A Problem-Oriented Approach to Description and Analysis of Geographic Requirements

by

Maria Augusta Vieira Nelson

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2003

Your file Votre référence
ISBN: 0-612-83012-8
Our file Notre référence
ISBN: 0-612-83012-8

Abstract

Software requirements describe a problem in the real world that a software system is intended to solve. Describing requirements is challenging because usually too much attention is given to the final software product instead of concentrating on the problem itself and the real world. The area of geographic applications is no exception.

A geographic application is a computer-based system that solves a geographic problem from the natural or man-made environment using a map. Geographic applications are usually complex and data-intensive. The real world has many details that may need to be captured. A map that acts as a model of the real world reflects this complexity and has many of its own intricacies, such as a projection, a datum, and a coordinate system. The problem that an application is intended to solve is often overlooked because of the difficulties in dealing with the data.

Existing approaches to software development that are specific to the geographic area, for example, GIS tools, spatial databases, geographic query languages, and spatial data structures, are suitable for designing and implementing geographic applications and are, therefore, solution-oriented. There appears to be no problem-oriented approach for requirements description of geographic applications. In addition, an informal requirements description relies on intuitive notions and ideas about geographic phenomena. This description is often imprecise. Formalizing the requirements description allows checking that the formal description has the expected characteristics of the informal notion of the geographic phenomena.

Most geographic applications are composed of well-known geographic subproblems. The proposed approach provides classes of common geographic subproblems that can be used to promote analysis and description of real-world problems. Each class of problems is presented as a problem frame. Problem frames are an existing technique in which each frame conveys the essence of a family of problems. To describe a complex problem, the analyst must recognize which parts of the problem can be framed into the given subproblems.

The proposed approach provides also a formalization of the geographic problem frames that can be used to describe the requirements of specific applications. The resulting formal descriptions can be verified against desired formal properties. The contributions of this dissertation include: an approach to informal problem description and analysis of geographic requirements, consisting of definitions of typical classes of common geographic problems and a detailed description of each geographic problem using problem frames; an approach to formal description and analysis of geographic requirements, consisting of formal descriptions of the geographic frames and of specialized requirements and specifications, as well as formal map domain descriptions; and a demonstration of informal and formal problem description and analysis using two case studies.

Acknowledgements

I would like to thank my supervisor, Don Cowan, for the years of friendship, support, patience, and most of all for allowing me to find my own path, guiding me with his wisdom. I have learned so much from Don.

Paulo Alencar played an important role during my years of work as a graduate student. He was always there to discuss any points I brought to his attention. I am grateful for all his guidance and for the many hours of dedication to my work.

I also thank Daniel Berry for his words of encouragement to finish this work ever since he came to the University of Waterloo. His technical comments were always appreciated as well. It was also a pleasure to have the opportunity to be around Dan at many conferences. I certainly know many researchers in my discipline now because of him.

Jo Atlee has always been my role model. She showed me that if I persisted I would succeed. In many converstions she shared her experience as a Ph.D. student, and that made me feel capable of completing this task. She had a major impact on my work because she introduced to me the two underlying techniques of my dissertation, problem frames and Alloy.

Colin Mayfield provided me with many examples throughout these years that made me reflect on what I was doing. I appreciate that he shared his expertise on geographic applications with Don and I, helping me to shape the scope of my work.

Michael Jackson contributed with so many valuable comments to my dissertation. I learned so much from him. I admire the clarity of his work. I couldn't have found a better external examiner. I thank him for coming to Waterloo from England to be present at my oral examination. We all enjoyed having him here for a few days. I was very honoured to have the opportunity to know him better. My admiration for him has grown considerably.

My friends from the Computer Systems Group always made the work environment more pleasant. In particular I would like to thank my fellow Ph.D. colleagues, Luis Nova, Kurt Lichtner, Daniel Morales-German, and Jing Dong. I miss our discussions and conversations about work and life.

Special thanks go to Maureen Stafford, my first contact person in Waterloo and a great friend. She is a sunshine and a very sweet person! Having her around all these years made me feel at home in Canada. I'm going to miss her! I also thank her for proofreading my thesis. Only a true friend could be so patient.

I have so many other friends to acknowledge and thank. In particular I would like to mention Aliz Csenki, Darlene Ryan from the International Student Office, and the friends from the Brazil-

ian community in Waterloo, from the Teaching Resources and Continuing Education Office, and from the Education Program for Software Professionals.

I would like to acknowledge the Brazilian agency CAPES that financially supported the first four years of my work. I would also like to thank Don Cowan for providing the remaining financial support for me to conclude the work.

My in-laws always believed that I was capable of doing this work and that increased my responsibility, but it also provided me with more support from loved ones.

My parents are my angels in life. They bear the most responsibility for who I have become. Their daily encouragement increased my motivation to finish and to be able to return to Brazil and be a part of their daily lives again. They have my deepest gratitude for everything they have taught me, for their love and sacrifice to raise us all, and for their prayers.

My brothers and sisters inspired me, giving me reasons to keep going, each in their own way: my brother João with his perseverance; my sister Fernanda with her dedication to her work; and my sister Ana with her courage to do what she wants. I would also like to thank all my aunts, uncles and cousins who were always so supportive, especially my sweet niece Gabriela.

My husband, Torsten, shared the complete experience of being a Ph.D. student with me. I couldn't have done this without him. I thank him for being such a nice person. We grew together in the field of Software Engineering because of the numerous discussions we have every day. It is great to share my passion for my profession with the person I love. As a friend he was very supportive, especially in the moments of desperation when I wanted to give up and go home. In all these years we achieved so many things together, and now we have so much to celebrate and for which to be thankful. I feel blessed every day for being able to share my life with him. It is so rewarding.

Finally, I would like to thank a little person who is still inside my womb moving around and kicking. The baby was responsible for the last push to finish this work and I will always be grateful!

To my parents, Antônio & Natália

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A geographic application is a computer-based system that uses a map of a region of the world to solve a problem. Developers of these applications often concentrate on implementation and design issues, such as deciding which implementation platform to use, designing the database, and capturing the data. The implementation platform is usually an issue since most applications are developed using a Geographic Information System (GIS). A GIS is a computer system that can assemble, store, manipulate and display geo-referenced information, that is, data that can be identified through their locations [Mag91]. There are many GIS systems available in the market that offer different capabilities, levels of customization, and tradeoffs. Geographic data are also a reason for concern. Modeling a database to accommodate all types of relationships among geographic phenomena is often a complex task. The data capture process can be expensive also, depending on both the precision and the amount of data required.

