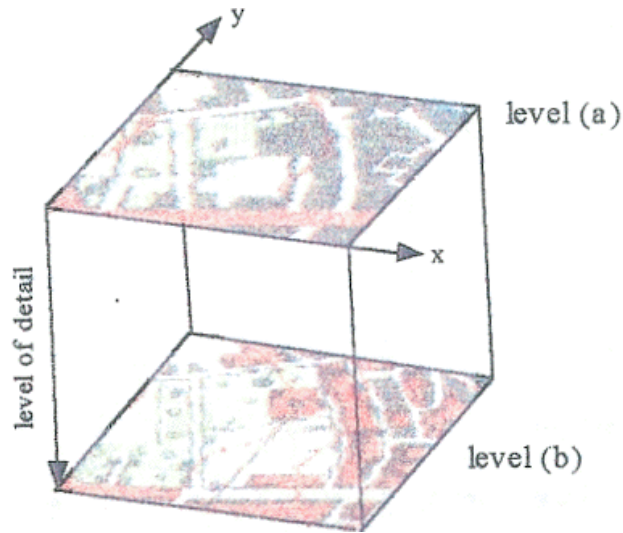


Figure 5.9: Map cube with two levels

The cube model combines the trans-hydro, container, area, and element graph in a single model. Each of these graphs could be represented in a separate cube. All four cubes are related to each other. The trans-hydro graph corresponds to the boundaries of the containers on each level. The areas are a refined version of the containers and the elements are contained in the areas. The cube for the trans-hydro graph can be flattened to the lowest level of detail, because the other levels of detail can be calculated from the most detailed trans-hydro network.

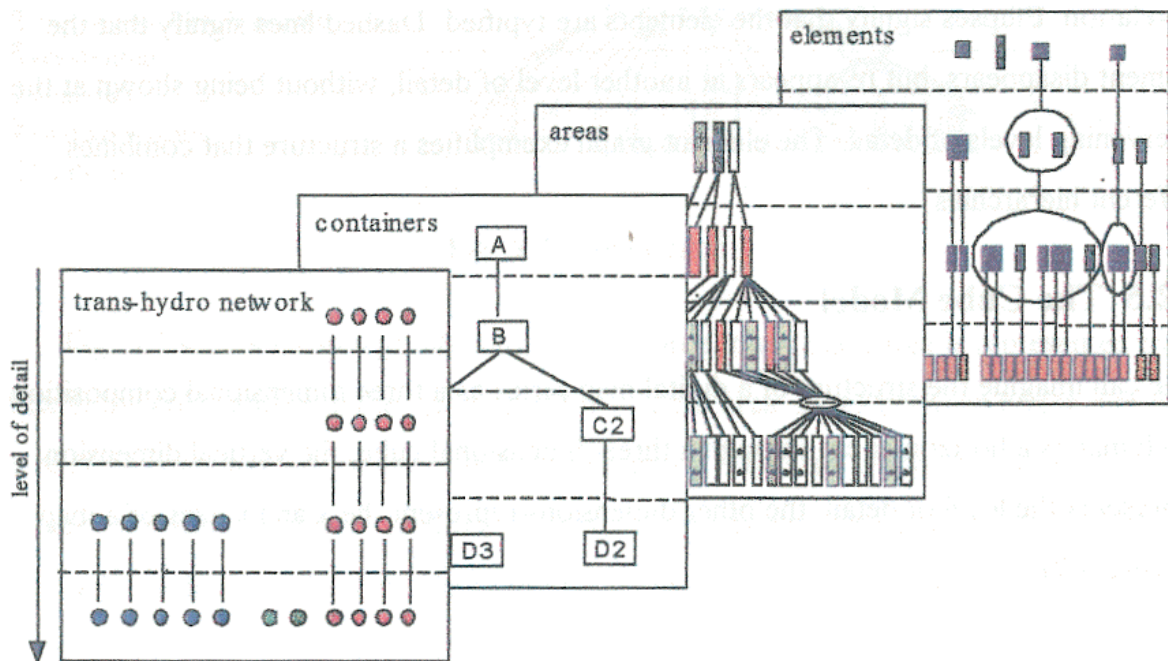


Figure 5.10: Components of a digital map series

Figure 5.10 shows the components of a digital map series. The same level in all graphs yields the components of a map at that level of detail. The x and y axes from Figure 5.8 are collapsed into a single symbolic dimension. The spatial objects of the map are represented as boxes and symbols. Figure 5.10 shows all the possible relationships between map objects. Each map object is related to its own type at a different level as shown on the plane. The map knowledge makes it possible to reason from one plane to the next, from left to right in the figure above, using “borders”, “contains”, and “contains” as the main relations.

5.3 Summary

In this chapter, we first presented a model for a map. We introduced four map objects, named trans-hydro network, container, area, and element. In an ontology of map elements we described which of the elements we included in our model and how these elements are assigned to the four map objects. The container is an exception, because it is a map object created by the subdivision of map space through the trans-hydro network. We also defined a formal model of the map in Gofer.

Based on the map model we developed a theory on the structure of a map series. We introduce the notion of a cube for a map series. Each horizontal plane represents a map, the third dimension adds the level of detail. The structure of the map objects over several levels of detail is a tree or a forest. Each of the map objects forms a cube of its own, showing their behavior over several levels of detail. The combination of all four map object cubes recreates the map. The formalization of the map cube model is described in chapter 8.

Chapter 6 - Observation Plan

This chapter describes in detail how we carry out the observations in the case studies. The first section explains the impact of the level of detail on map series and what this means for our case studies. The second section details how to compare map objects in general and lists special rules for the comparison of maps at different levels of detail. Section three specifies our strategy for matching map objects and gives an example how to build the container graph, the area graph, and the element graph.

6.1 Map series

A map series consists of several maps of the same area at different scales. The scales build a series; they are ordered from large scale (e.g., 1:1,000) to small scale (e.g., 1:500,000). We bring the maps to the same scale, i.e., we enlarge the small scale map to the scale of the large scale map (see Fig. 6.1). The resulting maps have the same nominal scale, but a different level of detail. The enlarged map shows building blocks but not single buildings. It has less streets, the letters of the labels and the symbols are larger, the symbols are different, etc.



Fig. 6.1: Map clips at different levels of detail shown at same nominal scale

The sequence of maps at different levels of detail is also a map series; the order of the sequence is determined by the level of detail. We say that the map with the least detail is the most abstract map and at the highest level. Consequently, the map with the most detail is at the lowest level. In Fig. 6.2, Map 1 is at the highest level and Map 3 is at the lowest level.

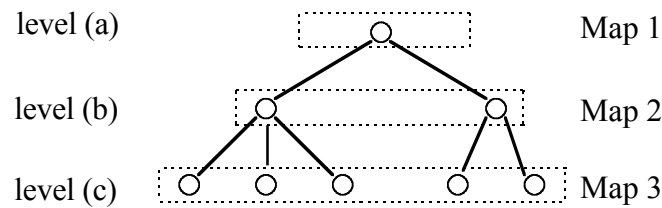


Fig. 6.2: Relations between maps at different levels of detail

One goal of this thesis is to observe the behavior of map objects over several levels of detail. The aim is to assign to each map object those map objects at the levels below and above that represent the same real-world entity. For a human this seems an easy task. A computer, however, needs additional assistance in identifying map objects as being related to each other. The map model described in the last chapter gives this assistance, because it assigns an *area* to a *container*, as well as a *map element* to an *area*. In addition, the knowledge about possible cartographic transformations of elements helps with the identification.

For this study, we consider map objects that have been transformed by cartographic functions (e.g., merge or symbolize) as sequels of the original map objects. Although objects disappear or are merged with other objects, they disappear in a certain area or are absorbed by a larger object. In those cases the original object and the higher level object are related to each other. The sequel sometime provides the connection between graphs of different map objects. For example, a cluster of buildings within a residential area is represented as a built-up area in a map of lower level of detail, making a connection between the element graph and the area graph.

6.2 Comparing maps

When comparing maps, one compares collections of map objects, i.e. containers, areas, and map elements. For our model we require that the data of the map series are consistent, i.e. that the graphs or subgraphs are trees. However, a given map series is not made to be consistent, since the “level (a) map” is not derived from the “level (b) map” but usually from the most detailed map. From the perspective of error propagation, this is a sensible rule, but its leads to inconsistencies. Cartographers differently apply the rules for cartographic generalization. In some cases it is then impossible to relate objects on two

levels of detail to each other. We stress the need for consistency in map series if the maps are included in Geographic Information Systems. This stands in direct relation to the concept of consistent databases and of error treatment in databases. The answers to queries and the reasoning results of the system should not change with the level of detail of the data.

Maps at different scales are made for different purposes (e.g., for topography versus transportation) and thus different things are stressed in the maps (e.g., landmarks versus networks). This means that objects may appear in one map, but may not be represented in another map of a different scale. Or that objects in one map are displaced more and cannot be matched to their counterpart in the other map. The map scales we chose do not exhibit this strong distinction of purposes.

Maps at different scales have different symbolization schemes. This makes comparison harder. For example in case study 7.2, the red and black areas supposedly give distinct information about the land-use. A red area signifies residential area and the black area signifies built-up area (can also be industrial). However, the rule when a certain area is considered built-up or residential is not applied consistently over several maps, so that the land-use class can switch from residential to built-up and back over several maps within a map series. Usually, topographic map series ranging from map scales 1:20,000 to 1:100,000 or even 1:200,000 use similar symbolization schemes. The symbolization scheme changes with the purpose of the map: e.g., a map at scale 1:500,000 is used to show connections between locations, so the symbolization stresses roads. The German map series we chose has the same symbolization scheme by convention (Haack, 1996).

6.2.1 Rules for comparison of map objects

We applied the following rules to make the comparison between maps easier. When we are in doubt, we decide in a manner that the final structure is consistent. We try to avoid complicating the structure.

When determining areas of land-use, the areas are not only circumscribed by borders, but also by buildings. Sometimes it is hard to determine if these buildings belong to that area or not. We determine the shape of the area, so that it is close to a rectangle and

incorporate the buildings accordingly into the border. Although this choice is arbitrary it does not have an impact on the data structure.

Along rivers, streets tend either to go along the river with only a minimal space between street and river or to dead-end just in front of the river. In the second case we introduced an artificial intersection of river and street to conclude the area. This prevents long thin slices along the river banks with many dead-end streets.

6.2.2 Remarks to the Observation Process

We exclude features of the map from consideration that appear only at one scale (e.g., power lines) or do not partition space (e.g., isolines). We also disregard labels, because they do not represent real-world entities on the map. They give special, supplementary information. The process of applying labeling to an otherwise finished map is a purely graphical process. There also exist algorithms which deal with this special aspect of map compilation (Freeman and Ahn, 1987; Freeman, 1991; Barrault, 1995).

Terrain features are not included at all into the current model. They do not partition space in the same sense as the transportation network does. The cartographic generalization of terrain is a special subfield and has received much attention (Weibel and Heller, 1991). Please refer to chapter 5 for a complete discussion.

6.3 A Strategy for Matching Map Objects

In this section we define a strategy for matching map objects from different levels of detail. Our aim is to link objects that represent the same real-world entity. A strategy is needed on how to identify that two map objects represent the same entity. The objects are different in the level of detail and in their symbolization. They also differ because of the effects of cartographic generalization: Objects may not be found at exactly the same place. Their shape may be simplified or exaggerated.

We subdivide map space into smaller parts, called containers, that can be more easily matched (section 6.3.1). This subdivision is refined, resulting in even smaller parts of the map, that are land-use areas (section 6.3.2). Map elements are contained in land-use areas. With this strategy, map elements can be compared and related to each other (section 6.3.3).

To preserve clarity of the figures, we pick a single container to be observed in the case studies. The observation is straightforward and can easily be extended to whole maps. The trans-hydro graph and the container graph are dual graphs. Therefore, the trans-hydro graph is not represented in our case studies.

6.3.1 Building a Container Graph



Fig. 6.3: Representation of example case study at three levels of detail

Figure 6.3 shows the example study area at three levels of detail. The level of detail progresses from the left to the right, from most abstract to most detailed. We picked the container in the center of the left map clip. It is circumscribed by two red roads and three white roads. All containers on a map constitute a subdivision of map space. Starting from the least detailed map, the subdivisions of map space are more refined the more detail is added (Fig. 6.4). This hierarchy of subdivisions can be represented by a tree. At each level of the tree the set of containers is a complete partitioning of map space.

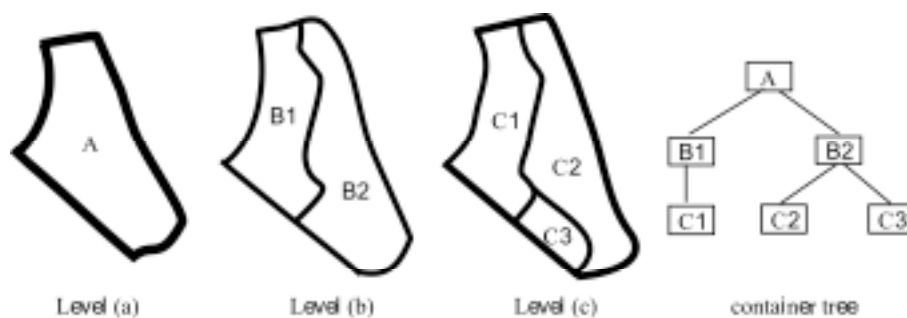


Fig. 6.4: Networks for spatial subdivision and Container Tree

In this particular example only the road network subdivides the map space. At Level (a) there is one container, called A. Level (a) is the most abstract level. Level (b) shows two areas B1 and B2. Areas B1 and B2 at correspond to area A on level (a). Level (c) shows three areas C1, C2, and C3. Areas C2 and C3 correspond to area B2 on level (b).

In this example, the partitions are consistent and can be represented by a partition tree (on the right in Fig. 6.4). A container graph is consistent if the partition on the lower level is a refinement of the partition on the higher level. Only in that case is the structure of the graph that of a tree or of a forest.

6.3.2 Building an Area Graph

Areas form a refined subdivision of space in the containers. In the following example four land-use areas have been identified. They are built-up area (black), residential area (red), garden area (green with circles), cemetery (green or white with crosses), non-designated area (white), and non-designated area with some trees (white with circles). Land-use areas are determined either by change of color or through an existing boundary (e.g., fence, ditch etc.). New containers are created if a street divides a land-use area into two or more separate areas (see section 6.3.1).



Fig. 6.5: Containers and Areas

For example, in Fig. 6.5 the container A consists of just one area, which is color-coded as a built-up area. The container B1 includes a residential area. The container B2 includes three areas: a residential area and two non-designated areas. The container C2 includes five areas: a residential area, two garden areas and two non-designated areas. The container C3 includes a residential area and a garden area. The results of this subdivision at each level are three area partitions shown underneath the map clips in Fig. 6.5.

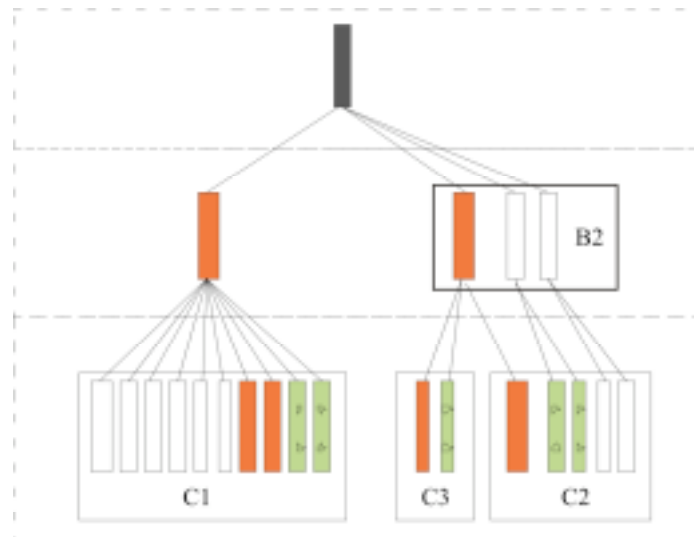


Fig. 6.6: Area Tree

Figure 6.6 shows the area tree, which is a refinement of the container tree. We already know from the container tree in figure 6.4 that the container B2 is subdivided into two containers C2 and C3. In the more detailed area tree, we can see that the residential area in B2 is split into three areas falling into both containers C3 and C2.

6.3.3 Building an Element Graph

So far we have looked at containers, areas, and their nesting. Now we will look at the elements that are included in an area. Elements are buildings, symbols and dead-end streets. After identifying the elements in each land use area, we match elements from different levels and build an element graph. Figure 6.7 presents the element graph for our example.

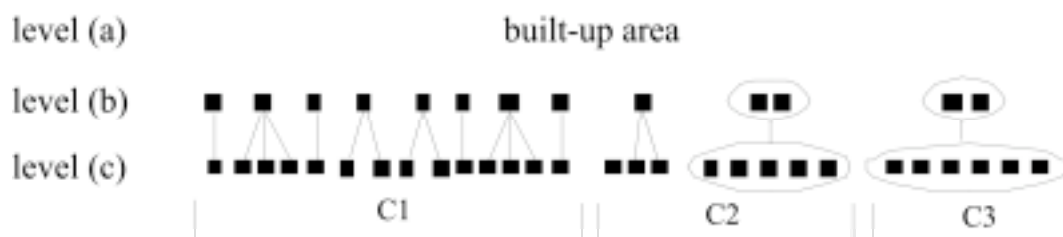


Fig. 6.7: Element Graph

We see that the element graph is a collection of trees. The areas in C1 contain fourteen buildings that are either still represented or merged to a building representation in B1. There are three buildings in the white area in C2. Two of the buildings are merged, the third disappears in level (b). In the residential area of container C2, there are five buildings

that are typified to two buildings in the residential area in B2. In the residential area of C3 there are six buildings that are typified to two buildings in the residential area in B2. It is also interesting to note that there are no single elements at level (a). It is an area representing the main land-use; in this example it is a built-up area.

There are special links in the element graph that allow different types of relations between elements. If two or more elements are combined to one element, then the elements are merged to a single element. Ellipses around elements signify that the group of elements is typified. A dashed line between elements means that the element disappears at one level, but reappears at another one.

6.5 Summary

A map series consists of several maps of the same area at different scales. The series is given by the map scales. In order to compare maps of a map series we need to bring the maps to the same scale. In our example we enlarge the small scale map to the scale of the large scale map. The resulting maps have the same nominal scale, but a different level of detail. When comparing maps we assume that the data are consistent in a map series. This implies that the resulting graphs are either trees or forests.