The focus on implementation and design issues have not been sufficient to produce geographic applications that satisfy clients and users. Requirements descriptions focused on design or implementation issues may lead to committing to a solution too early in the software development lifecycle. Such requirements may lead to applications that solve the wrong problems, because the real problems may have never been analyzed or understood in the first place [ZJ97]. The problem to be solved by a particular application is often overlooked [GW91]. Requirements descriptions should focus on the problems and decisions about which solution to adopt should be postponed until a later stage when the problems are better understood [Jac95c, Ber01].

In the geographic area, a real-world problem may involve a combination of the following sub-

problems: resource location and inventory, measurement, classification of phenomena, proximity, topology, site selection, and spatial definition. A general-purpose, problem-analysis technique such as top-down analysis or object-oriented analysis does not make use of specific geographic subproblems, concepts, characteristics, and definitions, and therefore is not the most appropriate for geographic-problem analysis [SC96]. This dissertation explores an approach to requirements description based on informal problem analysis aimed at geographic applications.

However, informal problem analysis and description uses terms that are often not well defined, and also relies on informal notions, ideas, and concepts [EM95]. Formalizing this description usually contributes towards revealing flaws in problem understanding [vL00, Ber02]. Consequently, this dissertation explores also an approach to formal description and analysis of geographic problems.

1.2 Background

One of the activities in requirements engineering is the informal description and analysis of requirements. There are several approaches that address the construction of abstract descriptions and techniques for analysis. Examples of some of these are scenario-based approaches [Mai98], goal-oriented approaches [vLLD98] [Ant96], data modeling approaches such as entity-relationship and object-orientation, domain modeling approaches [PDA89] [JZ93], problem-oriented approaches [Jac00b], and natural language approaches [KBP01] [GN02] [Kov98].

There are other approaches that deal with description and analysis, but from a formal standpoint. Some are geared towards requirements engineering, such as Software Cost Reduction (SCR) [HJL96] and KAOS [vLLD98]. Others are used in requirements engineering but also in other areas of software engineering, such as model checking [ACG96] [EC02], theorem proving [ORS92], and model finding [Jac00a]. Each of these techniques encompasses a number of notations and tools for analysis. These techniques allow the formal description of the requirements of a software system and analysis of this description by verifying properties that are expected to hold in the formal description.

The work introduced in this dissertation focuses on a problem-oriented approach that allows both informal and formal description and analysis, specific to geographic requirements.

1.3 Problem statement

The geographic application area has specific concepts, definitions, and subproblems that must be captured in order to describe requirements [KPS96]. Existing general-purpose approaches to problem analysis and description are not appropriate to describe geographic subproblems and map representations due to their general nature [FCTJ01].

The two problems that will be addressed in this dissertation are:

- Requirements descriptions of computer-based systems, including geographic applications, should focus on problems. Existing approaches to software development of geographic applications such as GIS tools, spatial databases, geographic query languages, and spatial data structures are oriented towards design and implementation. There appears to be no approach oriented towards describing geographic problems.

- Formal analysis of problems in the geographic area requires abstraction and formalization of the concepts, subproblems, and definitions in the area. Existing approaches to formal description and analysis are not tailored to the geographic area, and thus, do not address this issue.

The issue of requirements description is a concern in any application area. It is commonly accepted that requirements should describe *what* the application should do and not *how* it is supposed to do it [Dav93]. To create a description containing *what* the application is supposed to do without committing to *how* or without implementation bias is not an easy task. In other words, a requirement should be a description of the *problem* to be solved by an application, rather than a description of a *product* (the application) that solves the original problem. In the area of geographic applications, the issue of problem description versus product description tends to be aggravated by the fact that some of the most important contributions in this area are Geographic Information Systems (GISs), which are implementation platforms.

Another issue regarding requirement descriptions of geographic applications is related to the fact that an informal notion of geographic space is present in daily life. When using geography routinely, one does not worry about being precise or unambiguous with terminology. *Naive geography* is a term coined to express these widespread informal geographic concepts [EM95]. There is a tendency to carry this vague geographic and spatial notion into the descriptions. However, requirements need to be precise and unambiguous.

1.3.1 The need for informal problem description and analysis

Many geographic applications are failures because of inadequate problem analysis. For example, a study performed by Hamil [Ham01] concluded that poor planning is the number one cause of failure in geographic applications. Among the six solutions to poor planning suggested by the author, three involve problem analysis: "understand the problem before jumping to a solution," "distinguish the problem from the symptoms surrounding it," and "define requirements clearly."

A study of a specific use of a geographic application that resulted in failure is reported by Openshaw *et al* [OCCB90]. This was a geographic application built for a police station in Northumbria, UK. The application was used for crime pattern analysis. Many of the reasons for failure that were listed in the study involve the lack of problem analysis, such as: "no good prior understanding of what the system would deliver if it worked," "inadequate understanding of police operations and needs," and "the level of data being collected was too great given the likely uses of the system."

Failures in watershed protection planning applications have been studied by Iles [Ile02]. The author states that one of the main reasons for failure is the focus on the tools and implementation platforms of geographic analysis rather than on their outcomes.

These studies show that without adequate problem analysis, geographic applications may fail because they do not solve the problems for which they were created, since there was no clear problem being addressed. Problem analysis and description is often neglected because the final product becomes the focus of the descriptions of the application requirements. The description usually talks about a specific GIS in which the application will be implemented. Focusing on the capabilities of a GIS may lead the software developers to miss important requirements of the problem, because their concern becomes the implementation. Unfortunately, this is very common in the area of geographic applications. During the symposiums on Environmental Software Systems of 1997 and 1998 [Swa01], for instance, the discussions did not include levels of abstraction higher than the design of environmental applications. There was a lack of discussion at the level of the requirements. The few times a requirement was mentioned, it was solution-oriented as in the following examples:

"The vegetation layer should be distinguished from the watershed layer. The meta information should be stored in the application database."

Although these requirements may seem appropriate, they are too implementation specific. A less implementation biased description would say:

"The vegetation and the watershed elements of the map should be distinguishable. The meta

information is necessary for ..."

Another cause of failures is the fact that the requirements of most geographic applications and information systems in general focus primarily on data analysis [NE00]. The data analysis leads to the creation of a data set. However, the created data set may not be relevant if the intended use of the data is not clear. It may happen that data are either collected and not used, or the necessary data are not collected. Collecting data for a geographic application means either creating a map by capturing the data from the world, or extracting and converting data from an existing data set. Both processes for collecting data are expensive [Fis91]. Data analysis is important, but should be performed in the light of problem analysis, since to solve different problems, different data collections may be needed, or data may need to be organized in a different way.