We define a strategy for matching map objects from maps at different levels of detail. Our aim is to link objects that represent the same real-world entity. We subdivide map space into smaller parts, called containers, that can be more easily matched. This subdivision is refined, resulting in even smaller parts of the map, that are land-use areas. Map elements are contained in areas. With this strategy, map elements can be compared and related to each other. We give an example and explain in detail how to build the container graph, the area graph, and the element graph of our chosen map clip. The next chapter presents eight case studies, where we build graphs to construct a model of a map series.

Chapter 7 - Case Studies

7.1 Overview of Study areas

This section gives examples for the construction of graphs. Our examples are drawn from two map series. The first map series comes from former East Germany (see Table 7.1), including the areas of Alsleben/Saale and Berneburg/Saale. Berneburg and Alsleben are small cities on the Saale. The maps are topographic maps and their scales range from 1:10,000 to 1:100,000. The scales 1:25,000, 1:50,000, and 1:100,000 were created within the same symbolization scheme. We assume that equal symbolization schemes facilitate comparisons over a range of scales.

	1:10 000	1:25 000	1:50 000	1:100 000
Name of map sheet	Berneburg (Saale)	Berneburg (Saale)	Berneburg	Berneburg
Number	M-32-12-C-b-2	M-32-12-C-b	M-32-12-C	M-32-12
Name of map sheet	Alsleben (Saale)	Alsleben (Saale)		
Number	M-32-12-C-d-3	M-32-12-C-d		
Year of last update	1986	1986	1986	1986

Table 7.1: Information on map series Berneburg and Alsleben

The second series stems from Switzerland (see Table 7.2), including the area of Murten. Murten is a small town at the border of the lake Morat. The map scales in this case range from 1:10 000 to 1:500 000.

	1:25 000	1:50 000	1:100 000	1:200 000	1:500 000
Name of map sheet	Murten	Avenches	Saane/Sarine	Landeskarte	Landeskarte
Number	1165	242	36	Blatt 1	n/a
Year of last update	1975	1981	1981	1972	1979

Table 7.2: Information on map series Murten

Selected areas of both map series have been scanned with low-resolution equipment and saved as color TIFF files. Smaller areas have been extracted from these TIFF files and

brought to the same scale, so that comparison became possible. We describe ten study areas from three sites in the two map series. Each study area is a container at the highest level. This container is the root of each of the graphs we build. Each subsection shows a specific example and presents all three types of graphs: the container graph, the area graph and the element graph. In Berneburg/Saale (Fig. 7.1), four study areas have been selected.



Fig. 7.1: Berneburg/Saale

The study area Berneburg I is described in subsection 7.2. The study areas Berneburg II and III are jointly described in subsection 7.3. The study area Berneburg IV is described in subsection 7.4.



Fig. 7.2: Alsleben/Saale

Four study areas have been identified in Alsleben/Saale (Fig. 7.2). We describe three of them in this section. Study area Alsleben I has many small streets in subsequent scales. There is also a problem of continued subdivision in that area. It is treated in subsection 7.5.

Study areas Alsleben II and III present no special problems for our method. They are described in subsections 7.6 and 7.7. Study area Alsleben IV was described in chapter 6.3 as the running example.

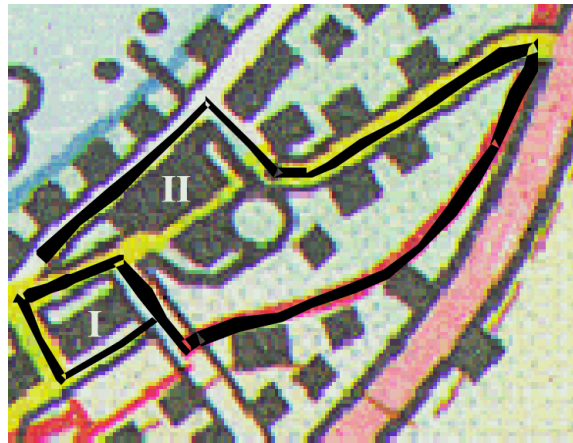


Fig. 7.3: Murten

In Murten (Fig. 7.3), two study areas have been selected. Study area Murten I is described in subsection 7.8. Study area Murten II presents a problem and is described in subsection 7.9.

It was not always possible to make the map clips exactly the same size, because of map generalization effects. These are, for example, exaggeration of the road size or displacement of the features.

7.2 Study area Berneburg I

The following figure (Fig. 7.4) shows the study area Berneburg I at four levels of detail. The level of detail progresses from the left to the right, from most abstract to most detailed. The containers of the area of interest are shown underneath the map clips.

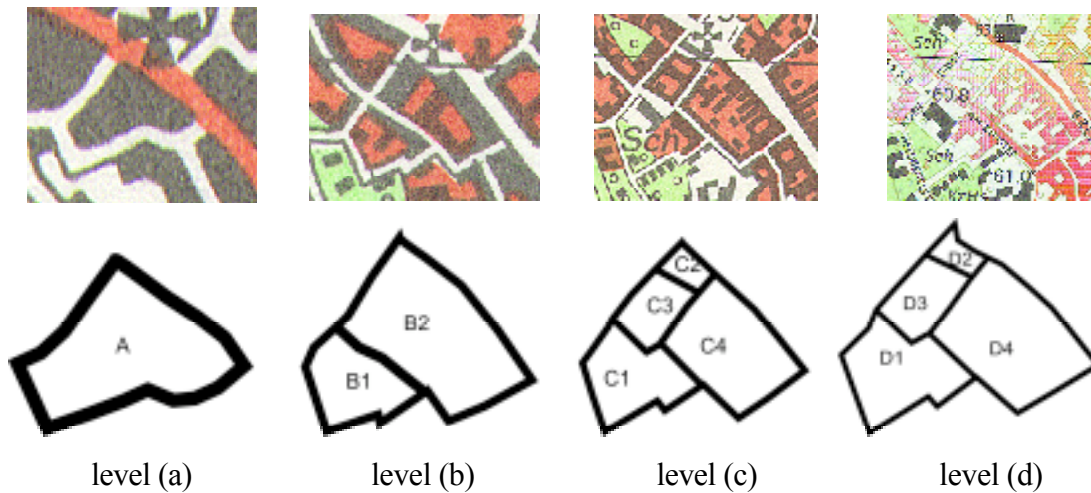


Fig. 7.4: Study area Berneburg I at different levels of detail

This study area features one container at level (a) that is divided into two, and then four containers at the subsequent levels (Fig. 7.4 and Fig. 7.5(a)). The major areas are residential areas and there is also a garden area, that contains an official building (a school).

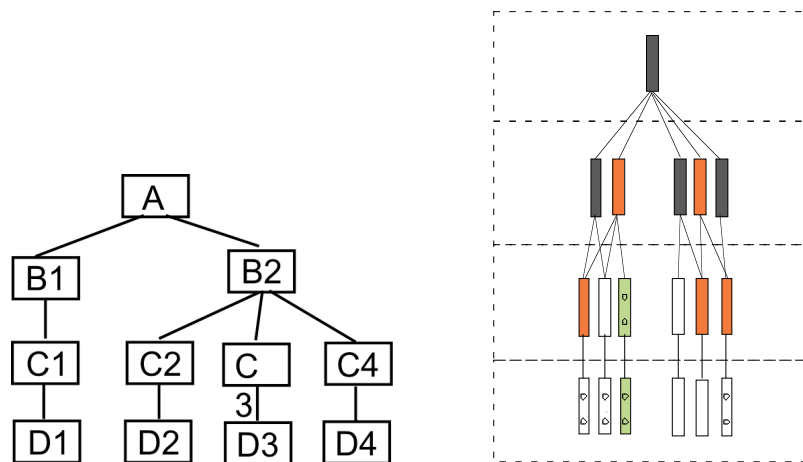


Fig. 7.5: Container Tree (a) and Area Tree (b) of Berneburg I

The area tree looks rather complex with some cross-links. Those cross-links stem from the different use of symbolization schemes in level (c) and level (b). In level (b) there

are two ways to describe a residential area: one way is to make the area red, the other way is to make it black. Both ways seem to be used equally often on the maps. If we did not distinguish black and red, then level (b) would have two areas and there would be no cross-links.

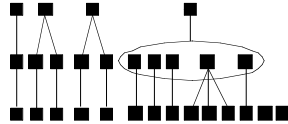


Fig 7.6: Element graph of Berneburg I

The element graph (Fig. 7.6) is a collection of element trees. The buildings at the highest level are combined to a built-up area, as can be seen in the area tree. This gives a connection to the area tree. At level (c) two objects from level (d) disappear. At level (c) the objects in the ellipse are typified to a single building.

7.3 Study area Berneburg II/III

Different from the other examples, Berneburg II/III holds two containers on level (a) (Fig. 7.7). This is necessary, because the containers are linked on one level where an area is claimed from both.

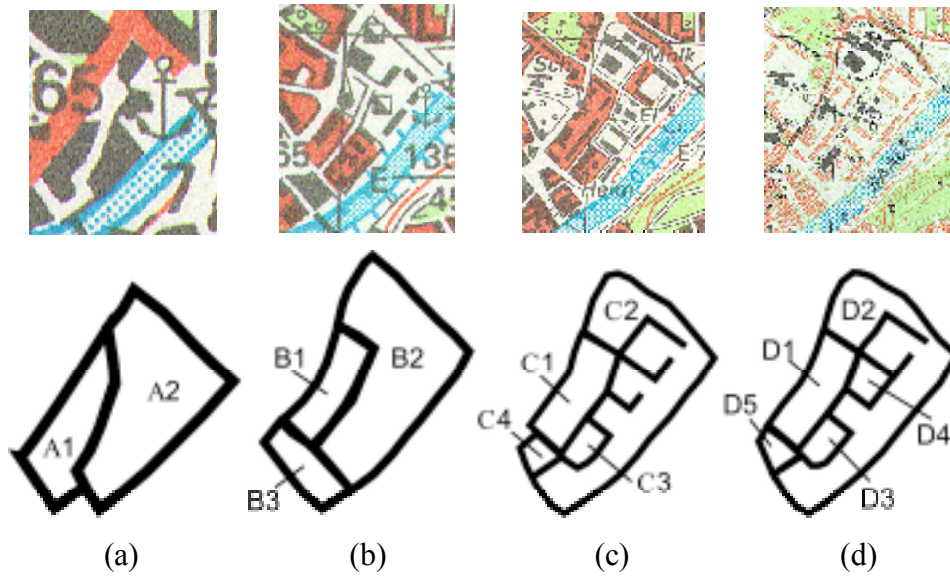


Fig. 7.7: Study Area Berneburg II/III at several levels of detail

Additionally to the two containers on level (a), the study area holds three containers on level (b), four containers on level (c), and five containers on level (d). It is bounded on three sides by streets and on one side by the river Saale.

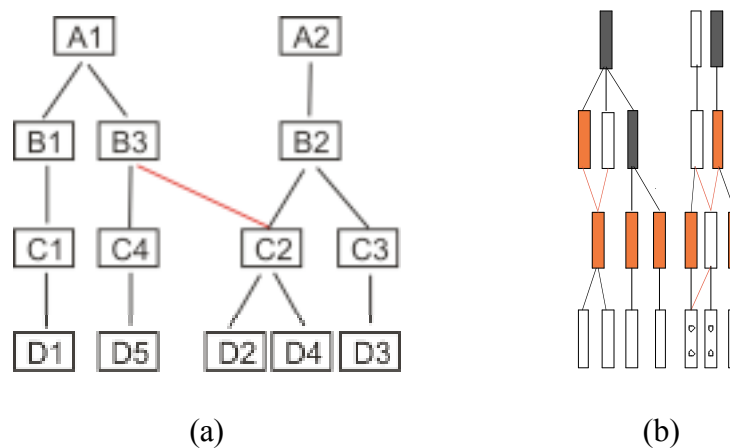


Fig. 7.8: Container Graph and Area Graph of Berneburg II/III

Tracing the containers over several levels results in the container tree of Fig.7.8(a). Note that both containers build their own trees with just one cross-link between the two trees. The division is even clearer in the area tree (Fig. 7.8 (b)). There are in fact two separate trees with no link between them. This suggests, that the link in the container tree is introduced artificially.

This area graph is a graph with several cycles and not a tree. This is due to the different handling or interpretation of the land-use classification, as explained in the previous example. Figure 7.9 shows the element graph of the study area.



Fig. 7.9: Element graph of Berneburg II/III

The element graph (Fig. 7.9) of this study area is a collection of trees except for the two symbols for industrial building (gray squares with gray triangle in one corner), that disappear on level (c) and reappear on level (b).

7.4 Study area Berneburg IV

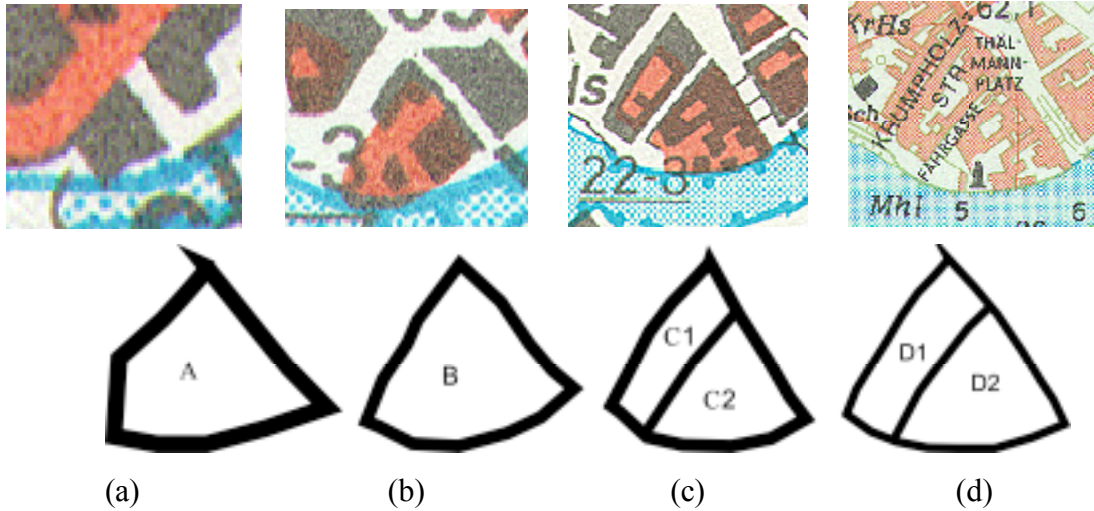


Fig. 7.10: Berneburg IV at different levels of detail

The map clip of study area Berneburg IV (Fig. 7.10) is bounded by two streets and a river. The main objects are buildings, the main areas residential areas. The single container at level (a) is divided into two containers at level (c) (Fig. 7.10 and Fig. 7.11(a)).

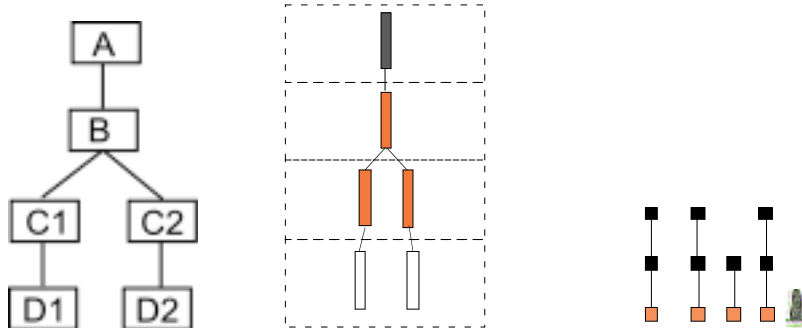


Fig. 7.11: Container Tree (a), Area Tree (b) and Element Graph (c)

In the area tree, there are two areas (white) at level (d), containing houses. On the next higher level, those areas are transformed into residential areas (red). At level (b) the areas are merged into a single area; this is concurrent with the merging of the containers C1 and C2 to container B. At the highest level, the area is a built-up area (black), that completely fills the container A. The element graph (Fig. 4.21(c)) shows four buildings on the most detailed level, that are preserved into the next level (although their shape is not). A symbol (for a tower) is only shown on the most detailed level. At level (b) there are only three buildings left that are then typified at level (a) as a built-up area.

7.5 Study area Alsleben I

The study area Alsleben I is shown at several levels of detail in Fig. 7.12, progressing from left to right. The most abstracted representation holds a single container A and the next level already has six containers B1 through B6.

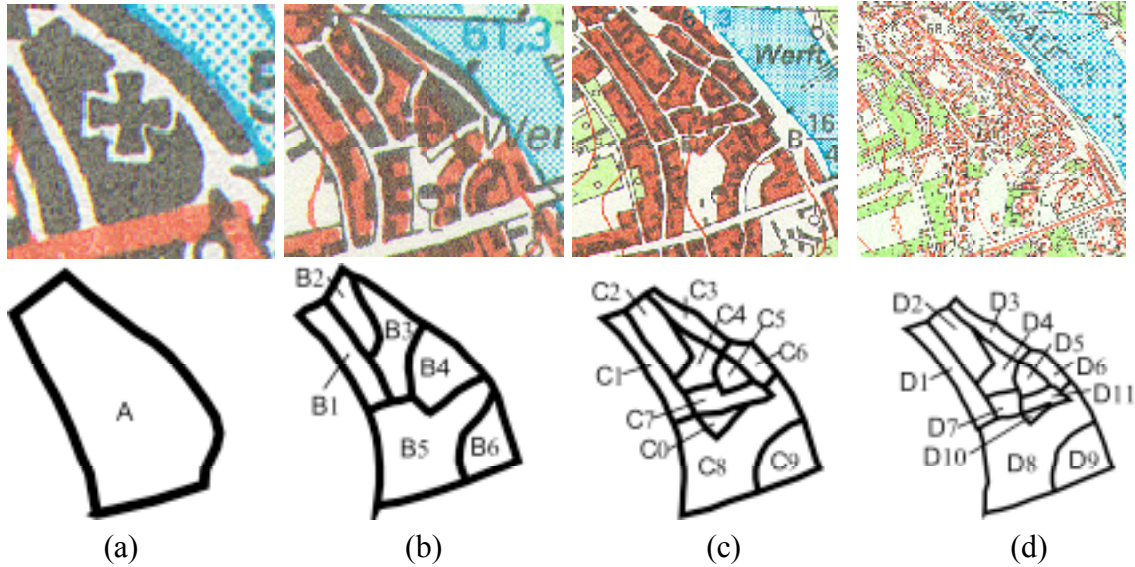


Fig. 7.12: Study Area Alsleben I at different levels of detail

Level (c) is composed of ten containers C1 through C0, and the most detailed level offers eleven containers D1 through D11. This series of containers leads to the container graph in Fig 7.13. From this container graph we can see that there is a problem with containers C7 and D8.