1.3.2 The need for formal description and analysis

"Naive Geography is the body of knowledge that people have about the surrounding geographic world" [EM95]. It captures how people think about the geographic space. Spatial relations are usually present in daily communication. For example, consider topological and metric relations. Topology as defined by McDonnell and Kemp [MK95] are "those properties of geometrical figures that are invariant under continuous deformation." For example, adjacency, containment, and overlap are topological spatial relations. Metric relations are those that involve size, shape, distance, or direction.

Some terms in human natural language refer to relations that are purely topological, such as *within* and *enters*, while others refer to metric relations, such as *north of* and *near*. Although these terms are acceptable in our daily life as referring to the way we think about the geographic space, they are not precise when referring to the real world or to the mathematical models of the geographic space. In the requirements for geographic applications, this terminology should be used only as a means to elicit information, not as a way to describe and document the requirements, since it is not precise in nature.

An example of how naive geography increases ambiguity in the requirements specification is related to the notion of distance. In Euclidean geometry, the distance from point A to point B is equal to the distance from B to A. In naive geography, this is frequently not the case, as distances may be seen as a measure of how long it takes to get from one place to another [Mar98]. In this case, the travel time between the two points may be different for reasons such as the need to take different one-way routes, or the route being uphill one way and downhill the other way, or varying traffic conditions.

Another example is related to the reference frame. The reference frame in geographic space is

the context in which spatial relations are conveyed. Many cultures use a reference frame based on cardinal directions. However, many others do not rely on this concept at all, preferring to indicate the direction of some well known location — an *object-centered* reference. Even in cultures that use cardinal directions, it is common to describe a location as, for example, "in front of the city hall", which is a directional reference frame relative to the city hall. If the main entrance to the city hall does not face the street where the city hall is located, does the phrase "in front of the city hall" refer to the street, or to the main entrance? The use of more than one reference frame during requirements description may cause ambiguity or multiple interpretations. This makes it difficult to analyze these descriptions in a rigorous manner.

1.3.3 Reconciling the needs

Figure 1.1 presents three "worlds" in which geographic applications exist [Sui98]. The *computer world* is mathematical in nature. It ranges from the axioms of Euclidean geometry to programming languages used to implement applications. The *naive geography world* is based on the human mind, including human senses and perceptions about the geographic environment. It is the set of informal ideas involving geographic relationships. The *real world* is the actual environment, including human and natural landscapes, which the field of geography attempts to study.

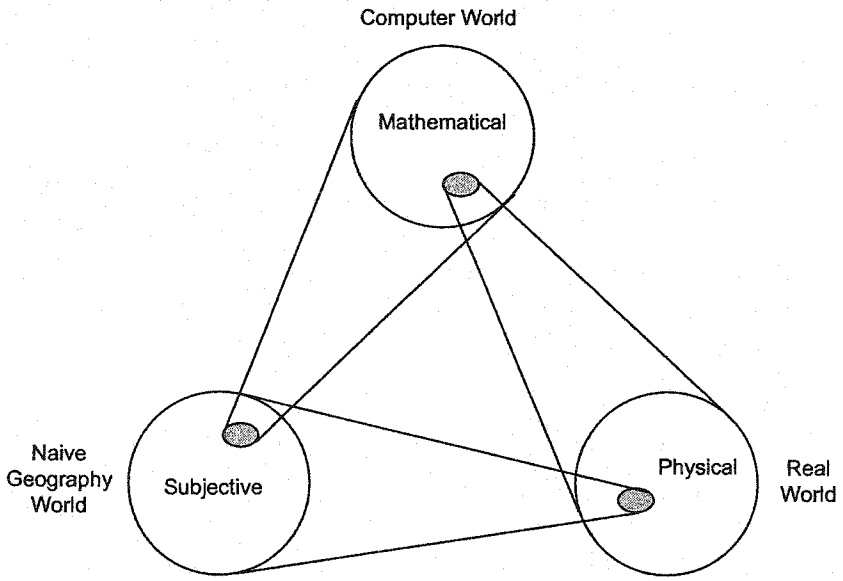


Figure 1.1: Different kinds of geographic worlds (modified from [Sui98])

The requirements description should concentrate on the actual problem as it manifests itself in the environment — the real world — that surrounds the computer and the people involved in the problem. Despite the focus on the real world, ultimately all three worlds must be accounted for in the requirements description and analysis. Informal description and analysis are carried out in the subjective world; formal description and analysis are carried out in the mathematical world; and both should refer to the problems that exist in the physical world.

1.4 Proposed solutions

The proposed approach focuses on problem analysis and description of geographic requirements. There are two main goals in the approach: to promote the understanding of a problem, and to achieve precise and consistent descriptions of the requirements of geographic applications. In order to achieve these goals, the approach provides a set of geographic problem frames and formal abstractions of these frames to allow formal problem analysis.

Part of this work involved the description of classes of problems that are most common in geographic applications. Each class stands for a number of similar problems, excluding details that are not relevant to the concern of the class, and concentrating on the difficulties of each problem. Most geographic problems that an analyst will face are captured in these classes. Chapter 3 presents and explains the classes of geographic problems.

Another part of this work is a detailed description of each class of problem, identifying its parts and their roles in the problem description. Each class of problem is described in the form of a generic template, so that the analyst can adapt it to the specifics of her problem. The problem templates are presented in this work using the problem frames technique [Jac00b]. A problem frame describes a simple problem using a diagram. Problem frames assist in analyzing and describing the elements involved in the problem, because the frames clearly indicate the problem's major parts. The advantage is that the analyst can determine the important parts that need to be described for each problem, and how they fit together in the frame. The geographic problem frames are detailed in Chapters 5 and 6.

Figure 1.2 gives an overview of informal problem analysis. The shaded areas represent the artifacts provided by the approach introduced in this dissertation. The unshaded areas represent the traditional problem frames approach. Essentially, the analyst breaks down an initial problem description into a set of subproblems based on the provided geographic classes. Each subproblem is described according to a problem frame. Because all geographic problems involve the use of a map, the maps for each subproblem must also be described. This work also provides different