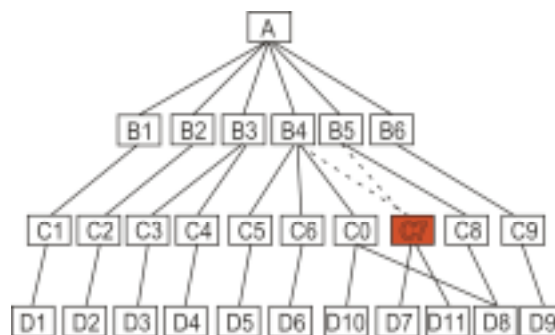


Fig. 7.13: Container Graph of Alsleben I

The containers at level (d) and those at level (c) form a partition each. The partitions do not correspond to each other. In the case of C7 the generalization from level (d) to level (c) did not take into account the street dividing D7 and D11, whereas the street was preserved in the generalization from level (d) to level (b).

In the case of D8, the problem is in the correct assignment of the space occupied by C0 and C8. A part of C0 is taken from a part of D8, enlarging the area of D10 and creating a street that does not exist in the most detailed level.

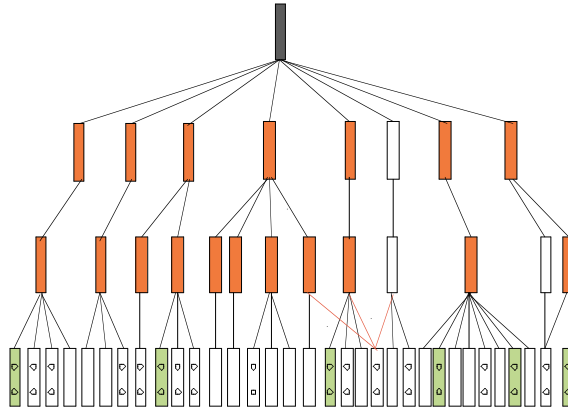


Fig. 7.14: Area Graph of Alsleben I

These two faults in the structure are also reflected in the area graph (Fig. 7.14, red links between areas in level (d) and in level (c)): A single area at level (d) is divided into three different areas at level (c). There is also a problem with the division of areas in D9 which corresponds to the last two areas at level (d) in Fig. 7.14. The white areas are represented in two different areas at level (c), namely in another white area and in a red area. This seems to be an artifact of the symbolization scheme: part of the white area at level (d) is classified as residential area at level (c). It is not clear to the observer why this distinction has been made.

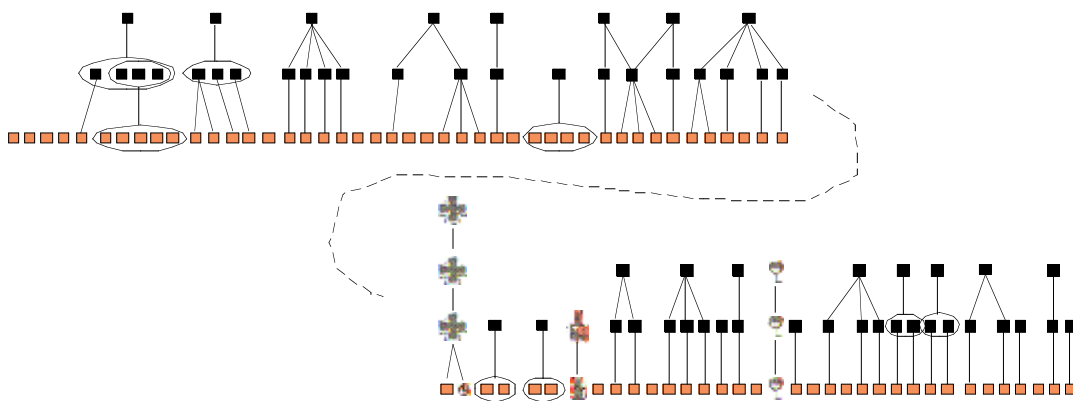


Fig. 7.25: Element Graph of Alsleben I

The element graph (Fig. 7.25) does have one small irregularity: There is one building representation at level (c) that is again divided into two building representations at level (b).

7.6 Study area Alsleben II

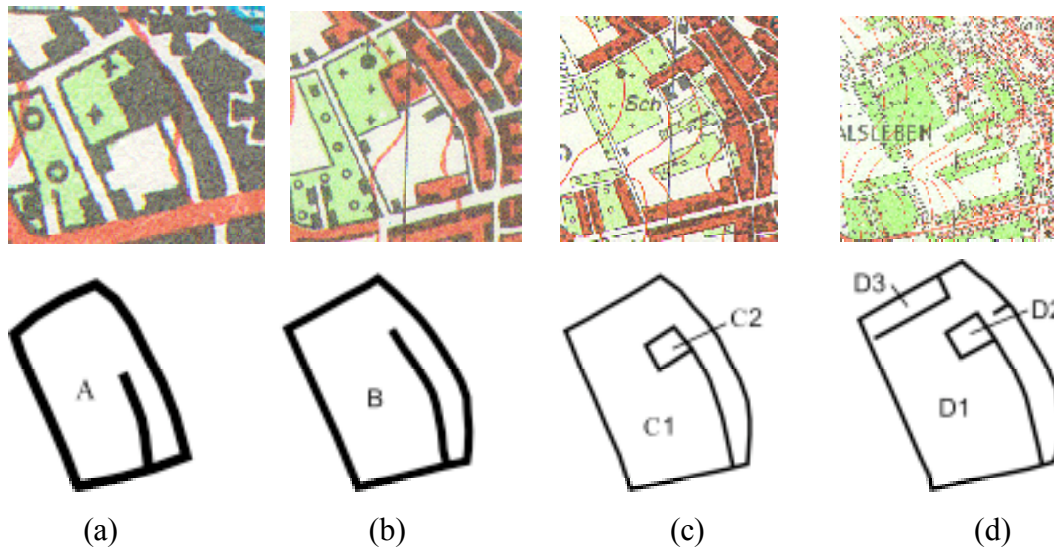


Fig. 7.16: Alsleben II at different levels of detail

The study area is represented by a single container (Fig.7.16a) and bounded by four street segments. There is a street that leads into the area but does not construct a new container. It is represented as a single element in the element graph. The container is preserved at level (b). At level (c) an island emerges as the street makes a loop, and this creates a new container C2. At level (d) a second street creates a container D3. This clip from a map series produces the container tree shown in Fig.7.17. Container B is divided into two containers at level (c) and container C1 is divided into two containers at level (d).

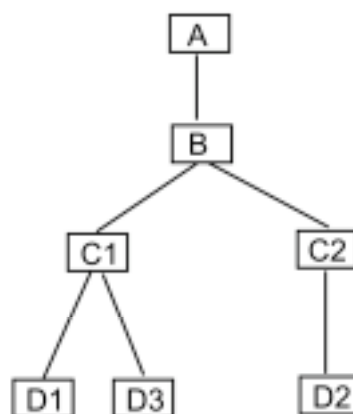


Fig. 7.17: Container Tree of Alsleben II

There are three areas in the container A: a black area, a green area with symbols, and a white area with three buildings. On the next lower level, the black area turns into two red

areas with black rectangles and the white area splits up into a white area and a red area. The green area is preserved. At level (c) the green area is again preserved, the white area is divided into two green areas and one white area. The boundaries of the white area at level B do not correspond to the boundaries of the same area at level (c). The big red area at level B is divided into two red, two green, and three white areas at level (c). At level (d) the red areas disappear and are replaced either by white or by green areas with red house symbols. The shape of the areas changes and some boundaries are added. It is only partly possible to derive areas in level (c) from areas in level (d). Where a direct correspondence cannot be determined, sets of areas are connected in the diagrams through an ellipse instead of straight lines.

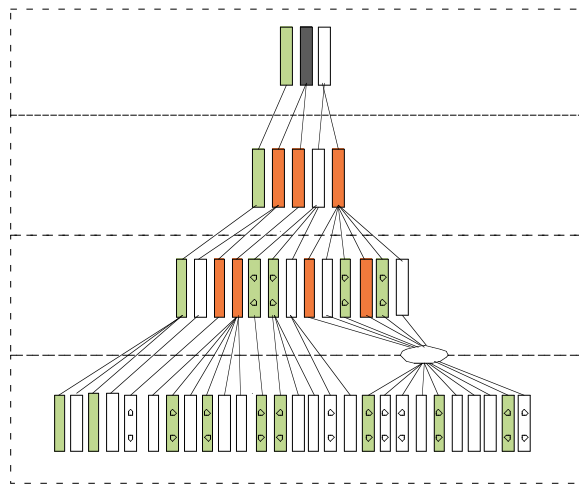


Fig. 7.18: Area Tree of Alsleben II

In the element graph (Fig. 7.19) are three types of elements: Streets, buildings, and a church symbol. In this example there is one irregularity: the dashed line signifies that the element had disappeared on a low level but reappears on a higher level.

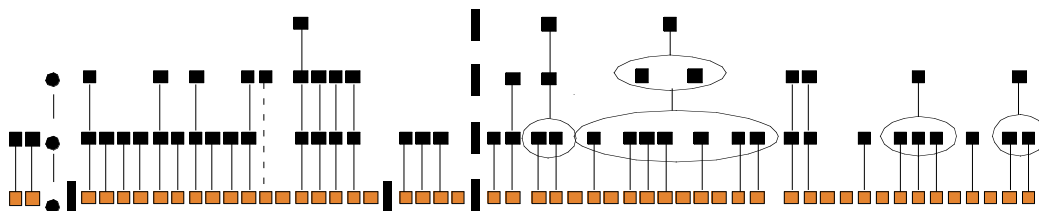


Fig. 7.19: Element Graph of Alsleben II

7.7 Study area Alsleben III

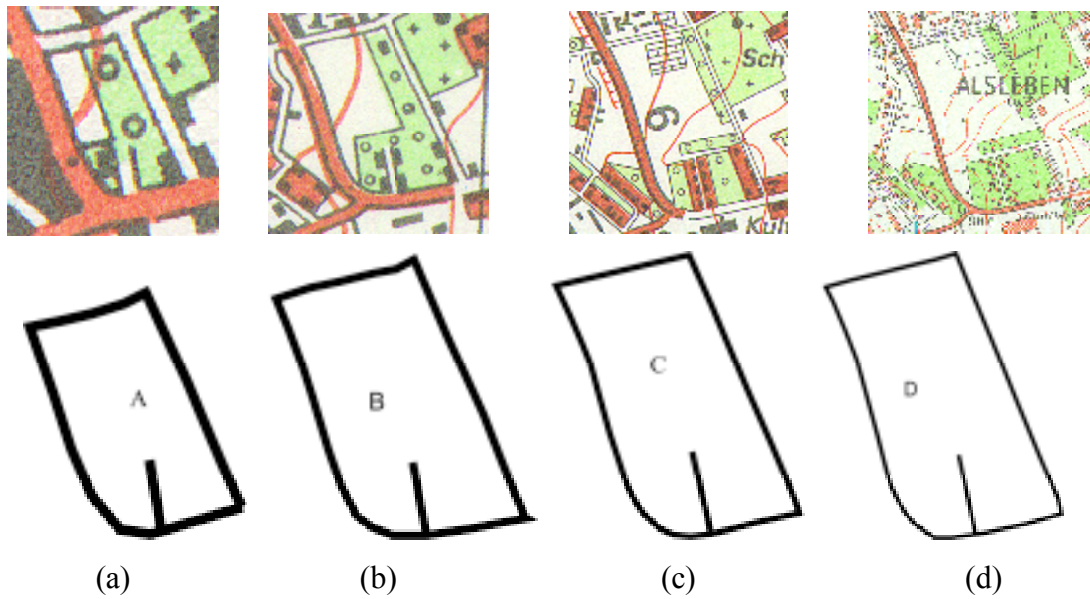


Fig. 7.20: Alsleben III at different levels of detail

This study area (Fig. 7.20) has only one container over the whole range of levels. Container A holds two areas and a street segment. This does not change for container B. Container C consists of more detailed areas and the street segment and container D has even more areas. This particular clip of a map series builds a container tree which has no branching (Fig. 7.21). This is the simplest form of a tree.

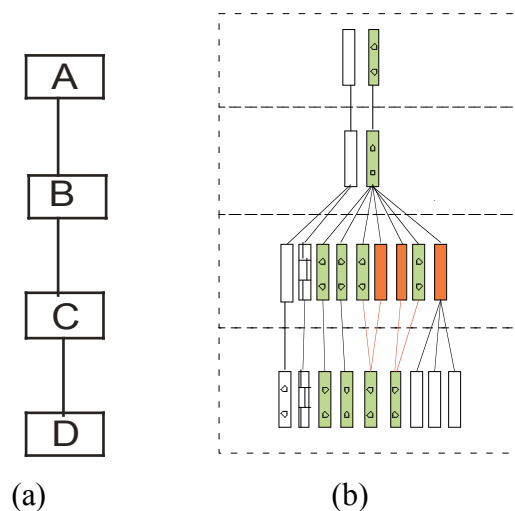


Fig. 7.21: Container Tree (a) and Area Graph (b) of Alsleben III

When building the area graph, we notice that in some parts the links come back together instead of spreading out. This is due to the different symbolizations of built-up areas in levels (c) and (d).

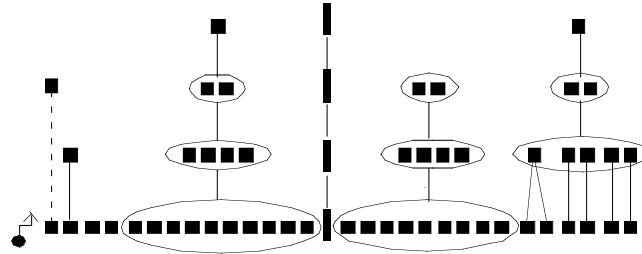


Fig. 7.22: Element Graph of Alsleben III

When two elements are combined to only one, the elements at the lower level are merged to a single element at the higher level. As in the study area Alsleben II, there is a small irregularity, because one element that disappears at level (c) is reintroduced at level (b).

7.8 Study area Murten I

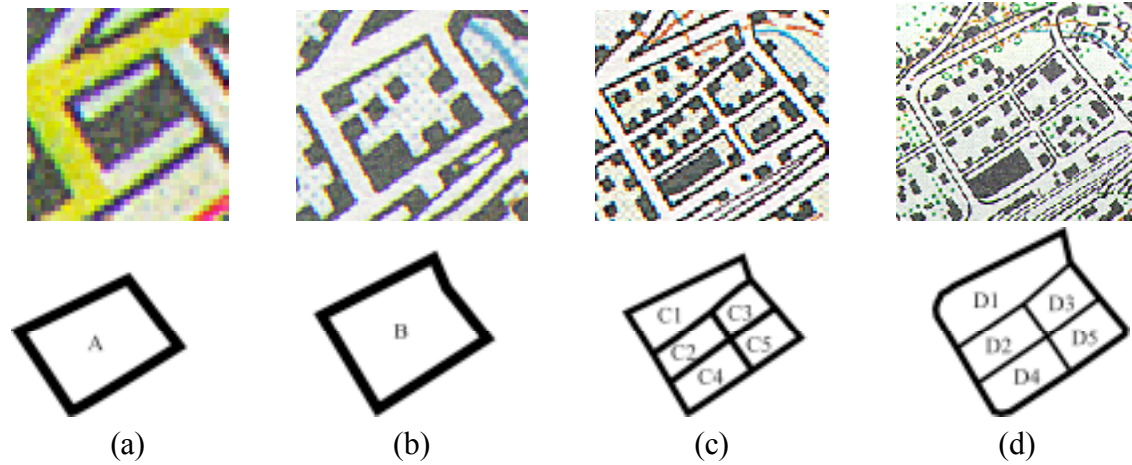


Fig. 7.23: Study Area Murten I at different levels of detail

The division over several levels results in the container tree in Fig. 7.24(a). The area tree (Fig. 7.24 (b)) has the same structure as the container tree.

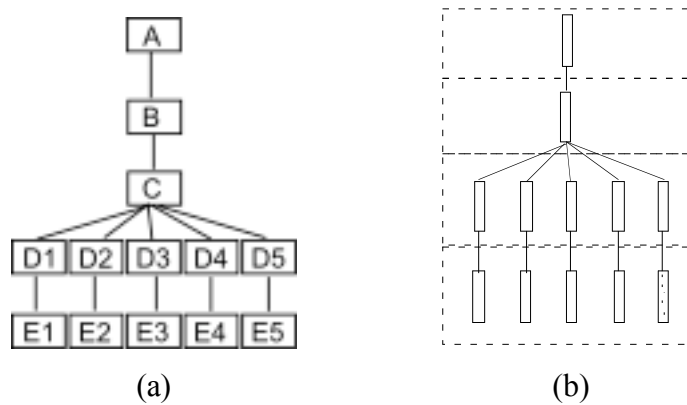


Fig. 7.24: Container Tree (a) and Area Tree (b) of Murten I

There is one element that only appears at level (c) (on the far right of Fig. 7.25). The appearance of this single element is most likely an update problem: The map clip of level (d) was last updated in 1975, whereas the map clip of level (c) was last updated in 1981.

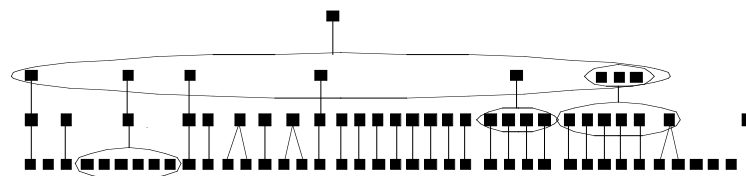


Fig. 7.25: Element Graph of Murten I

7.9 Study area Murten II

The area of interest (Fig. 7.26) is rather irregular and bounded by a railway on one side. It shows the inner part of the small city of Murten. One can notice that the first two levels and the last two levels are more similar in the amount of detail than level (b) compared to level (c).

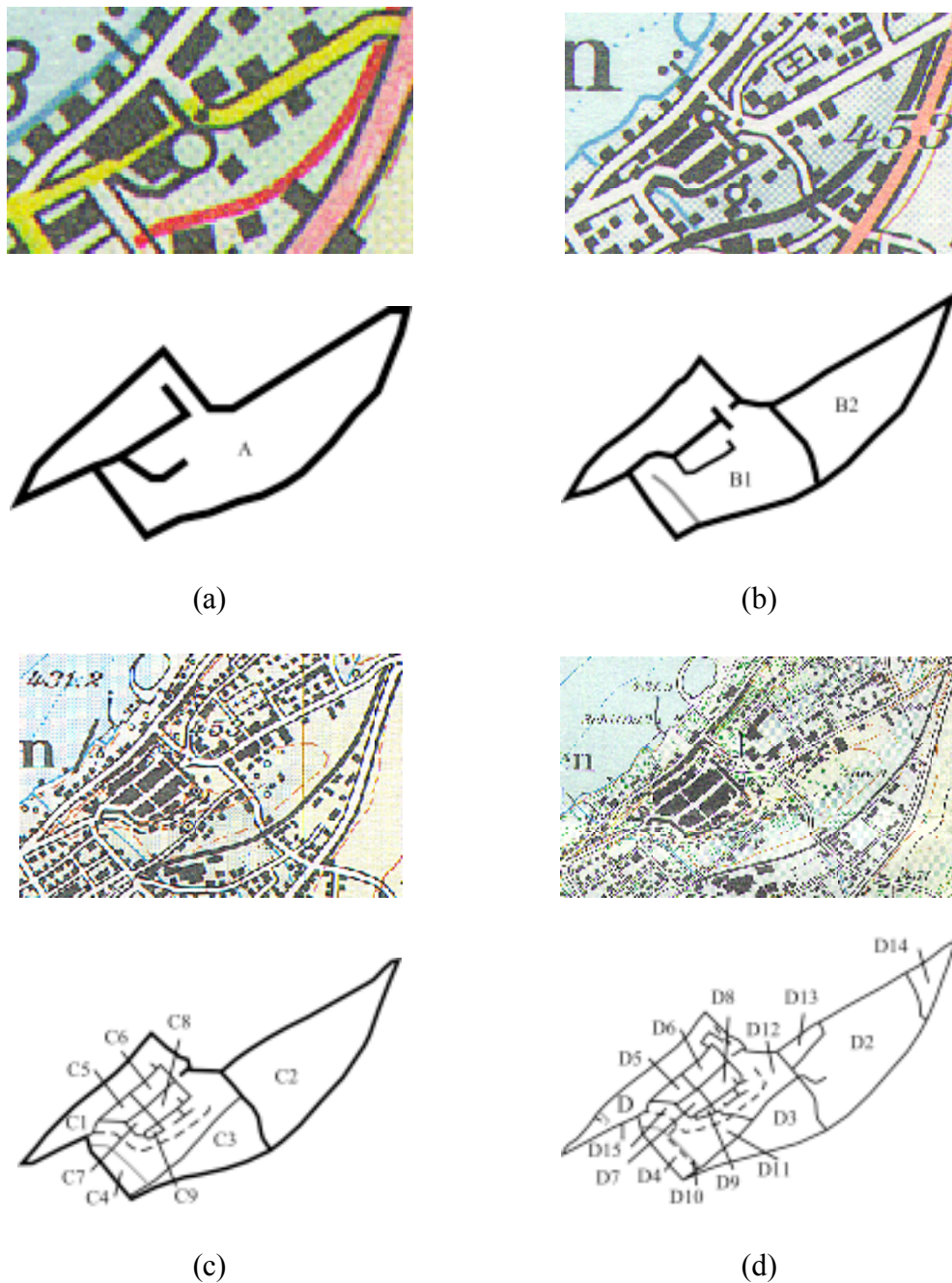


Fig. 7.26: Study area Murten II at different levels of detail

In this example it is not possible to derive a complete area tree, because the central area (the city of Murten) is considered a container, but not bounded by streets. It is not possible to derive from the existing map any criteria that would make a distinction between the black areas representing buildings (belonging to the element graph) and those representing walls surrounding the city (belonging to the area tree).

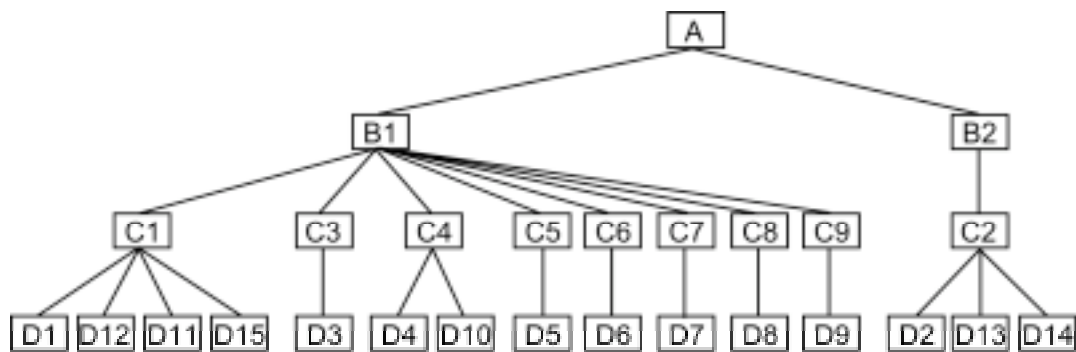


Fig. 7.27: Container Tree of Murten II

The underlying problem is the transition from buildings as elements (in more detailed representations) to building blocks as areas or even containers (in less detailed representations). This is also mirrored by the cartographic tradition to depict buildings in large scale maps in detail and leave the rest of the white space for streets (with or without additional boundaries). In small scale maps, streets are shown with boundaries, whereas only those buildings are depicted that will fit into the white space between streets.

7.10 Summary

In this chapter we presented examples for the construction of the container graph, area graph, and element graph. We used clips from two different map series for the examples. It was possible to construct graphs for all but the last study area. Most study areas have irregularities, that made it impossible to build forests or trees. Each of these irregularities will be analyzed in chapter 8.

Chapter 8 - Analysis and Formalization

Although there were irregularities in the map data, it was possible to build the graph structures as described in chapter 5. The irregularities in the data resulted in deviations from the tree or forest structure in the graphs. The reason there were irregularities is that the map data is not consistent over the four levels of detail. We discuss the issue of consistency in section 8.1.

Each of the components of the map cube model is analyzed and formalized in section 8.2 through 8.5. In section 8.6, we formalize the map cube model, describe the relations between the graphs in the model, and discuss the data structure resulting from our model. Section 8.7 summarizes this chapter.

8.1 The Issue of Consistency

One goal of this work is to derive a data structure from map series suited for the representation of map data in a database. A database needs to be consistent, because users will expect that the answers they receive do not contradict one another. We use the term consistency to describe the absence of contradictory information in the database, that is, in the map cube model.

The map cube model is consistent if all graphs of the model are forests. In particular, we define the following consistency constraints:

- the edges of the trans-hydro network are classified such, that they build a partition on each level of detail,
- the partition of containers at a level n is a refinement of the partition of containers at a level $n-1$,
- the partition of areas at a level n is a refinement of the partition of areas at a level $n-1$,
- the partition of areas at a level n is a refinement of the partition of containers at that level n ,
- every element on level n has a representation on a lower level $n+1$, unless level n is the most detailed level.

We already know that an analogue map series is not made to be consistent: the level (a) map is not derived from the level (b) map but usually from the most detailed map

(SGK, 1987). Therefore, we expected contradictory situations in our case studies from one level to the next level. These inconsistencies in the map data result in deviations from a tree or forest structure in the graphs. The deviations in the structures may be cycles or cross-edges yielding cyclic graphs (cf. Chapter 3), because they violate the transitivity property of trees. Here is a list of the deviations and their possible reasons we found in the case studies:

- Cross-edges between red and black areas exist, because the distinction between red areas (residential) and black areas (built-up) is not consistently applied (cf. 7.2).
- An edge in the container tree is artificially introduced resulting in a cross-edge between two trees. A street has been interpreted as dead-end street at one level and as a continuing street at another level (cf. 7.3).
- An object disappears at one level and reappears at a higher level. The importance of the objects has been differently judged at the two levels (cf. 7.3).
- A cyclic structure in the container tree results from inadvertently introducing a street and dropping another street (cf. 7.5).
- Cross-edges in the area tree result from different symbolization schemes. The built-up area is symbolized as red area on green area at one level and as houses on green area at another level (cf. 7.7).
- A single object appears at one level and disappears at the next higher level. This is an update problem - the two maps date from different years (cf. 7.8).
- The transition from buildings with a wall surrounding them (in more detailed representations) to a city as a whole (in less detailed representations) cannot be reflected in our model (cf. 7.9)

Inconsistent use of rules for land-use classifications or differing interpretations of the spatial situation yield cycles or cross-edges in the graphs. Other deviations from the tree structure are caused by the symbolization schemes or by the long update cycle of maps.

8.2 *Trans-Hydro Network and Trans-Hydro Graph*

The trans-hydro network is a map object where the network is a combination of the topological edges of the street network, the railway network, and the network of the hydrographic objects (see Table 8.1, line 2). We assume in our model that each intersection, bridge, over- and underpass, etc., of the lines marks the beginning and the end of each edge. That means that the appropriate vertices are included in the trans-hydro network.

The trans-hydro network of the most detailed level is a superset of the trans-hydro networks of the less detailed levels. In the process of changing levels of detail from more detailed to less detailed, some edges are filtered: the network has less edges and vertices. Thus, the trans-hydro network is a filter hierarchy.

We need two different classes to define the trans-hydro network formally. The class *TranS* provides us with the superset of all networks in a map series and calculates the total list of edges without duplicates (lines 5,7 of Table 8.1). The second class *TraS* creates a map object (line 14) and calculates the graph of the trans-hydro network for a certain level of detail (line 17). A classification of the edges determines which of the edges are part of the network.

```

1   class TranS t where
2     createTrans :: Int -> [Street] -> [Rail] -> [Hydro] -> t
3     tEdges :: t -> [DEdge]
4     getLev :: Lod -> t -> [DEdge]
5   data Trans = TN Int [DEdge]
6   instance TranS Trans where
7     createTrans i slist rlist hlist = TN i (nub
8       ((concat (map lEdges slist))
9        ++(concat (map lEdges rlist))
10       ++(concat (map lEdges hlist))))
11    tEdges (TN i list) = list
12    getLev i t = filter (hasLev i) (tEdges t)

13  class TraS t where
14    createTran :: ID -> Name -> Lod -> Trans -> t
15  type Tran = Mob
16  instance TraS Tran where
17    createTran i name l tn = Mob i 'T' name l g where
18      g = dGraph i (getLev l tn)
19      (nub(concat(map vertices (getLev l tn))))

```

Table 8.1: Gofer code for the classes *TranS* and *TraS*

In our formalization, each of the edges has a level of detail. This level of detail corresponds to the highest level of detail where the edge is still represented on the map. The edges may be classified e.g. by importance or by administrative rules (national road classification). The classification corresponds to a hierarchy of classes, each level corresponding to classes with a certain level of detail. For example all streets have a classification of importance, the major roads being at level 1, the local country paths being at level n. The same classification is made for railways and hydrographic objects. Displayed is a list of streets, railroads, and hydrographic objects, that accumulates all edges down to a certain level m, determined by the desired level of detail.

The trans-hydro network over several levels forms a graph (cf. Chapter 5), called trans-hydro graph. The graph has a special form: it is a forest, that is, a collection of trees. When a tree reaches its root, the edge disappears from the representation. It is a structure that filters a set of objects from a larger set of objects. The larger set is at a more detailed level (Table 8.2). Less important objects (by definition) disappear from the representation. Distinct levels (in our case levels (a) through (d)) can be stored separately, but it is more economic to filter the objects when needed (Table 8.1, line 11). The filtering criterion is rather simple: If the level of detail of the edge under consideration corresponds to the selected level of detail then the edge will be represented. We do not need a special data structure for the trans-hydro graph, because we can determine the objects to include at a certain level of detail by filtering.

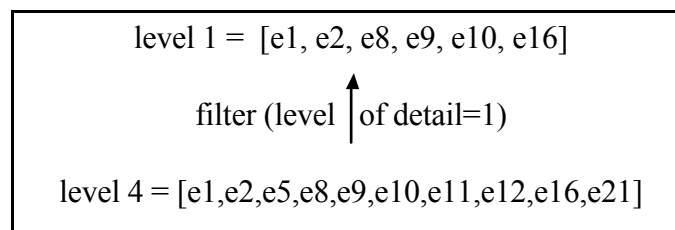


Table 8.2: Filtering a list of edges

The trans-hydro graph has the structure of a filter hierarchy. The importance function of an edge is the filter criterion. The hierarchy generated by this function selects more important edges the higher the level (higher = more important). It is equally possible

to have a function ‘length of edge’ or ‘expected speed average of edge’ (Car, 1996) as filter criterion, depending on the application.

There is a direct relation between the trans-hydro graph and the container tree: when an edge disappears, adjoining containers are aggregated (see section 8.6). The container tree is described in the next section.

8.3 Containers and the Container Graph

A container is a map object (*Mob*) with an identifier (*ID*), a code (*Code*), a name (*Name*), a level of detail (*LOD*), and a directed graph (*Dgraph*; see Table 8.3, line 2).

```
1   type Con = Mob
2   data Mob = Mob ID Code Name Lod DGraph
3   c1 = createMob 1 'C' "A0" 1 g11
```

Table 8.3: Gofer code of a Container

Name is (by our convention) the letter of the level of detail followed by a number. The name of a container can also be interpreted as a “Bezirk” or neighborhood with a special name. The code is always ‘C’ for container. The directed graph is a list of vertices and edges specifying the boundary of the container.

The set of containers of a map describes the coarse structure of the map; a first approximation. The boundaries of the containers are edges of the trans-hydro network. The set of containers of the map constitutes a partition or irregular tessellation of map space. The set of all containers at a level of detail is called a partition of map space if the containers are jointly exhaustive and pairwise disjoint (see also section 3.3).

In a map series, containers at different levels are related by aggregation: A set of containers at a low level of detail aggregates to a single container at a higher level of detail. Thus the container tree has the structure of an aggregation hierarchy. The disappearing edges of the trans-hydro network control which of the containers are aggregated, i.e., any time an edge disappears, adjoining containers are aggregated (see section 8.6).

The structure of the container graph is a tree, called container tree. Vertices in that tree are containers, and edges represent an aggregate function. Each subgraph of the container graph is also a tree: a container at level n is a list of containers at level $n-1$. The

edge function is always the same for the container tree, so it is sufficient to store the vertices, that is, the containers themselves.

The container tree is defined as a recursive data type in Gofer code. This definition of the tree corresponds to a Rose Tree (c.f. chapter 3):

```
data Tree Con = T Con [Tree Con]
```

The container tree (*Tree Con*) is built from a container (*Con*; see above) and a list of other container trees (*[Tree Con]*). The leaves of the container tree are container trees with an empty list of subtrees: *T Con []*. Each container tree is assigned a level of detail that corresponds to the level of detail of the root container.

<pre>ct1 = T c1 [ct2, ct3] ct2 = T c2 [ct4] ct3 = T c3 [ct5, ct6, ct7] ct4 = T c4 [ct8] ct5 = T c5 [ct9] ct6 = T c6 [ct10] ct7 = T c7 [ct11] ct8 = T c8 [] ct9 = T c9 [] ct10 = T c10 [] ct11 = T c11 []</pre>	<pre>--oA -----oB1 -- -----oC1 -- -- ---D1# -----oB2 -- -----oC2 -- -- ---D2# -- -----oC3 -- -- ---D3# -- -----oC4 -- -- ---D4#</pre>
--	---

Table 8.4: Example for a Container Tree in Gofer

The example in Table 8.4 is the formalized container tree of the study area Berneburg. *c1* through *c11* are predefined containers, *ct1* through *ct11* are the corresponding container trees. *ct8* through *ct11* are leaves, *ct1* is the root tree. The right side of Table 8.4 shows the visualized structure of the container tree *ct1*. It shows the name of the container instead of the name of the trees. This visualization corresponds to Figure 7.15(a).

The container tree shows the coarse structure of the map series. A finer structure is offered by the area tree. Areas and the area tree are described in the next section.

8.4 Areas and the Area Graph

An area is a map object (*Mob*) with an identifier (*ID*), a code (*Code*), a name (*Name*), a level of detail (*LOD*), and a directed graph (*Dgraph*; see Table 8.5).

```
1 type Area = Mob
2 data Mob = Mob ID Code Name Lod DGraph
3 a2 = createMob 14 'A' "RE" 2 g21
```

Table 8.5: Gofer code of an Area

The code of an area is always ‘A’ for area. The name is (by our convention) the type of the land-use area. In the countryside, areas have special names like “Zum weissen Kreuz” or “Merlachfeld”. If we chose to use these names for our areas, the type of land-use would have to be added to the area type as a special attribute, or the type map-object would have to be parameterized. We have identified the following land-use types in the case studies (taken from the legend):

- RE: Residential area (red)
- BA: Built-up area (black)
- GA: Garden area (green with circles)
- CA: Cemetery (green or white with crosses)
- NG: Non-designated area with some growth (white with circles)
- ND: Non-designated area (white)

A list of areas describes the map structure in more detail than the list of containers and with different primitives, namely land-use areas. The boundaries between areas are either graphical (color) or explicitly drawn boundaries (fences, cadastral borders). Areas build a partition of map space and in particular of a container.

In a map series, areas construct a general tree, called the area tree. Areas are vertices and a function “generalize” builds the edges. The areas are spatially aggregated if an area becomes too small by itself or too small to contain its objects, and if the two areas have a common generic superclass. The class system for the land-use areas, i.e. the rule system for combining land-use classes, is not documented and could not be derived from our case studies.

In Gofer, the area tree is defined exactly as the container tree: $\text{Tree Area} = \text{T Area} [\text{Tree Area}]$. Each level of the area tree is a partition of map space. The partition at level n is a refinement of the partition at level $n-1$. At the same time, the partition at level n is a refinement of the container partition on the same level n .

Areas show the refined structure of a map, whereas containers give a coarse structure. Areas also contain objects, which are those objects traditionally recognized as

cartographic objects. These objects and their structure over several levels is discussed in the next section.