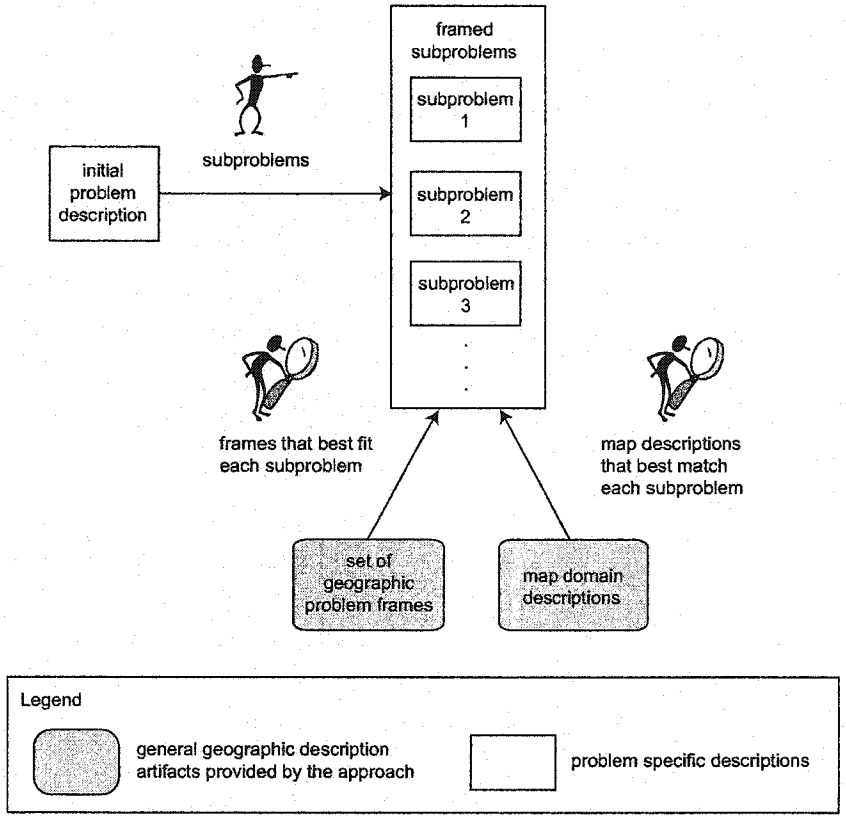


Figure 1.2: Informal problem analysis

kinds of map domain descriptions and their relation to the problem frames.

Once the description of each subproblem is done using the frames, the second stage is to verify the descriptions to detect problems that may have been overlooked or introduced in the informal analysis. This is done using formal methods for verification. This work provides a formal framework that presents formal descriptions of the geographic problem frames and formal representations of the map domain. The framework can be specialized by adding problem- and domain-specific details. Each subproblem can be described as an instance of a particular problem frame. The framework was implemented using the Alloy language [Jac]. Each subproblem description and the complete description can then be analyzed using the Alloy Constraint Analyzer tool. Properties that can be analyzed for each problem frame, such as range of measurements, feature dependencies, existence of paths, or availability of sites are also presented. It is also possible to check the consistency of descriptions, as well as verify that given the specifications and domain

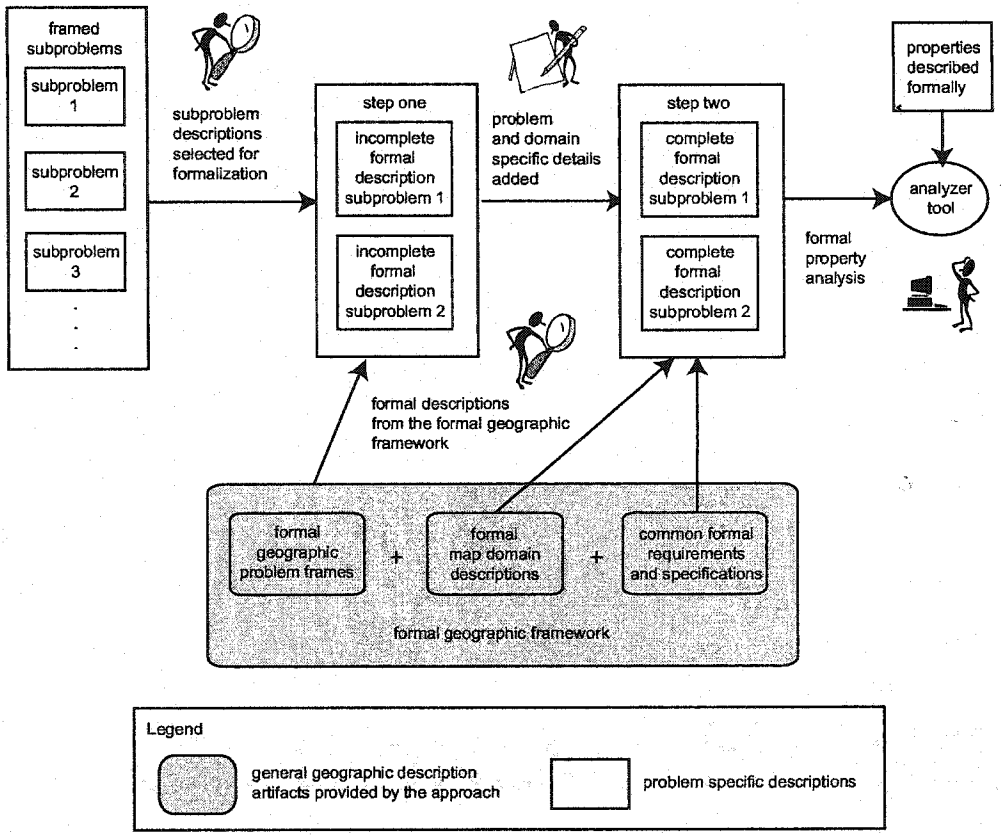


Figure 1.3: Formal problem analysis

descriptions, the requirements hold.

Figure 1.3 gives an overview of formal analysis of geographic applications. The shaded areas represent the artifacts provided by the approach introduced in this dissertation. The unshaded areas show traditional formal analysis techniques. Given the informal description of the framed subproblems, some subproblems are selected for formalization. Formal descriptions of the geographic frames are provided by the approach as part of the formal framework that also includes map domain descriptions, as well as specialized requirements and specifications. For example, if a subproblem is an instance of the *proximity* frame, the formal description of the proximity frame from the formal framework is the starting point for the subproblem description. Next, problem and domain specific information is added to the generic frames, in some cases by simply reusing map descriptions and specialized requirements specifications. For example, if the proximity subproblem requirement involves finding the *nearest* entity, the formalization of the *nearest* requirement

is reused from the framework. Because finding the nearest entity requires distance calculation, a map domain description with metric representation should be used. Finally, the resulting formal descriptions can be verified against specific properties using a formal analysis tool. Early ideas of this approach can be found in [NAC01, Nel99].

To demonstrate and validate the approach, two case studies were developed and are presented in Chapter 8. The first case study was adapted from [KPS97] and involves a power supply network. The case study involves problem-oriented analysis, describing an application in terms of its subproblems, and formalizing some subproblems to perform formal analysis.

The second case study involves fire detection and prevention in the Algonquin Park. It is spread throughout the dissertation and is used to explain each problem frame in detail, and to illustrate informal problem analysis. In Chapter 8, parts of this case study are formalized to illustrate topological map descriptions and property analysis.

1.5 Contributions

Thesis statement

The definition of classes of geographic problems and their formalization provides the basis for an approach to requirements description of geographic applications that focuses on problems and that allows informal and automated formal problem analysis addressing issues related to the failures of geographic applications and to naive geography.

Specific contributions of this dissertation include:

- an approach to informal problem description and analysis of geographic requirements;
  - definitions of typical classes of common geographic problems;
  - a detailed description of each geographic problem as a problem frame;
- an approach to formal description and analysis of geographic requirements;
  - formal descriptions of the geographic problem frames;
  - formal descriptions of specialized requirements and specifications;
  - formal map domain descriptions;
- a demonstration of both informal and formal problem description and analysis through two case studies.

It is important to note that the informal approach does not consist of a specific *use* of problem frames, but rather an *extension* of problem frames, specialized to geographic problems. A problem frame is *used* to describe a specific instance of a problem; in contrast, the geographic frames each represent a *class* of problems, not specific instances. The geographic frames are then used in the case studies to describe concrete instances of problems.

1.6 Related Work

In this section, a discussion of work related to this dissertation is presented. The related work is compared to the geographic problem-oriented approach introduced in this dissertation. Most of the related work focuses on informal description of maps.

1.6.1 Informal description and analysis of geographic requirements

Most geographic applications are information systems, in that their main purpose is to analyze and report on the status of a map which represents the real world. Most of the requirements engineering methods for information systems focus on data analysis and description [NE00]. The geographic area is no exception. There are many different notations and methods for describing and analyzing geographic applications. Some of these notations are extensions of entity-relationship models, while others extend object-oriented notations. These notations are suitable for describing map domains. However, unlike the approach introduced in this dissertation, none of these notations are problem-oriented. They can be used to describe problem domains, but not the problems themselves. Some do not specifically address requirements descriptions, but focus on conceptual modeling of geographic databases.

Extensions of the entity-relationship model

Four approaches rely on extending the entity-relationship model to describe geographic information: GISER [SCG+97], STER [TJ99], GEO-ER [HT97], and Calkins [Cal96].

GISER [SCG+97] extends the enhanced entity-relationship model [Nav92] introducing fields that are associated with discretization and interpolation models. This is done in order to unify the field and object representations of geographic space Fields and objects are explained in Appendix B.6.2.

STER [TJ99] is an icon-based extension to the entity-relationship model that includes spatiotemporal entities, attributes, and relationships. It supports spatial elements such as point, line,

and region and temporal elements such as existence time, valid time, and transaction time.

GEO-ER [HT97] attempts to model spatially-dependent attributes, position, spatial relations, and different views of geographic data.

Calkins [Cal96] extends basic entity-relationship models to handle spatial elements, multiple representations, temporal representations, and traditional coordinate and topological attributes associated with spatial elements.

Extensions of the object-oriented model

There are many approaches to geographic data modeling and analysis that are based on the object-oriented paradigm. A short description of each approach is given, highlighting its unique features.

GeoOOA [KPS96, KPS97] is a requirements engineering method that extends the object-oriented analysis method of Coad and Yourdon [CY91]. It provides spatial class types, topological whole-part structures, network structures, and temporal classes.

OMT-G [BLD99, BDL01] is an object-oriented data modeling notation that extends the Object Modeling Technique (OMT) [RBP+91]. It provides primitives for modeling the geometry and topology of spatial data, multiple views of objects, and spatial relationships. The approach enforces the spatial integrity constraints on the data. For example, it is possible to enforce the fact that a town's administrative regions must be contained within the town's limits.

GeoFrame [FI99] is a conceptual framework for modeling geographic data. It extends the Unified Modeling Language (UML) [BRJ98] and can be used to define new analysis patterns for geographic applications. A geographic analysis pattern shows a class of common spatial elements and their interrelationships. An analysis pattern describes a problem, its context, a solution, and an example of the problem. Though the idea of analysis patterns is, at first glance, similar to problem frames, the patterns focus mainly on map domain descriptions. The problems that analysis patterns address are related to common problems in describing maps, not to problems that are solved using maps.

The *Perceptory* tool [Béd99] is a CASE tool that implements a spatiotemporal plug-in for visual languages that is used to extend the UML. It is based on the Modul-R model which provides a set of geometric types through graphical symbols, allowing multiple views of the same entities [BCMM96].

Tryfona *et al.* [TPH97] also extend the OMT notation with the constructs of spatial aggregation or partition and spatial grouping or coverage. Other approaches and notations that are based on object-oriented modeling can be found in [PSZ99], [GR93], [Güt94], and [OPM97].

Summary

According to [FCTJ01], these notations include the ability to describe some of the following issues found in geographic applications: the location of geographic elements; complex spatial elements formed of multiple parts; attributes; objects and fields; generalizations; element constraints; relationships among geographic objects such as topological, metric, and semantic; relationship constraints; and multiple views of elements. These issues are mostly related to the description of the map domain. Therefore, these notations and methods are not suitable for expressing and analyzing the problem that a geographic application is supposed to address. A discussion of map domain descriptions can be found in Chapter 4.

1.6.2 Formal description of geographic requirements

The only approach oriented towards formal description and analysis of geographic applications found in the literature concentrates on geographic data modeling [vdAB91]. The approach is based on the TM language. TM is a specification language suitable for specifying object-oriented databases [BBdB$^+$]. TM specifications are declarative and may contain data, methods, and constraints based on typed first-order logic and set theory. TM is used for spatial data modeling by defining and reusing a number of common classes such as Point, Line, and Region, and associated methods and constraints on these classes. The TM constraints are similar to the constraints used in the approach introduced in this dissertation to describe map domains. However, their goals are different. TM constraints are added to classes to provide a precise description of objects that belong to that class. When building a database, these constraints are enforced to ensure that every object in the database is valid. The TM approach uses these constraints to perform a limited type of analysis that focuses on type checking and database prototyping.

1.6.3 Formal descriptions of problem frames

There has been a previous attempt at formalizing problem frames [BKNS97]. The formalization was done using the RAISE specification language [Gro93], a model-based language derived from the Vienna Development Method (VDM) [Jon90] and Communicating Sequential Processes (CSP) [Hoa85]. Like the approach introduced in this dissertation, the formalization draws a distinction between three parts of each problem frame: the domains, the requirements, and the machine design. However, the specifications are general in nature, at a high level of abstraction that includes few constraints on each problem. For example, the formalization of the translation frame

states that there are languages, meaning functions, and translators, and that a translator is a machine that transforms strings of a language into strings of another such that both have the same meaning. The frame is not applied to any specific domain, but is left unconstrained to allow for many possibilities of domains. This approach is also not used for formal verification, but only for making more precise certain properties of each frame.