8.5 Elements and the Element Graph

An element is a map object (*Mob*) with an identifier (*ID*), a code (*Code*), a name (*Name*), a level of detail (*LOD*), and a directed graph (*Dgraph*; see Table 8.6). The code of an element is always an 'O'. The name can be used for labeling purposes. The directed graph may be either a polygon or a single vertex. There are three types of elements: buildings, single symbols, and dead-end streets.

```
1      type Element = Mob
2      data Mob = Mob ID Code Name Lod DGraph
3      o24 = createMob 51 'O' "o24" 3 go24
```

Table 8.6: Gofer code of an Element

Elements may be considered the only traditionally cartographic objects in this model. Even the trans-hydro network is only the topological skeleton of the network of streets, railways, and hydrographic objects.

The element graph is different from the two previously described graph structures. It does not represent a partition of map space. It describes how elements or a set of elements relate to other elements at a lower or higher level of detail. It gives information about the

- merging of neighboring elements, the
- omission of elements (filtering), the
- typification of groups of elements (ellipses in the case studies), and the
- continued representation of elements.

The vertices in the element graph represent the elements. The edges are associated with one of the four functions *merge*, *omit*, *typify*, and *continue*. In contrast, in the container tree and the area tree a single function is associated with the edges, namely *aggregate* and *generalize* respectively.

The element graph is a combination of the generalization and the filtering hierarchies. The *merging* of buildings, for example, is a generalization of the class building with an error x to the class building with an error y , where $y > x$. Each time an element is *omitted*, it is in

fact filtered (by importance) and does not pass the filter. The *typification* of a group of elements is a generalization between groups of individuals. The operation *continue* corresponds to the filter operation, where the element passes the filter.

The structure of the element graph is a list of non-connected trees. This structure is called a forest (section 3.2) in graph theory. The connection between the trees in the forest is the knowledge about the levels and (if a metric is introduced) the spatial relations between the elements. However, the trees are indirectly indexed through the containing areas.

8.6 The Map Cube Model

The map model incorporates a trans-hydro network, a set of containers, a set of areas, and a set of elements. This corresponds to an x-y-plane of the map cube model. A map series adds the level of detail as the third dimension. We now describe the map series that incorporates the trans-hydro network graph, the container graph, the area graph, and the element graph.

Each graph describes a different object of the map over n levels of detail. The width of the graphs depends on the complexity and density of the map space they represent. Therefore, no fixed width can be assigned to the graphs. The depth of the graphs corresponds to the amount of levels of detail that are given. Visually, a map series corresponds to a combination of four cubes: one for each graph.

The cube for the trans-hydro graph can be flattened to the lowest level of detail, because the other levels of detail can be calculated from the most detailed set of trans-hydro objects. We reason in the trans-hydro cube from low level to high level by filtering the superset with the correct level of detail. We reason in the container cube by aggregating containers and in the area cube by generalizing land-use classes. Reasoning in the element cube is done by typifying, omitting, merging, or continuing.

In Gofer, the structure of the interrelated cubes is hard to realize. We chose to specify a list of containers. Those containers consist of a list of areas; the areas contain a list of elements. The trans-hydro network is given at the lowest level of detail. The

information in the area graph and in the element graph cannot be directly connected to the tree of containers. Therefore, we have to add them specifically to our model (Table 8.7, line 4).

```

1 class MapDBS mdb where
2   createMDB :: Trans -> (Tree MCon) -> (Tree Area) ->
3               [Tree Element] -> mdb
4   getMap :: mdb -> Lod -> Name -> Mappa

5 data MapDB = MDB Trans (Tree MCon) (Tree Area) [Tree Element]

6 instance MapDBS MapDB where
7   createMDB t tm ta tob = MDB t tm ta tob
8   getMap (MDB t tm ta tob) lod n1 = TL (createTran lod n1 lod t)
9                                     (filter (hasLev lod) (nodes tm))

```

Table 8.7: Gofer code for a map database

A map database (*MapDB*) is composed of independent but interrelated elements. *Trans* is the superset of the trans-hydro network. *Tree MCon* is the container tree with edges to the embedded areas and elements. *Tree Area* is the area tree and *[Tree Element]* is the list of element trees that comprise the element graph.

The function *getMap* extracts a single map from the *MapDB*. It requires a map database, a level of detail, and a name for the map. It returns the trans-hydro network *Tran* and a list of map-containers *MCon* at the specified level of detail. The map-containers *MCon* include the list of map-areas *MArea* and their lists of elements.

8.6.1 Relations between Graphs

All four graphs are related to each other. The trans-hydro network corresponds to the boundaries of the containers at each level. The areas are a refined version of the containers and the elements are contained in the areas.

The trans-hydro network *borders* the containers of a map. If an edge in the trans-hydro network is left out, then the adjoining containers are aggregated to a single container. The boundary of the new container is the sum of all edges of the original containers minus the edges that were left out. The new spatial distribution must again be a partition of map space.

A single container *contains* a least one area. The list of areas at a given level of detail forms a partition of map space. This partition is a refinement of the partition of containers

at the same level. This relationship between containers and areas must hold at all levels. Then the property ensures that the topological relationships of containment and neighborhood are consistent over all levels of detail.

A single area *contains* zero to n elements. Buildings can be merged to the point where they describe a whole area. Then the building representations are “swallowed” by the area representation. This behavior justifies that the area representation is a sequel to the building representations.

8.6.2 Recommended Data Structure

One of the goals of this work is to recommend a data structure for the representation of map series in databases. We have shown in this chapter that the deviations in the graph structure of the case studies are due to the inconsistencies in map series. The graph structures completely describe the data of the map series. Thus, the map cube model is a good model for map series. We recommend to use the map cube model as a data structure for map series.

The model can be improved if we re-incorporate the map elements intentionally excluded from the model (see chapter 5). Information on the geomorphology of the space depicted in the map (e.g., ridges, faults, etc.) helps to guide the process of cartographic generalization. It would be beneficial to incorporate that structural information into our model.

Our model is structured such that the automated acquisition of data in map series ought to be possible. An importance criterion for the hydrographic objects, roads and railways must be given in order to derive the levels with less detail. Then the containers and the areas can be determined automatically. How elements at different levels are related is still an area of considerable research. However, we narrowed down the amount of elements per level and per area to a few types easily recognized, namely buildings, symbols, and dead-ends.

8.7 Summary

We have shown in this chapter that the deviations in the graph structures of the case studies are due to the inconsistencies in analogue map series. The graph structures completely describe the data of the map series in a consistent manner. Databases need to be consistent. Thus, the map cube model is a good model for map series. We recommend to use the map cube model as a data structure for map series.

We have shown that a map series can be seen as a combination of four entities: a trans-hydro network graph, a container graph, an area graph, and an element graph. Each of these four entities is a special hierarchy. The trans-hydro graph is a hierarchy that filters levels of objects from a superset of objects. The container tree is a hierarchy that aggregates its objects to form a new level. The area tree is a hierarchy that reclassifies its objects according to a hierarchy of generic land-use classes. The element graph is a combination hierarchy that filters and generalizes objects according to graphical criteria.

The data structure for the map database is a combination of the trans-hydro network, a tree of containers including the information on areas and elements, an area graph, and an element graph. The area graph and the element graph need to be represented separately in the model. The tree of containers stores only which container includes which areas and elements. It does not store how these areas and elements are related to areas and elements at a lower or higher level of detail. This information is only encoded in the area graph or element graph, respectively.

Chapter 9 - Results and Future Work

This chapter first summarizes our research. The results of our investigation are a formal map model and a formal map cube model for a map series. Limitations and advantages are discussed in the subsequent section. Finally, we present our ideas on how to continue research on this topic.

9.1 Summary

In this research we investigated how to link map objects at different levels of detail that represent the same real-world entity or a set of real-world entities. We proposed a hierarchical data structure that supports and describes the behavior of map objects over four levels of detail using these links.

We defined several types of hierarchies with their respective properties and their methods to change levels. Hierarchies are defined according to the functions associated with their edges: aggregate, generalize, and filter. The aggregation hierarchy is built by aggregating objects. The generalization hierarchy defines how classes relate to more generic superclasses. The filter hierarchy filters objects according to a filter criterion. We identified for which structures and tasks these hierarchies can be used. This is important for the design of databases for multiple representations.

We described our theory on maps and map series, because every investigation is theory-dependent. A map is composed of four types of map objects:

- a trans-hydro network, that is a combination of the street network, the railroad network, and the hydrographic network,
- containers, that represent the regions between the links of the trans-hydro network,
- areas, that represent the land-use areas and are contained in the containers, and
- map elements, that are contained in the areas, representing buildings, symbols, and dead-end streets and railroads.

One can imagine the structure of the map series, that is the *map cube model*, as a three dimensional composition. Each map is a level in a three-dimensional cube, the vertical

axis represents the level of detail. The hierarchical data structure for the map series is a combination of the trans-hydro network, a tree of containers including the information on areas and elements, an area graph, and an element graph. The area graph and the element graph need to be represented separately in the model. The tree of containers stores only which container includes which areas and elements. It does not store how these areas and elements are related to areas and elements on a lower or higher level of detail. This information is only encoded in the area graph or element graph, respectively.

We investigated the behavior of objects over four levels of detail in eight case studies (chapter 7). We applied an observation method derived from the method of abduction. It relies on careful and controlled observation of phenomena, similar to the method of reverse engineering in computer science.

We observed ten types of map elements in the case studies. The following six are incorporated into our data structure: streets, railways, hydrographic objects, buildings, areas, and symbols (Table 9.1). Streets, railways, and hydrographic elements are linear structures on the map, that are combined to a trans-hydro network. Buildings and single symbols are map elements contained in areas. Areas correspond to land-use classes. A container is a region between the lines of the trans-hydro network.

Map elements	Map	Map series
streets	trans-hydro network	trans-hydro graph
railways	trans-hydro network	trans-hydro graph
hydrographic elements	trans-hydro network	trans-hydro graph
regions (between links of trans-hydro network)	containers	container graph
land-use classes	areas	area graph
buildings	map elements	element graph
symbols	map elements	element graph

Table 9.1: Map elements

This structure represents a model of a map, that is hierarchical with respect to the inclusion of elements. Containers build the largest partition of map space and contain areas.

Areas compose a refinement of this container partition and contain elements. Finally, elements are contained in the areas.

In chapter 8, we analyzed the structures derived from the case studies and discussed the issue of consistency. We formalized the map model and the map cube model in a functional programming language with an object-oriented paradigm. The formalization was tested with one of the case studies.

The map cube model is consistent if all graphs of the model are forests, that means that there are no derivations in the graphs. Inconsistent use of rules for land-use classifications or differing interpretations of the spatial situation yield cycles or cross-edges in the graphs. Other deviations from the tree structure are caused by the symbolization schemes or by the long update cycle of maps. Our data structure is consistent if

- the edges of the trans-hydro network are classified such, that they build a partition on each level of detail,
- the partition of containers at a level n is a refinement of the partition of containers at a level $n-1$,
- the partition of areas at a level n is a refinement of the partition of areas at a level $n-1$,
- the partition of areas at a level n is a refinement of the partition of containers at that level n ,
- every element on level n has a representation on a lower level $n+1$, unless level n is the most detailed level.

In our model the trans-hydro network is represented by a filter hierarchy, the filter criterion is the relative importance of the link in the network. The container tree is an aggregation hierarchy - the more abstract the representation, the more containers are aggregated. The area graph is a collection of area trees; each tree represents a generalization hierarchy. The element graph is a collection of trees; each tree is a constructed by several functions. Those functions may be 'omit', 'typify', 'continue', and 'merge'. The first three functions correspond to the cartographic functions with the same name. Thus, the model incorporates all three types of hierarchies.

9.2 Results and Conclusions

The main result of our investigation is the definition of a *formal map model* and a *formal map cube model* for a map series. The model incorporates four different hierarchical data structures that support and describe the behavior of map objects. The final data structure is a combination of three different types of hierarchies.

Our hypothesis was that several types of hierarchies are superimposed in a map series. Hierarchies are necessary for multi-scale databases. The main reasons are that hierarchies reduce processing time, increase the stability of an information system, and break down the tasks into manageable portions. We distinguished hierarchies by their functions and defined them formally. Table 9.2 relates the type of hierarchy to the type of graph in the map cube model. We found that the combination of three different types of hierarchies results in an adequate structure for a map series.

Type of graph	type of hierarchy
trans-hydro network	filter hierarchy
container graph	aggregation hierarchy
area graph	generalization hierarchy
element graph	combination of filter and generalization hierarchy

Table 9.2: Type of hierarchies and Graphs

Another result of this research is the definition of consistency constraints for a digital map series. The map cube model is consistent if all graphs of the model are forests. A database needs to be consistent, because users will expect that the answers they receive do not contradict one another. We use the term consistency to describe the absence of contradictory information in the database, that is, in the map cube model. We conclude that the hierarchical data structure is only possible if the data given in the map series is consistent.

As shown, all deviations that arise in the structure of the graphs (yielding cyclic graphs) are caused by differing graphic or structural interpretations of the situation in different maps. This result is important in the context of acquiring cartographic knowledge for representation in a GIS. It shows that the data repository 'map series' must be carefully checked for inconsistencies before acquiring knowledge for a database.

Last but not least, we contributed a new method for the acquisition and formalization of cartographic knowledge. In empirical studies it is necessary to define a theory on the subject under investigation, because each observation is theory dependent. Next, an observation plan must be defined and the observations carried out under controlled conditions, similar to psychological experiments. The observations will refine and sometimes change the theory. The result of the observations is a proven model of the theory. As a last step of the method the model is formalized.

9.3 Discussion

The map element ontology given in chapter five is only valid for the scale ranges of our map series, that is for maps with scales from 1:10,000 to 1:200,000. The symbolization scheme of maps changes between the scales 1:200,000 to 1:500,000 rather drastically. The change is due to the change in the purpose of the maps (e.g., from topographic map to route map). Our research has shown that map elements can be assigned to four formal map objects. We believe that an enlargement of the scale range will not add map elements to the original list and thus change our model. However, a second case study, perhaps only of the legends, is necessary to prove this belief. An ontology of map elements is useful to determine what data to acquire for a multi-level database.

A collection of buildings that is divided by streets can not be recognized as a coherent structure, that is as a town or village, in our present model. We propose to solve this pattern recognition problem by introducing a new map object for administrative regions. This map object is used as a constraint on the container and area graphs. The knowledge about the administrative regions of a town is necessary for labeling and symbolization purposes.

Our model can be improved if we re-incorporate the map elements intentionally excluded from the model. Information on the geomorphology of the space depicted in the map (e.g., ridges, faults, etc.) divides space in a way similar to the trans-hydro network. It is beneficial to incorporate this structural information into our model, because it constrains the functions in the element graph (e.g., two buildings cannot be merged if they are on the other sides of a ridge).

Our model is structured such that data in map series can be acquired automatically. The relative importance of hydrographic objects, roads and railways must be determined beforehand. Data at a lower level of detail can be derived from this original trans-hydro network. The container tree can be determined automatically. How elements of different levels are related is still an area of considerable research. However, we narrowed down the amount of elements per level and per container to a few types that are easily recognized.

The data structure can be applied to solve changes of the level of detail. These changes are required for the typical tasks of zooming and panning in Geographic Information Systems. Our data structure provides the necessary data for the display of more detailed information.

9.4 Future Work

We derive from the conclusions and the results the following list of research topics:

A case study with maps at scales smaller than 1:500,000 will provide us with another map element ontology. We want to prove that an enlargement of the scale range will not add map elements to the original list and thus change our model. This proof will validate our map cube model for a larger scale range. As a consequence, we will obtain a method to acquire cartographic knowledge over a large range of scales.

We desire to re-introduce the map elements originally excluded into the model. Especially embankments, ridges, canyons, etc. are necessary to describe the behavior of map objects. We need to research how to incorporate the geomorphology into the model.

In our model we distinguished between linear hydrographic objects for the trans-hydro network and differently shaped hydrographic objects such as ponds and lakes for the element graph. Can this classification be carried out automatically?

In our model we use three types of object-hierarchies for the map objects. The model is designed in a way that automatic derivation of the hierarchies is possible. From the trans-hydro network the container hierarchy can be derived. Areas inside a container behave according to certain rules for land-use classification. How can we automate the process of creating the hierarchies for our map cube model?

For the acquisition of cartographic knowledge it is interesting to analyze statistically the amount of trees per map. Does the knowledge about the amount of objects per level contribute to the knowledge about the structure of the map? Can we derive a map density measure from, e.g., the amount of elements in each area and the size of areas in containers? Does the width of the tree express facts about the structure of the area under investigation?

The main structure of our map cube model are graphs or more specifically trees. How can we implement an efficient data structure for the container trees, the area trees, and the element trees? Gueting has proposed a model for the representation and the querying of graphs in standard databases (Gueting, 1994). His model is especially suited for spatially embedded networks, as for example the trans-hydro network. Can we use this model for the hierarchical graphs in our model?

Our multi-level data structure lends itself to dynamic level changes as they are required for zooming in panning in Geographic Information Systems. However, those changes from one level of detail to the next are discrete changes that may confuse the user of a GIS. Can this effect be mitigated? Or does the abrupt change help the user? Is there a difference between novice and expert users?

We proposed a data structure that is 'layered', that means that each map is still represented in the complete data structure. The assumption there is that each map object is at the same hierarchical level as its neighbors. For graphic purposes this assumption needs not be true. We proposed to define prototypes of map objects for a certain scale range and to zoom graphically between these prototypes (Timpf and Frank, 1995b). How can we

define prototypes for each of the map elements? How can we determine the scale range of these prototypes?

The map structure may be useful for progressive transmission. Progressive transmission of vector data still presents a problem. In the case of maps, first the trans-hydro network from the least detailed to the most detailed network should be transmitted. This information automatically yields the containers. Then the information on land-use from the area tree can be filled in. Finally, the objects in the areas are transmitted. This would transmit the most numerous group of map objects last. How can we implement an algorithm for progressive transmission?

Appendix - Gofer code

This chapter presents the complete Gofer code as used in the thesis (sections A.1 through A.6). We also show the example we used to check our code (section A.7).