While the approach introduced in this dissertation is similar in nature, it represents frames that deal with specialized map domains, machines, and requirements, and provides a number of specific constraints that deal with common geographic problems. In addition, it is inserted in a framework that allows formal analysis of geographic problems.

1.7 Outline of this dissertation

The dissertation begins by introducing Michael Jackson's problem frames approach in Chapter 2, since a basic knowledge of the approach is required to understand the contributions of this work.

Chapter 3 presents a comprehensive set of geographic problems organized into classes, as well as some background on geographic concepts.

Because most geographic applications use a map to address a problem in the real world, the map and the real world are important domains which need to be described. Chapter 4 contains the various phenomena that may be used when describing the map and real-world domains.

Chapters 5 and 6 present the geographic problems described in Chapter 3 using the problem frames approach. Chapter 5 describes problem frames concerning the creation and manipulation of maps. Chapter 6 describes geographic problem frames that deal with using a map to solve real-world problems.

The formalization of several geographic problem frames and map domains and a list of properties that can be analyzed for each frame are presented in Chapter 7. Chapter 8 presents two case studies used to demonstrate the informal and formal problem analysis.

Finally, Chapter 9 presents the conclusions of this work, including a discussion about the approach and paths for future work.

Chapter 2

Problem Frames

This chapter contains an introduction to problem frames [Jac00b]. This introduction is intended to familiarize the reader with the technique, since the problem-oriented approach introduced in this dissertation is based on problem frames.

Problem frames assist in the description and analysis of problems. A problem frame conveys the essence of a familiar class of small problems. Given a new complex problem, an analyst studies the problem in the light of a collection of problem frames. He then tries to describe the complex problem by fitting each part of the problem into one frame. This gives the analyst the chance to isolate the parts and study one subproblem at a time.

In order to discuss problem frames, some concepts related to requirements and specifications need to be defined. Although these terms have been used in a broad sense, in the scope of this work they are used as defined by Jackson & Zave [JZ93, JZ95, Jac95c, Jac95b]. These definitions have been adopted by many members of the requirements engineering research community.

2.1 Requirements, specifications and programs

Requirements define the problem to be solved by a software system; they do not describe the software. Requirements are focused on the customer's needs. They describe a part of the real world that is of interest to the customer. This part of the world is called the *problem domain*, and is where the software is to produce the desired effects. *Requirements* are the effects that the system is to cause in the problem domain.

In addition to the problem domain, there is the *machine domain*, which encompasses everything related to the software system, that is, code, data, protocols, and so on. In the intersection

between these two worlds, the problem domain and the machine domain, lies the interface between them. The *specification* is a description about this interface. A specification is a set of rules relating how the software interacts with entities of the problem domain that are in direct contact with it. The specification is basically the interface bridging the gap between the machine and the real world.

Descriptions have two different moods: indicative and optative [ZJ97]. Indicative statements are those that point out facts or descriptive information about the problem domain. Optative statements describe what should be observed in the problem domain in the presence of the software that will be built. Both requirements and specifications are described in the optative mood. Figure 2.1 shows the relationship among the problem and machine domains.

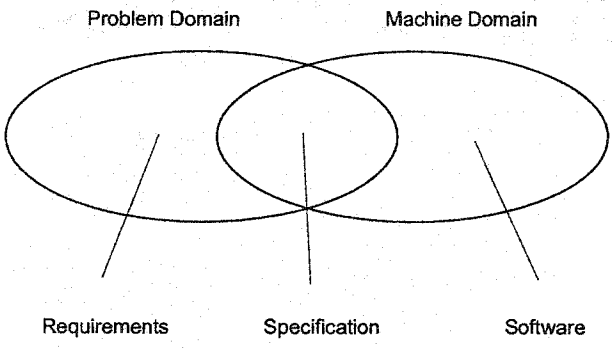


Figure 2.1: Problem and Machine domains

2.2 Problem domains and shared phenomena

In order to illustrate these concepts, an example concerning a Video Cassette Recorder (VCR) is developed in this dissertation.

Informal description of the VCR problem.

A simple VCR integrated with a TV accepts infrared signals coming from a user through a remote control. The signals are received through the VCR's infrared port which is connected to a computer. A software system for this computer is needed to process the infrared incoming signals and dispatch the appropriate signals to control the mechanism of the VCR. The VCR's remote control has the following buttons: *play, stop, rewind, fast-forward,* and *pause*. The VCR is always on and its mechanisms recognize signals to: start the motor that makes the heads and rollers spin at normal speed; start the motor at fast speed; stop the motor; engage and disengage the head so it reads from the tape; and set the direction of tape movement (forward or backward). (See Appendix A for a complete description of the problem.)

Domains are the parts of the real world that are relevant to describe the problem. Each domain is a collection of related phenomena. In the VCR problem description, at least three domains can be identified. The VCR mechanism is the domain that needs to be controlled. The operator in charge of the remote control represents another domain involved in the problem. The machine to be built is the third domain. It should allow a user from the operator domain to give commands to control the behaviour of the VCR domain. These three domains exhibit different phenomena.

Phenomena are elements that can be observed in the world [Jac00b]. The VCR domain is concerned about the motor, the heads and the tape, while the remote control user is concerned about issuing commands and observing their effects. The VCR mechanism accepts commands while the remote control user does not. The VCR does not initiate commands, it only responds to commands initiated by the remote control operator. It is the responsibility of the operator to initiate actions by pressing the remote control buttons.

Domains are connected to each other through *shared phenomena.* If some phenomena happen in one domain and are also experienced by another domain, this means that the two domains have shared phenomena. In the case of the VCR, the user of the remote control shares phenomena with the machine. The action of pressing any button on the remote control is a phenomenon experienced by the remote control user and the machine that is connected to the infrared port. The machine and the VCR also share some phenomena. For example, the machine generates signals to start and stop the motor and these signals are also experienced by the VCR.

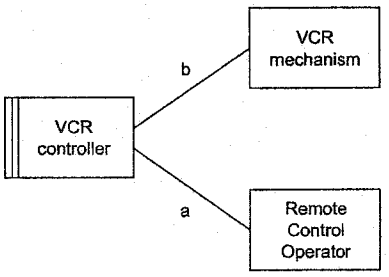


Figure 2.2: The VCR context diagram

Figure 2.2 presents a context diagram for the VCR problem. A *context diagram* shows the domains and the shared phenomena at their interface. Its purpose is to locate and bound the problem by showing the relevant domains and their connections. Each set of shared phenomena

that appears in the diagram has an identifier (e.g., *a, b*) and an annotation listing the phenomena.