A.1 Generalities

```
type Name = String
type Code = Char
type ID = Int
type Lod = Int

class Nameable a where
  name :: a -> Name
  nam  :: a -> Name

instance Nameable Int where
  name a = show' a

instance Nameable Char where
  name a = [a]

instance Nameable String where
  name a = a

class Idable a where
  ident :: a -> ID

instance Idable Int where
  ident i = i

class Levelable a where
  level :: a -> Lod
  hasLev :: Lod -> a -> Bool

class StringS a where
  text :: a -> String
  texts :: a -> [String]
```



```

-----
class DGraphS dg where
  dGraph :: Int -> [DEdge] -> [Vertex] -> dg
  vlist  :: dg -> [Vertex]
  elist  :: dg -> [DEdge]
  addE   :: DEdge -> dg -> dg
  addV   :: Vertex -> dg -> dg

data DGraph = DGraph ID [DEdge] [Vertex]

instance DGraphS DGraph where
  dGraph id el vl = DGraph id el vl
  elist (DGraph id el vl) = el
  vlist (DGraph id el vl) = vl
  addV v1 (DGraph id el vl) = case (elem v1 vl) of
    True -> DGraph id el vl
    False -> DGraph id el (v1:vl)
  addE e1 (DGraph id el vl) = case (elem e1 el) of
    True -> (DGraph id el vl)
    False -> (addV (end e1) (addV (start e1)
                                   (DGraph id (e1:el) vl)))

instance Idable DGraph where
  ident (DGraph id el vl) = id

instance Eq DGraph where
  (==) (DGraph id1 el1 vl1) (DGraph id2 el2 vl2)
    = id1==id2 && el1==el2 && vl1==vl2

instance StringS DGraph where
  text (DGraph id el vl) = "G("++(show' id)++",E"
                          ++(show' (map ident el))
                          ++(show' (map ident vl))
                          ++")"

```

A.3 Map Object

```

class MapobjectS mob where
  createMob :: ID -> Code -> Name -> Lod -> DGraph -> mob
  poly :: mob -> DGraph
  code :: mob -> Code

data Mob = Mob ID Code Name Lod DGraph

instance Idable Mob where
  ident (Mob id1 c1 n1 l1 p1) = id1

instance Nameable Mob where
  name (Mob id1 c1 n1 l1 p1) = n1

instance Levelable Mob where
  level (Mob id1 c1 n1 l1 p1) = l1
  hasLev i m = i==(level m)

instance Eq Mob where
  (==) (Mob i1 c1 n1 l1 p1) (Mob i2 c2 n2 l2 p2) = ((i1==i2)
    && (n1==n2) && (c1==c2) && (l1==l2) && (p1==p2))

instance StringS Mob where
  text (Mob i1 c1 n1 l1 p1) = "Mob(I"++(show' i1)++",C"
    ++(show' c1)
    ++",L"++(show' l1)++",N"
    ++(show' n1)++", "++(text p1)++)"

instance MapobjectS Mob where
  createMob id1 c1 n1 l1 p1 = Mob id1 c1 n1 l1 p1
  poly (Mob id1 c1 n1 l1 p1) = p1
  code (Mob id1 c1 n1 l1 p1) = c1

-----derived types-----
type Con = Mob
type Area = Mob
type Object = Mob

```

A.4 Trans-hydro Network

```

const = 99
-----Linework-----
class LineworkS l where
    createLW :: Name -> [DEdge] -> l
    lEdges :: l -> [DEdge]

data Linework = LW Name [DEdge]

instance LineworkS Linework where
    createLW n list = LW n list
    lEdges (LW n list) = list

instance Levelable Linework where
    level l = foldr min const (map level (lEdges l))

instance Nameable Street where
    name (LW n _) = name n

type Rail = Linework
type Street = Linework
type Hydro = Linework

-----Transport-----
class TranS t where
    createTrans :: Int -> [Street] -> [Rail] -> [Hydro] -> t
    tEdges :: t -> [DEdge]
    getLev :: Lod -> t -> [DEdge]

data Trans = TN Int [DEdge]

instance TranS Trans where
    createTrans i slist rlist hlist = TN i (nub
        ((concat (map lEdges slist))
         ++(concat (map lEdges rlist))
         ++(concat (map lEdges hlist))))
    tEdges (TN i list) = list
    getLev i t = filter (hasLev i) (tEdges t)

class TraS t where
    createTran :: ID -> Name -> Lod -> Trans -> t

type Tran = Mob

instance TraS Tran where
    createTran i name i2 tn = Mob i 'T' name i2 g where
        g = dGraph i (getLev i2 tn) (nub(concat(map vertices
            (getLev i2 tn))))

instance Idable Trans where
    ident (TN i _) = ident i

instance Nameable Trans where
    name t = "TN"++(show' (ident t))

instance StringS Trans where
    texts t = (name t):(map text (tEdges t))

instance Levelable Trans where
    level t = foldr min const (map level (tEdges t))

```

A.5 Rose Tree

```

data Tree a = T a [Tree a]

class (Eq a, Nameable (t a)) => TreeS t a where
  createTree :: a -> t a
  insTree :: t a -> t a -> t a
  delTree :: t a -> t a -> t a
  root :: t a -> t a
  root = id
  node :: t a -> a           -- root node of the tree
  subtrees :: t a -> [t a]  -- the immediate children
  alltrees :: t a -> [t a]  -- lists all subtrees in the tree
  nodes :: t a -> [a]       -- lists all nodes in the tree
  leaves :: t a -> [t a]    -- all leaves in tree
  isLeaf :: t a -> Bool     -- checks if a tree is a leaf
  isLeaf = null.subtrees
  depth :: t a -> Int       -- depth of the tree
  depth = (+1).foldl max 0.map depth.subtrees
  size :: t a -> Int        -- size of the tree
  size = (+1).sum.map size.subtrees
  elemTree :: t a -> t a -> Bool  -- tree element of tree?
  prTree :: t a -> [String]
  drawTree :: t a -> String
  drawTree = unlines.prTree

instance Nameable a => Nameable (Tree a) where
  name (T node _) = name node

instance (Eq a, Eq [Tree a]) => Eq (Tree a) where
  (==) (T n1 t1) (T n2 t2) = ((n1==n2) && (t1==t2))

instance Nameable a => StringS (Tree a) where
  text (T n1 []) = "-"++(name n1)+"#"
  text (T n1 ts) = "--o"++(name n1)---++(concat (map text ts))

instance Levelable a => Levelable (Tree a) where
  level (T n1 _) = level n1
  hasLev i (T n1 _) = i==(level n1)

instance (Eq (Tree a), StringS (Tree a)) => TreeS Tree a where
  createTree n1 = T n1 []
  node (T n1 _) = n1
  subtrees (T n1 ts) = ts
  insTree t2 (T n1 ts) = case (elemTree t2 (T n1 ts)) of
    True -> error "is already element of tree"
    False -> T n1 (t2:ts)
  delTree t2 (T n1 ts) = case (elemTree t2 (T n1 ts)) of
    True -> T n1 ((\\) ts [t2])
    False -> error "tree not element of supertree"
  alltrees (T n1 ts) = (T n1 ts) : concat (map alltrees ts)
  leaves (T n1 []) = [(T n1 [])]
  leaves (T n1 ts) = concat (map leaves ts)
  nodes t1 = map node (alltrees t1)
  elemTree t1 t2 = elem t1 (alltrees t2)
  prTree (T n1 ts) = text (T n1 ts) : map (" |--" ++)
    (concat (map prTree ts))

instance Functor Tree where
  map f (T n1 ts) = T (f n1) (map (map f) ts)

```


A.6 Map and Map Series

```

data TList a b = TL a [b]

class TListS tl a b where
  createTL :: a -> [b] -> tl a b
  ttyp :: tl a b -> a
  tlist :: tl a b -> [b]

instance TListS TList a b where
  createTL a b = TL a b
  ttyp (TL a _) = a
  tlist (TL _ b) = b

instance Nameable a => Nameable (TList a b) where
  name (TL a _) = name a
  nam (TL a _) = ", "++(name a)

instance (Nameable a, Nameable b) => StringS (TList a b) where
  text (TL a b) = (name a)++("++(concat (map nam b))++")
  texts (TL a b) = (name a):(map name b)

instance (Eq a, Eq [b]) => Eq (TList a b) where
  (==) (TL a1 b1) (TL a2 b2) = a1==a2 && b1==b2

instance Levelable a => Levelable (TList a b) where
  level (TL a _) = level a
  hasLev i (TL a _) = i==(level a)

instance MapobjectS (TList Mob Mob) where
  code (TL a b) = code a
  poly (TL a b) = poly a

-----class MapDBS-----
class MapDBS mdb where
  createMDB :: Trans -> (Tree MCon) -> (Tree Area) -> [Tree
Object] -> mdb
  getMap :: mdb -> Lod -> Name -> Mappa

data MapDB = MDB Trans (Tree MCon) (Tree Area) [Tree Object]

instance MapDBS MapDB where
  createMDB t tm ta tob = MDB t tm ta tob
  getMap (MDB t tm ta tob) lod name = TL (createTran lod name lod
t)
                                (filter (hasLev lod) (nodes tm))

-----map structure-----
type MArea = TList Area Object
type MCon = TList Con MArea
type Mappa = TList Tran MCon

```

A.7 Example

We use the area Berneburg I (cf. 7.2) from our case studies as example. Fig. A.1 shows the map clips of Berneburg I.

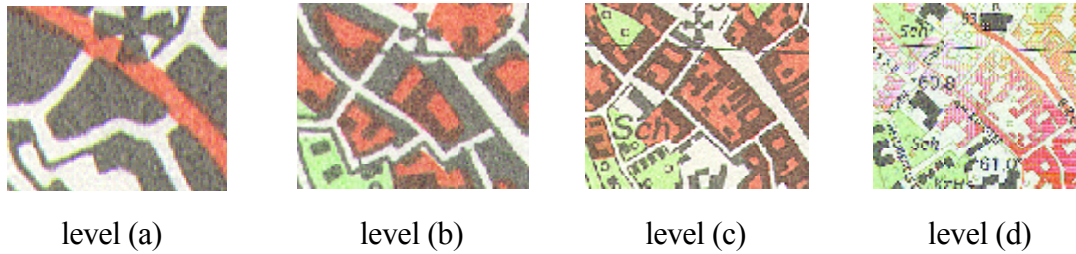


Fig. A.1: Map series Berneburg I

Fig. A.2 shows a simplified version of the map clip.

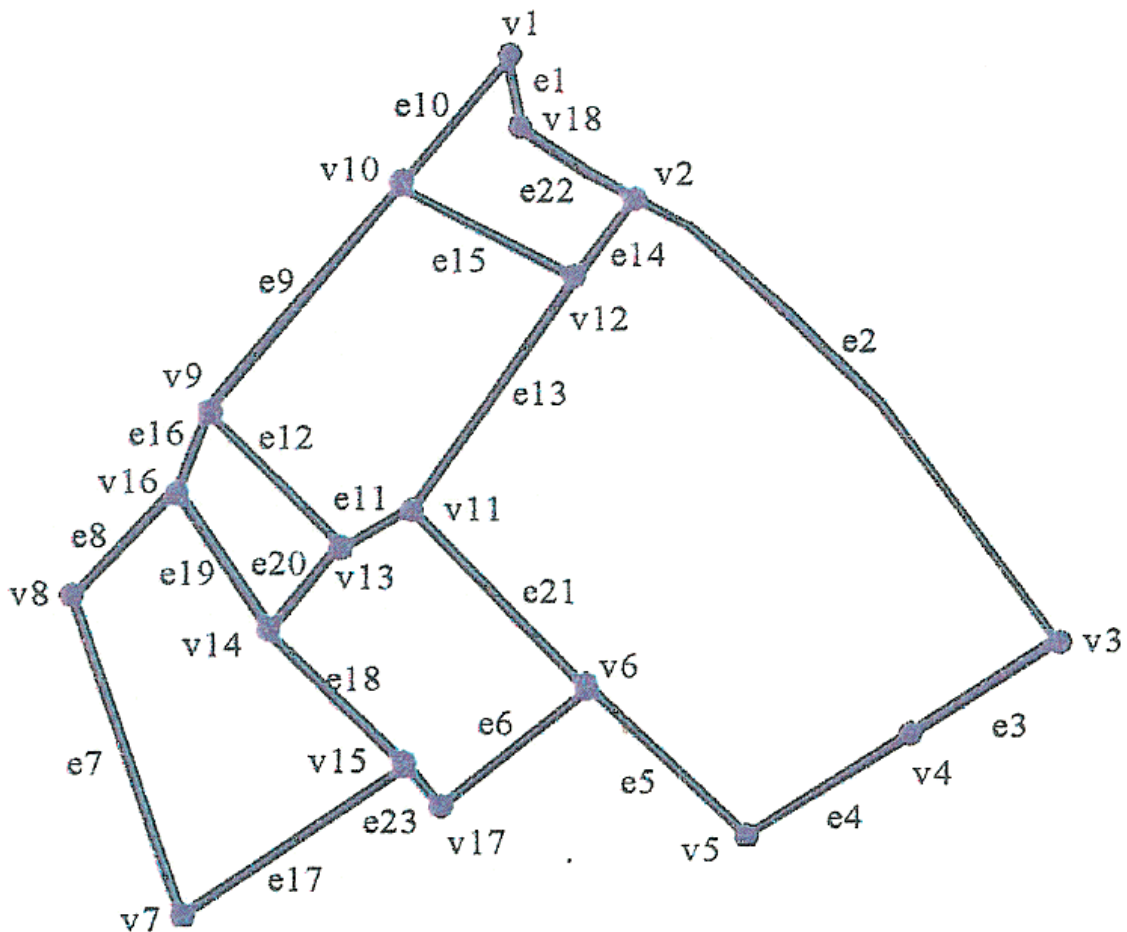


Fig. A.2: Graph representation of the study area Berneburg I

```

-----Vertices-----
v a = vertex a :: Vertex

-----Edges-----
el = [(v 1,v 18), (v 2,v 3), (v 3,v 4), (v 4,v 5), (v 5,v 6), (v 6,v 17),
      (v 7,v 8), (v 9,v 16), (v 9,v 10), (v 10,v 1), (v 13,v 11),
      (v 13,v 9), (v 11,v 12), (v 12,v 2), (v 10,v 12), (v 16,v 9),
      (v 15,v 7), (v 15,v 14), (v 14,v 16), (v 13,v 14), (v 6,v 11),
      (v 18,v 2), (v 17,v 15)]
lodl = [1,1,1,1,1,1,1,1,1,1,2,2,3,3,3,1,1,3,3,3,2,1,1]
el' = zip (zip [1..] el) lodl
ed((a,(b,c)),d) = dEdge a b c d
eL :: [DEdge]
eL = map ed el'
e a = head (drop (a-1) eL) :: DEdge

-----DGraph-----
g411,g412,g413,g41,g42,g43,g44 :: DGraph
g311,g312,g313,g31,g32,g33,g34 :: DGraph
g21,g22,g11 :: DGraph

g411 = dGraph 411 [e 21,e 11,e 6,e 23,e 18,e 20]
      [v 11,v 17,v 6,v 13,v 14,v 15]           --D1a (area)
g412 = dGraph 412 [e 12,e 16,e 19,e 20]
      [v 13,v 14,v 16,v 9]                   --D1b (area)
g413 = dGraph 413 [e 18,e 19,e 8,e 7,e 17]
      [v 15,v 14,v 16,v 8,v 7]               --D1c (area)

g41 = dGraph 41 [e 21,e 11,e 12,e 16,e 8,e 7,e 17,e 23,e 6]
      [v 6,v 13,v 11,v 9,v 16,v 8,v 7,v 15,v 17] --D1
g42 = dGraph 42 [e 1,e 22,e 14,e 15,e 10]
      [v 1,v 2,v 12,v 10,v 18]               --D2
g43 = dGraph 43 [e 15,e 13,e 12,e 11,e 9]
      [v 10,v 12,v 11,v 13,v 9]              --D3
g44 = dGraph 44 [e 2,e 3,e 4,e 5,e 21,e 13,e 14]
      [v 2,v 3,v 4,v 5,v 6,v 11,v 12]        --D4

g311 = dGraph 311 (elist g411) (vlist g411)  --C1a (area)
g312 = dGraph 312 (elist g412) (vlist g412)  --C1b (area)
g313 = dGraph 313 (elist g413) (vlist g413)  --C1c (area)

g31 = dGraph 31 (elist g41) (vlist g41)      --C1
g32 = dGraph 32 (elist g42) (vlist g42)      --C2
g33 = dGraph 33 (elist g43) (vlist g43)      --C3
g34 = dGraph 34 (elist g44) (vlist g44)      --C4

g21 = dGraph 21 (filter (hasLev 2) (nub ((elist g311)
      ++(elist g312)++(elist g313))))
      (nub(concat(map vertices (elist g21))))  --B1
g22 = dGraph 22 (filter (hasLev 2) (nub ((elist g32)
      ++(elist g33)++(elist g34))))
      (nub(concat(map vertices (elist g22))))  --B2

g11 = dGraph 11 (filter (hasLev 1) (nub ((elist g21)++(elist g22))))
      (nub(concat(map vertices (elist g11))))  --A

-----DGraphs for Objects-----
-- generalized assumption for this example: only point objects
-- object numbers start with 1000

go a = dGraph (1000+a) [] [v (1000+a)] :: DGraph