The context diagram locates the problem in the real world. However, it does not describe what the machine is required to do. A context diagram that is enhanced to also picture the requirements and their relation with the problem domains is called a *problem diagram*. Figure 2.3 shows the VCR problem diagram.

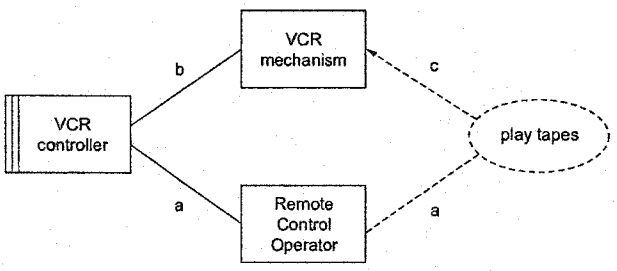


Figure 2.3: The VCR problem diagram

The requirements are represented in the diagram as a dashed oval. The requirements refer to some phenomena in the problem domains. The *requirement reference* is represented in the diagram as a dashed line connecting domains to the requirement. After all, requirements are all about the effects that the machine will produce in the problem domain. In the case of the VCR problem, the requirement is to play tapes in the VCR. The operator should be able to enter commands through the remote control and the effects of these commands should be observable in the VCR. The requirement refers to the commands issued by the operator and to the observable phenomena of the VCR domain. The arrowhead in the dashed line with phenomena *c* means that the requirements not only refer to those phenomena but that they also constrain them. In this problem, the requirement constrains the behaviour of the VCR allowing it to be in one of the following modes: *playing, pausing, rewinding, fast-forwarding, rewinding with image, fast-forwarding with image*, and *stopped*. These are effects that should be noticed in the VCR as commanded by the operator. The requirements description should explain which effect is expected after each button in the remote control is pressed. Note that in this problem, the requirements do not constrain the operator.

The annotations in the problem diagram are also more detailed. Each set of shared phenomena

has a prefix (e.g., VM! which stands for VCR mechanism) that indicates which domain controls or causes the phenomena to happen. In the VCR problem diagram, pausing is a shared phenomenon controlled by the VCR mechanism.

In order to analyze and describe the VCR problem, the domains, their phenomena, the requirements and the specification need to be studied and documented. See Appendix A for these descriptions.

2.2.1 The relation between requirements, specifications, and domains

When describing and analyzing a problem, requirements, specifications and domain properties have different purposes and roles. Relating these three descriptions allows the analyst to argue that the requirements will be satisfied.

Informally, if the real world acts as described by the domain properties (D) and the machine interacts with the real world as described by the specification (S), then the effects described by the requirements (R) are observable in the real world [GGJZ00]. Formally: $S \wedge D \rightarrow R$

This argument can be applied to any problem description. If the above expression does not hold, then the requirements were not satisfied. Thinking about this argument helps to find flaws in the descriptions. A mathematical proof that the requirements hold from the specification and domain descriptions can also be developed. However, this requires a formal description of each of the three aspects involved.

In the VCR problem, the argument can be informally stated as:

"If the operator presses some buttons on the remote control, the machine will discard commands that are not sensible and process sensible commands, causing some events in the VCR; if the VCR mechanism is working properly as expected, the desired effects can be observed in every case."

2.3 Problem frames

The VCR problem is just one example of a class of similar problems. The main concern of problems in this class is to build a machine that allows its operator to control the behaviour of a problem domain. They all have the same pattern. This core idea of the commanded behaviour problem is captured in a problem frame. "A problem frame is a kind of a pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interface and requirement" [Jac00b]. The problem frame is represented by a diagram that shows the relationship between the domains. Figure 2.4 shows the commanded behaviour frame diagram.

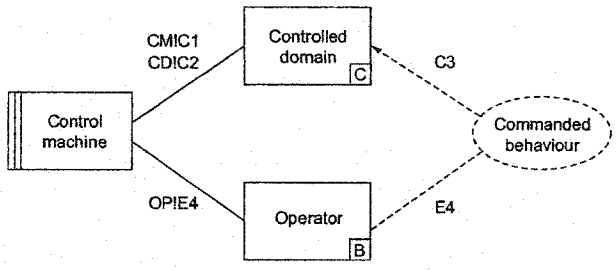


Figure 2.4: Commanded behaviour problem frame diagram

A problem frame diagram is similar to a problem diagram, but it does not express a particular problem or domain. It is a general diagram that captures the essence of the problem class. It shows the involved parts with relevant names that explain their involvement in the problem, the connections between the parts and the requirement. The problem frame diagram introduces two new symbols:

- a classification of the problem domains marked in the lower right corner of the domain box. The operator is a biddable domain and is represented by a *B* while the controlled domain is a causal domain and is represented by a *C*. The machine is always a causal domain, so it does not need to be marked. The other type of domain is called lexical. Their characteristics are presented in Section 2.3.3.

- a classification of the shared or referenced phenomena marked in the interface between the domains and the requirements. Each set of phenomena in an interface is identified by the class that they belong to and by which domain controls the phenomena. For example, CM!C1 means that CM!, which stands for control machine, causes the phenomena C1 which belong to the class of causal phenomena. All different types of phenomena are presented in Section 2.3.2.

Each problem frame captures the essence of a simple problem. However, realistic problems are usually much more complex. In order to understand a complex problem fully, it is helpful to decompose it into subproblems. Successful decompositions identify subproblems with the following characteristics:

- separation of concerns: each subproblem contains only the elements that are relevant to that particular problem, and isolate what is not important [Dij76]. Each element may play a role in more than one problem.

- partial completeness: each subproblem is complete, in the sense that it can be removed from the original complex problem context and still be a problem on its own.

However, it is usually not easy to find a successful decomposition, unless there is some guidance in the process of identifying potential subproblems.

Problem frames come into this scenario to guide the decomposition process, showing patterns of problems that occur frequently. When decomposing a realistic problem, the frames help in the identification of the parts of the problem as well as their connection. Subproblems not only match frames as a pattern, but their parts should also "fit" into the frames. Each frame provides enough information about the involved domains and phenomena to help recognize if a subproblem really fits into a frame, or if it is a misfit. Once a subproblem has been identified as an instance of a frame, its description becomes an exploration and documentation of the domain properties, shared phenomena, requirements and machine specification as seen in the VCR example presented in Appendix A.