```



```
o1,o2,o3,o4,o5,o6,o7,o8,o9,o10,o11,o12,o13,o14 :: Object
o1 = createMob 28 'O' "o1" 4 (go 1)
o2 = createMob 29 'O' "o2" 4 (go 2)
o3 = createMob 30 'O' "o3" 4 (go 3)
o4 = createMob 31 'O' "o4" 4 (go 4)
o5 = createMob 32 'O' "o5" 4 (go 5)
o6 = createMob 33 'O' "o6" 4 (go 6)
o7 = createMob 34 'O' "o7" 4 (go 7)
o8 = createMob 35 'O' "o8" 4 (go 8)
o9 = createMob 36 'O' "o9" 4 (go 9)
o10 = createMob 37 'O' "o10" 4 (go 10)
o11 = createMob 38 'O' "o11" 4 (go 11)
o12 = createMob 39 'O' "o12" 4 (go 12)
o13 = createMob 40 'O' "o13" 4 (go 13)
o14 = createMob 41 'O' "o14" 4 (go 14)

o15,o16,o17,o18,o19,o20,o21,o22,o23,o24 :: Object
o15 = createMob 42 'O' "o15" 3 (go 15)
o16 = createMob 43 'O' "o16" 3 (go 16)
o17 = createMob 44 'O' "o17" 3 (go 17)
o18 = createMob 45 'O' "o18" 3 (go 18)
o19 = createMob 46 'O' "o19" 3 (go 19)
o20 = createMob 47 'O' "o20" 3 (go 20)
o21 = createMob 48 'O' "o21" 3 (go 21)
o22 = createMob 49 'O' "o22" 3 (go 22)
o23 = createMob 50 'O' "o23" 3 (go 23)
o24 = createMob 51 'O' "o24" 3 (go 24)

o25,o26,o27,o28 :: Object
o25 = createMob 52 'O' "o25" 2 (go 25)
o26 = createMob 53 'O' "o26" 2 (go 26)
o27 = createMob 54 'O' "o27" 2 (go 27)
o28 = createMob 55 'O' "o28" 2 (go 28)
```

```
s1,s2,s3,s4,s5,s6,s7,s8 :: Street
s1 = createLW "Korngasse" [e 8,e 16,e 9,e 10]
s2 = createLW "Breite Str." [e 1,e 22,e 2]
s3 = createLW "Str.a" [e 3,e 4]
s4 = createLW "Str.b" [e 6,e 23,e 17]
s5 = createLW "Hellriegelstr." [e 7]
s6 = createLW "Am Kloster" [e 5,e 21,e 11,e 12]
s7 = createLW "Str.c" [e 13,e 14]
s8 = createLW "Str.d" [e 15]

tn1 :: Trans
tn1 = createTrans 1 [s1,s2,s3,s4,s5,s6,s7,s8] [] []

ta1,ta2,ta3,ta4 :: Tran
ta1 = createTran 1 "1:100 000" 1 tn1
ta2 = createTran 2 "1:50 000" 2 tn1
ta3 = createTran 3 "1:25 000" 3 tn1
ta4 = createTran 4 "1:10 000" 4 tn1

at1,at2,at3,at4,at5,at6,at7,at8,at9 :: Tree Area
at10,at11,at12,at13,at14,at15::Tree Area
at1 = T a1 [at2,at3]
at2 = T a2 [at4,at5,at6]
at3 = T a3 [at7,at8,at9]
at4 = T a4 [at10]
at5 = T a5 [at11]
at6 = T a6 [at12]
at7 = T a7 [at13]
at8 = T a8 [at14]
at9 = T a9 [at15]
at10 = T a10 []
at11 = T a11 []
at12 = T a12 []
at13 = T a13 []
at14 = T a14 []
at15 = T a15 []
```

```
ot1,ot2,ot3,ot4,ot5,ot6,ot7,ot8,ot9,ot10 :: Tree Object
ot11,ot12,ot13,ot14 :: Tree Object
ot1 = T o1 []
ot2 = T o2 []
ot3 = T o3 []
ot4 = T o4 []
ot5 = T o5 []
ot6 = T o6 []
ot7 = T o7 []
ot8 = T o8 []
ot9 = T o9 []
ot10 = T o10 []
ot11 = T o11 []
ot12 = T o12 []
ot13 = T o13 []
ot14 = T o14 []

ot15,ot16,ot17,ot18,ot19,ot20,ot21,ot22,ot23,ot24 :: Tree Object
ot15 = T o15 [ot1]
ot16 = T o16 [ot2]
ot17 = T o17 [ot3]
ot18 = T o18 [ot4]
ot19 = T o19 [ot5]
ot20 = T o20 [ot6]
ot21 = T o21 [ot7]
ot22 = T o22 [ot8]
ot23 = T o23 [ot9,ot10,ot11]
ot24 = T o24 [ot12]

ot25,ot26,ot27,ot28 :: Tree Object
ot25 = T o25 [ot15]
ot26 = T o26 [ot16,ot17]
ot27 = T o27 [ot18,ot19]
ot28 = T o28 [ot20,ot21,ot22,ot23,ot24]
```

```

ma1,ma2,ma3,ma4,ma5,ma6,ma7,ma8,ma9,ma10 :: MArea
ma11,ma12,ma13,ma14,ma15 :: MArea
mc1,mc2,mc3,mc4,mc5,mc6,mc7,mc8,mc9,mc10,mc11 :: MCon

ma1 = createTL a1 []
mc1 = createTL c1 [ma1]

ma2 = createTL a2 [o25,o26]
ma3 = createTL a3 [o27,o28]
mc2 = createTL c2 [ma2]
mc3 = createTL c3 [ma3]

ma4 = createTL a4 [o17]
ma5 = createTL a5 [o16]
ma6 = createTL a6 [o15]
ma7 = createTL a7 [o18]
ma8 = createTL a8 [o19]
ma9 = createTL a9 [o20,o21,o22,o23,o24]
mc4 = createTL c4 [ma4,ma5,ma6]
mc5 = createTL c5 [ma7]
mc6 = createTL c6 [ma8]
mc7 = createTL c7 [ma9]

ma10 = createTL a10 [o3]
ma11 = createTL a11 [o2]
ma12 = createTL a12 [o1]
ma13 = createTL a13 [o4]
ma14 = createTL a14 [o5]
ma15 = createTL a15 [o6,o7,o8,o9,o10,o11,o12,o13,o14]
mc8 = createTL c8 [ma10,ma11,ma12]
mc9 = createTL c9 [ma13]
mc10 = createTL c10 [ma14]
mc11 = createTL c11 [ma15]

tmc1,tmc2,tmc3,tmc4,tmc5,tmc6,tmc7,tmc8,tmc9,tmc10,tmc11 :: Tree MCon
tmc1 = T mc1 [tmc2,tmc3]
tmc2 = T mc2 [tmc4]
tmc3 = T mc3 [tmc5,tmc6,tmc7]
tmc4 = T mc4 [tmc8]
tmc5 = T mc5 [tmc9]
tmc6 = T mc6 [tmc10]
tmc7 = T mc7 [tmc11]
tmc8 = T mc8 []
tmc9 = T mc9 []
tmc10 = T mc10 []
tmc11 = T mc11 []

mdb :: MapDB
mdb = MDB tn1 tmc1 at1 [ot25,ot26,ot27,ot28]

```


References

- Ahl, Valerie and Allen, T.F.H. *Hierarchy Theory - A Vision, Vocabulary, and Epistemology*. New York: Columbia University Press, 1996.
- Alexander, Christopher, Ishikawa, Sara, and Silverstein, Murray. *A Pattern Language - Towns, Buildings, Construction*. New York: Oxford University Press, 1977.
- Asperti, Andrea and Longo, Giuseppe. *Categories, Types and Structures - An Introduction to Category Theory for the Working Computer Scientist*. 1 ed., Foundations of Computing, ed. Albert, Garey Michael and Meyer. Cambridge, Mass.: The MIT Press, 1991.
- Ballard, Dana H. "Strip Trees: A Hierarchical Representation for Curves." *ACM Comm.*, 24 (5 (May) 1981), pp. 310-321.
- Bannert, Birgit. "Herstellung fachthematischer Karten auf der Grundlage der TK 25 (R)." *Nachrichten aus dem Karten- und Vermessungswesen*, 115 (1996), pp. 9-18.
- Barrault, Mathieu. "An Automated System for Linear Feature Name Placement which complies with cartographic quality criteria." In *Autocarto 12* in Charlotte, NC, ACSM/ASPRS, pp. 321-330, 1995.
- Bertin, J. *Sémiologie Graphique*. Gauthier-Villars: 1967.
- Birkhoff, G. "Universal Algebra." In *First Canadian Math. Congress Toronto* University Press, pp. 310-326, 1945.
- Birkhoff, G. and Lipson, J. D. "Heterogeneous Algebras." *Journal of Combinatorial Theory*, 8 (1970), pp. 115-133.
- Bjørke, Jan T. and Aasgaard, Rune. "Cartographic Zoom." In *Fourth International Symposium on Spatial Data Handling* in Zurich, Switzerland; July 1990, edited by Kishimoto, Kurt Brassel and H., IGU, pp. 345-353, check 2:859-868, 1990.
- Bochenski, I.M. *Die zeitgenoessischen Denkmethode*. 10th ed., UTB, Tuebingen: A. Francke Verlag, 1954.
- Borgida, A, Mylopoulos, J., and Wong, H.K.T. "Generalization/Specialization as a Basis for Software Specification." In *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, edited by M. M. L. Brodie Schmidt, M. L. Brodie, Mylopolous and Schmidt. New York: Springer Verlag, pp. 87-117, 1984.
- Bowen, J.P., Breuer, P.T., and Lano, K.C. "Formal specifications in software maintenance: from code to Z++ and back again." *Information and Software Technology*, 35 (11/12 1993), pp. 679-690.
- Braitenberg, Valentino. *Vehicles, experiments in synthetic psychology*. Cambridge, MA: MIT Press, 1984.

- Brassel, K.E. and Weibel, R. "A review and conceptual framework of automated map generalization." *IJGIS*, 2 (3 1988), pp. 229-244.
- Brodie, Michael L., Mylopoulos, John, and Schmidt, Joachim W. *On Conceptual Modelling: perspectives from artificial intelligence, databases, and programming languages*. Springer-Verlag, 1984.
- Bruegger, B. P. and Frank, A. U. "Hierarchies over Topological Data Structures." In *ASPRS-ACSM Annual Convention* in Baltimore, MD, , pp. 137-145, 1989.
- Bundy, G.L.L., Jones, C.B., and Furse, E. "A topological structure for the generalization of large scale cartographic data." In *GIS/UK* in Leicester, England, edited by Fisher, P., , pp. 87-96, 1994.
- Bundy, Geraint Ll.; Jones, Christopher B, and Furse, Edmund. "A topological structure for the generalization of large-scale cartographic data." In *GIS Research UK (GIS/UK)* in Leicester, England, 1994.
- Bunemann, Peter and Nikhil, Rishiyur. "The functional data model and its uses for interaction with databases." In *On Conceptual Modelling: perspectives from artificial intelligence, databases, and programming languages*, edited by Brodie, Michael L., Mylopoulos, John, and Schmidt, Joachim W. Springer-Verlag, pp. 359-380, 1984.
- Buttenfield, Barbara and McMaster, Robert, eds. *Map generalization - Making rules for knowledge representation*. London: Longman, 1991.
- Buttenfield, Barbara P. and Delotto, Joseph S. *Multiple Representations: Report on the Specialist Meeting - Initiative 3*. NCGIA, Santa Barbara, CA, 1989. Report 89-3.
- Buttenfield, Barbara P. *et al.* . "How does a cartographic object behave? Computer inventory of topographic maps." In *GIS/LIS* in Atlanta, , pp. 891-900, 1991.
- Campari, I. and Frank, A. U. "Cultural Differences and Cultural Aspects in GIS." In *Cognitive Aspects of Human-Computer Interaction for Geographic Information Systems - Proceedings of the NATO Advanced Research Workshop*, edited by Nyerges, T. L. *et al.* 83. Dordrecht, The Netherlands: Kluwer Academic Publishers, pp. 249-266, 1995.
- Car, Adrijana. *Hierarchical Spatial Reasoning: Theoretical Consideration and its Application to Modeling Wayfinding*. Ph.D. Thesis. Vol. 10. GeoInfo Series, ed. Frank, Andrew U. and Haunold, Peter. Vienna: Department of Geoinformation, Technical University Vienna, 1996.
- Cardelli, Luca and Wegner, Peter. "On Understanding Types, Data Abstraction, and Polymorphism." *ACM Computing Surveys*, 17 (4 1985), pp. 471 - 522.
- Casati, Roberto and Varzi, Achille C. *Holes and Other Superficialities*. Cambridge, Mass.: MIT Press, 1994.
- Centennia. Chicago: Clockwork Software Inc., 1992-1995.

- Chalmers, A. F. *What is the thing called Science?* Second Edition ed., Indianapolis, USA: Hackett Publishing Company, 1994.
- Devogele, Thomas, Parent, Christine, and Spaccapietra, Stefano. "On spatial database integration." *IJGIS*, (special issue 1997),
- Devogele, Thomas and Raynal, Laurent. "Liens de correspondance entre des Bases de Donnees a differentes echelles." In *CASSINI'95* in Marseille, CNRS, pp. 1-12, 1995.
- Devogele, Thomas, Trevisan, Jenny, and Raynal, Laurent. "Designing and producing a multi-scale database from mono-scale databases." 1996.
- Dutton, Geoffrey. "Digital Map Generalization Using a Hierarchical Coordinate System." In *Autocarto 13* in Seattle, WA, USA, ACSM/ASPRS, pp. 367-376, 1997.
- Egenhofer, M.J., Clementini, E., and DiFelice, P. "Evaluating inconsistencies among multiple representations." In *SDH'94* in Edinburgh, U.K., edited by Waugh, T.C. and Healey, R.G., IGU, pp. 901-920, 1994.
- Egenhofer, Max J. and Franzosa, R. "Point-Set Topological Spatial Relations." *IJGIS*, 5 (1991),
- Ehrich, H.-D., Gogolla, M., and Lipeck, U.W. *Algebraische Spezifikation abstrakter Datentypen*. Leitfäden und Monographien der Informatik, ed. Appelrath, H.-J. et al. Stuttgart: B.G. Teubner, 1989.
- Erickson, Carl. *Polygonal Simplification: An Overview*. Department of Computer Science, UNC Chapel Hill, 1996. TR96-016.
- Erwig, Martin and Schneider, Markus. "Partition and Conquer." In *COSIT'97* in Laurel Highlans, PA, edited by Hirtle, Stephen C. and Frank, Andrew U., Springer Verlag, pp. 389-407, 1997.
- Fisher, Peter. "Propagating effects of databse generalization on the viewshed." *Transactions in GIS*, 1 (2 1996), pp. 69-81.
- Floriani, Leila De and Puppo, Enrico. "A Hierarchical Triangle-Based Model for Terrain Description." In *Theories and Models of Spatio-Temporal Reasoning in Geographic Space*, edited by Frank, A. U., Campari, I., and Formentini, U. 639. Pisa: Springer-Verlag, pp. 236-251, 1992.
- Fotheringham, A.S. "Encoding Spatial Information: The Evidence for Hierarchical Processing." In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, edited by Frank, A.U., Campari, I., and Formentini, U. 639. Heidelberg-Berlin: Springer Verlag, pp. 269-287, 1992.
- Frank, A.U. "MAPQUERY: Database Query Language for Retrieval of Geometric Data and its Graphical Representation." *ACM SIGGRAPH*, 16 (3 1982), pp. 199 - 207.

- Frank, A. "Datenstrukturen für Landinformationssysteme - Semantische, Topologische und Räumliche Beziehungen in Daten der Geo-Wissenschaften." Dissertation, ETH Zürich, Institut für Geodäsie und Photogrammetrie, 1983.
- Frank, A. U. "Spatial Concepts, Geometric Data Models and Data Structures." In *GIS Design Models and Functionality* in Leicester, UK, edited by Maguire, David, Midlands Regional Research Laboratory, University of Leicester 1990.
- Frank, A. U., Campari, I., and Formentini, U., eds. *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*. Vol. 639. Lecture Notes in Computer Science. Berlin: Springer Verlag, 1992.
- Frank, A. U. "Beyond Query Languages for Geographic Databases: Data Cubes and Maps." In *International Workshop on DBMS for Geographic Applications*, edited by Ausiello, A. Heidelberg: Springer, pp. 74 - 86, 1992.
- Frank, A. U. *Using Hierarchical Spatial Data Structures for Hierarchical Spatial Reasoning. Internal Report*. Dept. of Geoinformation, Technical University Vienna, 1996. Internal Report
- Frank, A. U., Volta, G. S., and McGranaghan, M. "Formalization of Families of Categorical Coverages." *IJGIS*, 11 (3 1997), pp. 215-231, 1997a.
- Frank, A. U. "Higher Order Functions Necessary for Spatial Theory Development." In *Auto-Carto 13* in Seattle, WA SH. (7 - 10 April 1997), edited by Chrisman, Nick, ACSM/ASPRS, pp. 11-22, 1997b.
- Frank, A. U. "How to write specifications in class based functional programming." In *Gofor Manual*, edited by Frank, Andrew U. 12. Vienna: *Dept. for Geo-Information E127 Technische Universität Wien* 1997c.
- Frank, A. U. and Timpf, Sabine. "Multiple Representations for cartographic objects in a multi-scale tree - an intelligent graphical zoom." *Computers and Graphics Special Issue on Modeling and Visualization of Spatial Data in GIS*, 18 (6 1994), pp. 823-829.
- Freeman, H. "Computer name placement." In *Geographical Information Systems: principles and applications*, edited by Maguire, David J., Goodchild, Michael F., and Rhind, David W. 1. Essex: Longman Scientific & Technical, pp. 445-456, 1991.
- Freeman, H. and Ahn, J. "On the Problem of Placing Names in a Geographic Map." *International Journal of Pattern Recognition and Artificial Intelligence*, 1 (1 1987), pp. 121-140.
- Freksa, C. "Qualitative Spatial Reasoning." In *Cognitive and Linguistic Aspects of Geographic Space*, edited by Mark, D. M. and Frank, A. U. Dordrecht, The Netherlands: Kluwer Academic Press, pp. 361-372, 1991.
- Gibson. *The ecological approach to visual perception*. Houghton Mifflin, 1979.

- Gill, A. *Applied Algebra for the Computer Sciences*. Englewood Cliffs, NJ: Prentice-Hall, 1976.
- Glasgow, J. "A Formalism for Model-Based Spatial Planning." In *Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'95)*, edited by Frank, A.U. and Kuhn, W. 988. Berlin-Heidelberg: Springer-Verlag, pp. 501-518, 1995.
- Gold, Christopher M. "Dynamic Spatial Data Structures - The Voronoi Approach." In *The Canadian Conference on GIS Proceedings in Ottawa*, , pp. 245-255, 1992.
- Gueting, R.H. *GraphDB: A Data Model and Query Language for Graphs in Databases*. FernUniversität Hagen, 1994. Informatik-Bericht 155.
- Guptill, S.C. "Speculations on seamless, scaleless, cartographic data bases." In *Auto-Carto 9* in Baltimore, MA, edited by Anderson, E., ASPRS & ACSM, pp. 436-443, 1989.
- Guttag, J.V. and Horning, J.J. "The Algebraic Specification of Abstract Data Types." *Acta Informatica*, 10 (1 1978), pp. 27-52.
- Guttag, J.V., Horowitz, E., and Musser, D.R. "The Design of Data Type Specifications." In *Current Trends in Programming Methodology*, edited by Yeh, R.T. vol. 4: Data Structuring. Prentice Hall, pp. 60-79, 1978.
- Guttman, Antonin. "R-Trees: A Dynamic Indexing Structure for Spatial Searching." (1984),
- Haack, E. "Dokumentation ueber die Herstellung und Fortfuehrung der amtlichen topographischen Kartenwerke der ehemaligen DDR (1945-1990)." *Nachrichten aus dem Karten- und Vermessungswesen*, Reihe I (116 1996), pp. 5-59.
- Hake, G. *Kartographie*. Sammlung Göschen, de Gruyter, 1975.
- Hall, P. A. V., ed. *Software reuse and reverse engineering in practice*. London: Chapman and Hall, 1992.
- Head, C. Grant. "Mapping as language or semiotic system: review and comment." In *Cognitive and Linguistic aspects of space*, edited by Mark, D.M. and Frank, A.U. Dordrecht: Kluwer Academic Publishers, pp. 237-262, 1991.
- Herot, Christopher F., Carling, Richard, Friedell, Mark, and Kramlich, David. "A Prototype Spatial Data Management System." In *ACM SIGGRAPH'80* , pp. 63-70, 1980.
- Hirtle, S.C. "Representational Structures for Cognitive Space: Trees, Ordered Trees and Semi-Lattices." In *Spatial Information Theory-A Theoretical Basis for GIS*, edited by Frank, A.U. and Kuhn, W. 988. Berlin-Heidelberg-New York: Springer, pp. 327-340, 1995.
- Hirtle, S. C. and Jonides, J. "Evidence of Hierarchies in Cognitive Maps." *Memory & Cognition*, 13 (3 1985), pp. 208-217.

- Hoschek, Josef and Dankwort, Werner, eds. *Reverse engineering*. Stuttgart: Teubner, 1996.
- Hudak, P *et al.* . "Report on the functional programming language Haskell, Version 1.2." *SIGPLAN Notices*, 27 (1992),
- Hudson, John C. "Scale in Space and Time." In *Geography's Inner World: Pervasive Themes in Contemporary American Geography*, edited by Abler, Ronald, Marcus, Melvin, and Olson, Judy. Rutgers University Press, pp. 280-300, 1992.
- Jones, Mark P. *An Introduction to Gofer*. Yale University, 1991. (available by anonymous ftp)
- Jones, Mark P. "Functional Programming with Overloading and Higher-Order Polymorphism." In *Advanced Functional Programming*, edited by Jeuring, Johan and Mejer, Erik. 925. Berlin: Springer, pp. 331, 1995.
- Kakas, A.C., Kowalski, R.A., and Toni, F. "Abductive Logic Programming." *Journal of Logic and Computation*, 2 (6 1993), pp. 719-770.
- Kidner, David B. and Jones, Christopher B. "A deductive object-oriented GIS for handling multiple representations." In *Six International Symposium on Spatial Data Handling* in Edinburgh, edited by Waugh, T. C. and Healey, R. G., AGI, pp. 882 - 900, 1994.
- Kilpelainen, Tiina. "Updating multiple representation geodata bases by incremental generalization." In *Spatial Information from digital photogrammetry and computer vision* in Munich, Germany, ISPRS, pp. 440-447, 1994.
- Koestler, Arthur. *The Ghost in the Machine*. London: Pan, 1967.
- Kottik, Peter. "Morphen in GIS." Master thesis, Technical University Vienna, 1997.
- Kuhn, Werner. "Defining Semantics for Spatial Data Transfers." In *6th International Symposium on Spatial Data Handling* in Edinburgh, UK, edited by Waugh, Thomas C. and Healey, Richard G., IGU, pp. 973-987, 1994.
- Kuhn, Werner. *Semantics of Geographic Information*. Vol. 7. Geoinfo Series, ed. Frank, Andrew U. Vienna, Austria: Dept. of Geoinformation, TU Vienna, 1995.
- Kuhn, Werner and Blumenthal, Brad. *Spatialization: Spatial Metaphors for User Interfaces*. Vol. 10. CHI 96 Tutorials, Vancouver, BC: ACM, 1996.
- Kuipers, B. et al. "The Semantic Hierarchy in Robot Learning." In *Robot Learning*, edited by Connell, J. and S. Mahadevan. Kluwer Academic 1993.
- Lagrange, J.P. and Ruas, A. "Data & Knowledge Modelling for Generalisation." In *6th International Symposium on Spatial Data Handling* in Edinburgh, Scotland, edited by T. Waugh and Healey, R.G., IGU 1994.

- Lakoff, G. "Cognitive Models and Prototype Theory." In *Concepts and Conceptual Development: Ecological and Intellectual Factors in Categorization*, edited by Neisser, Ulric. New York: Cambridge University Press 1987a.
- Lakoff, George. *Women, Fire, and Dangerous Things: What Categories Reveal About the Mind*. Chicago, IL: University of Chicago Press, 1987b.
- Langacker, Ronald W. *Foundations of Cognitive Grammar*. Vol. 1 Theoretical Prerequisites. Stanford, Cal.: Stanford University Press, 1987.
- Lano, Kevin and Haughton, Howard. *Reverse engineering and software maintenance : a practical approach*. London: McGraw-Hill, 1994.
- Laurini, Robert and Thompson, Derek. *Fundamentals of Spatial Information Systems*. APIC Series, Academic Press, 1991.
- Lee, Dan. *Cartographic Generalization*. Intergraph Corporation, 1992.
- Leitner, Michael. "Prototype rules for automated map generalization." Master thesis, State University of New York at Buffalo, 1993.
- Leitner, Michael and Battenfield, Barbara P. "Acquisition of Procedural Cartographic Knowledge by Reverse Engineering." *Cartography and Geographic Information Systems*, 22 (3 1995), pp. 232-241.
- Liebermann, Henry. "Powers of ten thousand: navigating in large information spaces." In *UIST'94* in Marina del Rey, ACM, pp. 15-16, 1994.
- Lindholm, Mikko and Sarjakoski, Tapani. "User Models and Information Theory in the Design of a Query Interface for GIS." In *Theories and Models of Spatio-Temporal Reasoning in Geographic Space*, edited by Frank, A. U., Campari, I., and Formentini, U. 639. Pisa: Springer-Verlag, pp. 328-347, 1992.
- Lindholm, M. and Sarjakoski, T. "Designing a visualization user interface." In *Visualization in modern cartography*, edited by MacEachren, A. and Taylor, D.R.F. Oxford: Pergamon, Elsevier Science Ltd., pp. 167-184, 1994.
- Liskov, Barbara and Guttag, John. *Abstraction and Specification in Program Development*. Cambridge, MA: MIT Press, 1986.
- Lokuge, Ishantha and Ishizaki, Suguru. "GeoSpace: An interactive visualization system for exploring complex information spaces." In *CHI'95* edited by ACM, ACM, pp. 409-414, 1995.
- Mackinlay, Jock. "Automating the Design of graphical presentations of relational information." *ACM Transactions on Graphics*, 5 (2, April 1986), pp. 110-141.
- Mainguenaud, M. and Simatic, X.T. "A Data Model to Deal with Multi-Scaled Networks." *Computers, Environment and Urban Systems*, 16 (1992), pp. 281-288.
- Marciniak, J.J., ed. *Encyclopedia of software engineering*. Vol. 2. New York: John Wiley & Sons, 1994.

- Mark, D.M. "Toward a Theoretical Framework for Geographic Entity Types." In *Spatial Information Theory: Theoretical Basis for GIS*, edited by Frank, A.U. and Campari, I. 716. Heidelberg-Berlin: Springer Verlag, pp. 270-283, 1993.
- Mark, David M. and Egenhofer, Max J. "An Evaluation of the 9-Intersection for Region-Line Relations." In *GIS/LIS '92 Proceedings in San Jose, ACSM-ASPRS-URISA-AM/FM*, pp. 513-521, 1992.
- McMaster, Robert B. and Shea, K. Stuart. *Generalization in Digital Cartography*. AAG, 1992.
- Mesarovic, M.D., Macko, D., and Takahara, Y. *Theory of hierarchical, multilevel, systems*. Mathematics in science and engineering, ed. Bellmann, Richard. New York: Academic Press, 1970.
- Minsky, M. *The Society of Mind*. New York: Simon & Schuster, 1985.
- Misund, Gunnar. "Varioscale TIN based surfaces." In *7th International Symposium on Spatial Data Handling* in Delft (NL), IGU, pp. 6.36-6.45, 1996.
- Molenaar, M. "Object hierarchies or uncertainty in GIS or why is standardisation so difficult." *GeoInformations-Systeme*, 6 (3 1993), pp. 22-28.
- Mueller, Jean-Claude, Lagrange, Jean-Philippe, and Weibel, Robert, eds. *GIS and Generalization - Methodology and Practice*. GISDATA. London: Taylor&Francis, 1995.
- Mukherjea, Sougata, Foley, James D., and Hudson, Scott. "Visualizing complex hypermedia networks through multiple hierarchical views." In *CHI'95* edited by ACM, ACM, pp. 331-337, 1995.
- Oosterom, P. van. "The Reactive Tree - A Storage Structure of a Seamless, Scaleless Geographic Database." In *Tenth International Symposium on Computer-Assisted Cartography* in Baltimore, MD, edited by Mark, D.M. and White, D., ACSM/ASPRS, pp. 393-407, 1991.
- Oosterom, Peter van. "Maintaining consistent topology including historical data in a large spatial database." In *Autocarto 13* in Seattle, WA, USA, ACSM/ASPRS, pp. 327-336, 1997.
- Oosterom, Peter van and Schenkelaars, Vincent. "The Development of an interactive multiscale GIS." *International Journal of Geographical Information Systems*, 9 (5 1995), pp. 489-507.
- Palmer, Bruce and Frank, Andrew. "Spatial Languages." In *Third International Symposium on Spatial Data Handling* in Sydney, Australia, edited by Marble, D., International Geographical Union, Commission on Geographical Data Sensing and Processing, pp. 201-210, 1988.
- Parsons, Jeffrey and Wand, Yair. "Choosing Classes in Conceptual Modeling." *Communications of the ACM*, 40 (6 1997), pp. 63-69.

- Pattee, Howard H., ed. *Hierarchy Theory - The Challenge of Complex Systems*. New York: Braziller, 1973.
- Perl, J. *Graph Theory (in german)*. Wiesbaden (FRG): Akademische Verlagsgesellschaft, 1981.
- Piff, Mike. *Discrete Mathematics - An Introduction for Software Engineers*. Cambridge: Cambridge University Press, 1991.
- Plazanet, Corinne. "Geometry modeling for linear feature generalization." In *Joint ESF/NSF Summer Institute 1995*, edited by Craglia, Max. Taylor & Francis 1995.
- Powitz, Bernd Michael. "Zur Automatisierung der kartographischen Generalisierung topographischer Daten in Geo-Informationssystemen." Dissertation, Hannover, 1993.
- Rekoff, M.G. "On Reverse Engineering." *IEEE Transactions on systems, man, and cybernetics*, 15 (2 1985), pp. 244-252.
- Rieger, M. and Coulson, M. "Consensus of Confusion: Cartographers' Knowledge of Generalization." *Cartographica*, 30 (1 1993), pp. 69-80.
- Rigaux, P. and Scholl, M. "Multi-Scale Partitions: Application to Spatial and Statistical Databases." In *Fourth International Symposium on Large Spatial Databases-SSD95* in Portland, ME, edited by Egenhofer, M. J. and Herring, J., Springer Verlag, Heidelberg, pp. 170-183, 1995.
- Robertson, Geroge G. and Mackinlay, Jock D. "The document lens." In *UIST'93* in Atlanta, ACM, pp. 101-108, 1993.
- Robinson, A. and Petchenik, B. *The Nature of Maps*. University of Chicago Press, 1976.
- Rosch, E. "On the internal structure of perceptual and semantic categories." In *Cognitive Development and the Acquisition of Language*, edited by Moore, T. E. New York: Academic Press 1973.
- Rosch, E. "Principles of categorization ." In *Cognition and Categorization*, edited by Rosch, E. and Lloyd, B. B. Hillsdale, NJ: Erlbaum 1978b.
- Rosch, E. "Principles of Categorization." In *Cognition and Categorization*, edited by Rosch, E. and Lloyd, B. B. Hillsdale, NJ: Erlbaum 1978a.
- Rosenfeld, A., Samet, H., Shaffer, C., and Weber, R.E. *Applications of hierachical data structures to geographical information systems*. Computer Vision Laboratory, Computer Science Center, University of Maryland, 1982.
- Ruas, Anne and Plazanet, Corinne. "Strategies for automated generalization." In *7th internation symposium on spatial data handling* in Delft (NL), IGU, pp. 6.1-6.18, 1996.
- Samet, H. *The Design and Analysis of Spatial Data Structures*. Reading, MA: Addison-Wesley Publishing Company, 1989.

- Schlichtmann, Hansgeorg. "Characteristic traits of the semiotic system 'map symbolism'." *Cartographic Journal*, (1985), pp. 23-30.
- SGK. *Cartographic Generalization*. 2nd edition ed., Zuerich: SGK-Publikationen, 1987.
- Simon, Herbert A. *The Sciences of the Artificial*. Cambridge, MA: MIT Press, 1981.
- Staufenbiel, W. *Zur Automation der Generalisierung topographischer Karten mit besonderer Berücksichtigung grossmasstäbiger Gebäudedarstellungen*. Kartographisches Institut Hannover, 1973. Wissenschaftliche Arbeiten (WissArbUH) WissArbUH 51.
- Stevens, A. and Coupe, P. "Distortions in judged spatial relations." *Cogn. Psychology*, 10 (1978), pp. 422 - 437.
- Stone, Maureen C., Fishkin, Ken, and Bier, Eric A. "The movable filter as a user interface tool." In *CHI'94* in Boston, edited by ACM, ACM Press, pp. 306-312, 1994.
- Tarski, Alfred. *Introduction to Logic and to the Methodology of Deductive Sciences*. Translated by Olaf Helmer. Mineola, N.Y.: Dover Publications, 1995.
- Thiemann, Peter. *Grundlagen der funktionalen Programmierung*. Leitfaden der Informatik, Stuttgart: B. G. Teubner, 1994.
- Timpf, S. "Cartographic objects in a multi-scale data structure." In *Geographic Information Research: Bridging the Atlantic*, edited by Craglia, Massimo and Couclelis, Helen. 1. London: Taylor&Francis, pp. 224-234, 1997.
- Timpf, S. and Devogele, T. "Multi-scale representations - why do we need them and what tools exist in GIS?" In *ICC'97* in Stockholm, 1997.
- Timpf, S. and Frank, A. U. "A Multi-Scale Dag for Cartographic Objects." In *ACSM/ASPRS* in Charlotte, NC, , pp. 157-163, 1995a.
- Timpf, S. and Frank, A. U. "A multi-scale data structure for cartographic objects." In *17th International Cartographic Conference ICC'95* in Barcelona, Spain, edited by ICA, ICA, pp. 1389-1396, 1995b.
- Timpf, S. and Frank, A. U. "Exploring the life of screen objects." In *Auto-Carto 13* in Seattle, WA, USA (April 7-10, 1997), ACSM/ASPRS, pp. 195-203, 1997a.
- Timpf, S and Frank, A. U. "Using hierarchical spatial data structures for hierarchical spatial reasoning." In *Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'97)*, edited by Hirtle, Stephen C. and Frank, Andrew U. Lecture Notes in Computer Science 1329. Berlin-Heidelberg: Springer-Verlag, pp. 69-83, 1997b.
- Timpf, S., Volta, G.S., Pollock, D.W., and Egenhofer, M.J. "A Conceptual Model of Wayfinding Using Multiple Levels of Abstractions." In *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, edited by Frank, A.U., Campari, I., and Formentini, U. 639. Heidelberg-Berlin: Springer Verlag, pp. 348-367, 1992.

- Tversky, B. "Cognitive Maps, Cognitive Collages, and Spatial Mental Model." In *Spatial Information Theory: Theoretical Basis for GIS*, edited by Frank, A.U. and Campari, I. 716. Heidelberg-Berlin: Springer Verlag, pp. 14-24, 1993.
- Volta, G.S. and Egenhofer, M.J. "Interaction with GIS Attribute Data Based on Categorical Coverage." In *Spatial Information Theory: Theoretical Basis for GIS*, edited by Frank, A.U. and Campari, I. 716. Heidelberg-Berlin: Springer Verlag, pp. 215-247, 1993.
- Walters, R.F.C. *Categories and computer science*. Vol. 1. Cambridge Computer Science Texts, Cambridge, UK: Carslaw Publications, 1991.
- Weibel, R. "Amplified intelligence and rule-based systems." In *Map generalization: Making rules for knowledge representation*, edited by Buttenfield, Barbara P. and McMaster, Robert B. Essex: Longman Scientific & Technical, pp. 172-186, 1991.
- Weibel, R. and Heller, M. "Digital terrain modelling." In *Geographical Information Systems: principles and applications*, edited by Maguire, David J., Goodchild, Michael F., and Rhind, David W. 1. Essex: Longman Scientific & Technical, pp. 269-297, 1991.
- Weibel, R., Keller, S., and Reichenbacher, T. "Overcoming the Knowledge Acquisition Bottleneck in Map Generalisation: The Role of Interactive Systems and Computational Intelligence." In *Spatial Information Theory-A Theoretical Basis for GIS*, edited by Frank, A.U. and Kuhn, W. 988. Berlin-Heidelberg-New York: Springer, pp. 139-156, 1995.
- Wilson, Robin J. and Watkins, John J. *Graphs - An introductory approach*. New York: John Wiley & Sons, 1990.
- Youngmann, Carl. "A linguistic approach to map description." In *First International Advanced Study Symposium on Topological Data Structures for Geographic Information Systems* in Cambridge, Mass., edited by Dutton, Geoffrey, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, pp. Youngman/1-17, 1977.

Biography of the Author

Sabine Timpf was born in Lustenau, Austria on September 26, 1967. She was moved to Koblenz, Germany shortly afterwards. She received the 'Brevet des Colleges' from Lycee Emilie de Rodat in Toulouse, France in June 1983 after a year as exchange student. She graduated from Hilda Gymnasium in Koblenz, Germany in June 1987. Ms. Timpf attended Hannover University where she earned a Diplom-Ingenieur (Vermessungswesen) in 1993. Starting August 1991 she visited the University of Maine, U.S.A as exchange student on a grant from the German academic exchange service (DAAD) and received a Master of Science (in Surveying Engineering) in August 1992. Since January 1994 Ms. Timpf works at the Department of Geoinformation at the Technical University of Vienna. She has received a fellowship for the participation in the Joint ESF/NSF Summer Institute (GISDATA Program) at Wolfe's Neck, Maine, U.S.A in July/August 1995 . She received a grant to visit the Institut Geographique National in Paris for two weeks in February 1996 from the Centre des Etudiants International, Paris. In April 1997 she was awarded first prize in the student paper competition of the ACSM/ASPRS annual conference (Autocarto 13). She has authored or co-authored ten research papers published in conference proceedings and two research papers published in scientific journals. She is a candidate for the 'Doktor der technischen Wissenschaften' degree from the Technical University Vienna in 1998.