Each frame helps to provide a partial description of the complete problem. There are several advantages to having multiple partial descriptions of a problem. Each description can be written in a notation that is more appropriate to that particular perspective [FS96]. Having multiple descriptions also provides higher separation of concerns, making the descriptions easier to read and understand. This technique is also useful in allowing the specifier to describe the problem in different ways so that different subproblems use the most appropriate phenomena of the domain for their description [Jac95a]. Existing literature [EN96, Jac95b] gives a more detailed discussion of the benefits of multiple partial descriptions.

A realistic problem is usually described as a composition of subproblems. The decomposition helps with analysis and understanding. In order to implement a machine that solves all subproblems together, bringing a solution to the original problem, the composition of the subproblems also needs to be addressed. During composition some difficulties may be experienced, such as inconsistencies between properties of the same domain that were described in two separate problem frames. However, the advantages of decomposition usually overcome the difficulties of composition.

2.3.1 Characteristics of problem frames

Decomposition may produce disjoint or overlapping parts. If the decomposition results in disjoint parts, they are called partitions. Otherwise the parts are called projections [Jac00b].

Decomposing a problem into subproblems according to problem frames produces projections.

A subproblem is a partial description of the problem domains, requirements and specification of the machine for the complete problem. Subproblems may have overlapping parts. For example, a problem domain may be part of more than one subproblem, and for each subproblem description the relevant phenomena of the problem domain may differ.

In the case of the VCR example, the VCR mechanism is a problem domain that is part of the commanded behaviour frame and is described in Appendix A.2 in terms of state phenomena related to the state of the motor, heads and direction of the movement of the tape. However, if the VCR example is made more realistic it may also address the problem of displaying the elapsed playing time of a tape in a small digital display. In this case, the second subproblem may be framed as an information display frame. The domains involved are the VCR mechanism domain and the display domain. The relevant properties of the VCR mechanism domain may now be related to phenomena that describe the current position of the tape, the end of tape sensor, and so on. When composing the VCR mechanism description, both projections of the domain have to be considered.

2.3.2 A classification of phenomena

The kind of phenomena that happen in a problem domain may provide hints as to the characteristics of the domain, and as a consequence, may help to determine an adequate problem frame fit. Phenomena fall into two large distinct categories: individuals and relations among individuals.

An individual is anything that has a unique identity and can be distinguished from all other individuals. Individuals are further classified as:

- Events

  An event is an individual occurring in time. For example, pressing the *play* button on the remote control is an event. Events are atomic, instantaneous and occur in a certain order in time. An assumption is made here that two events cannot occur at the same time. In this way, there is always a relationship between two events that establish the order in which they occur. If *Play* and *Rewind* are two distinct events, either *Play* happens before *Rewind*, or *Rewind* happens before *Play*. Events are abstractions of what really happens in the real world, since many details are often ignored in order to describe relevant individuals for each problem. For example, in the VCR problem, the event of pressing the *Play* button is considered instantaneous, but is not so in the real world. It involves at least three actions: the user presses the button; the infrared port recognizes the signal; and the signal reaches the computer. However, for the commanded behaviour problem, it is sufficient to consider the

event atomic and instantaneous. If the problem were, for example, to audit the capacity of the infrared port to recognize the correct signal from the remote control, a finer distinction between events would be necessary.

- Entities
  "An entity is an individual that persists over time and may change its properties from one point in time to another" [Jac00b]. Entities may cause internal state changes, may initiate events or may be passive and not cause any phenomena. For example, the motor, the heads and the tape are entities from the VCR mechanism domain; the user of the remote control is an entity from the operator domain. The tape is a passive entity while the user initiates events and the motor and heads cause changes in the state of the VCR mechanism. A state is a relation and is defined below.

- Values
  A value is an individual that does not change over time. For example, numbers and characters are values. The VCR display shows the number of hours, minutes and seconds elapsed since the beginning of the tape. This number is a value.

Associations among individuals are called relations. Each tuple of a relation may involve one or more individuals. However, tuples of the same relation have the same number of individuals involved. Relations are further classified as:

- States
  A state relates entities to values and can change over time. In the VCR problem, *MotorSpeed(Fast)* is a tuple of the state *MotorSpeed(x)* where x can be one of three values $\{On, Off, Fast\}$. *MotorSpeed(Fast)* holds (is true) only when the speed of the motor is *Fast*.

- Truths
  Truths are relations between values; they do not change over time. For example, *SpeedGreaterThan (Fast, On)* expresses the fact that the *Fast* value of *Speed* is always greater than the *On* value of *Speed*.

- Roles
  "A role is a relation between an event and individuals that participate in it in a particular way" [Jac00b]. In this work, roles are not used and will not be further explored.

These different classes of phenomena can be grouped to characterize two larger categories:

- Causal phenomena
  Any events, roles or states relating entities are causal phenomena since they can cause other phenomena and are controlled by a domain. In the VCR problem, all described phenomena are causal. Some are events (e.g., *SetMOff*) that are controlled by the machine and cause state changes in the VCR mechanism (e.g., going from *Playing* to *Pausing* state).

- Symbolic phenomena
  Any values, truths and states relating only values are symbolic phenomena since they symbolize other phenomena or the relationships among them. For example, the data of a spreadsheet containing values in different cells is a symbolic state that relates values. It does not cause changes to itself or to any other domain.

2.3.3 Domain types and their characteristics

Each domain has characteristics that determine how it can interact with other domains and with the machine domain. Domains can be classified according to their characteristics into three main categories:

- Causal domains present well-defined and predicted causal relationships among its phenomena. It is possible to calculate the effect of the machine behaviour when interfacing with a causal domain. Causal domains may control all, some or none of the shared phenomena at the interface with another domain. The VCR mechanism is a causal domain that controls internal state changes in response to events caused by the machine. It also controls shared state with the machine (e.g., *Pausing* state).

- Biddable domains consist mostly of people. No external action controls their internal motivation. The operator in the VCR problem is a biddable domain. As mentioned before, instructions recommending the behaviour of the operator while operating the machine should be provided. Although the operator is expected to follow the procedure, there is no action that would guarantee the operator's behaviour.

- Lexical domains consist of any physical representation of data. A lexical domain brings symbolic phenomena into physical existence. It is a combination of causal and symbolic phenomena. The causal phenomena allows the data to be created physically in the storage device in a computer. The symbolic phenomena allows the bits or signals to be interpreted as symbols that represent a piece of data. In this sense, a lexical domain can be seen as a

causal domain (e.g., read, write, load, and store events) or simply as a structure of symbolic phenomena (e.g., records, fields, and values).

2.3.4 Summarizing the frame diagram notation

The diagram below presents the data capture frame that is explored in Chapter 5. It is used here to summarize the notation of the problem frame diagrams.

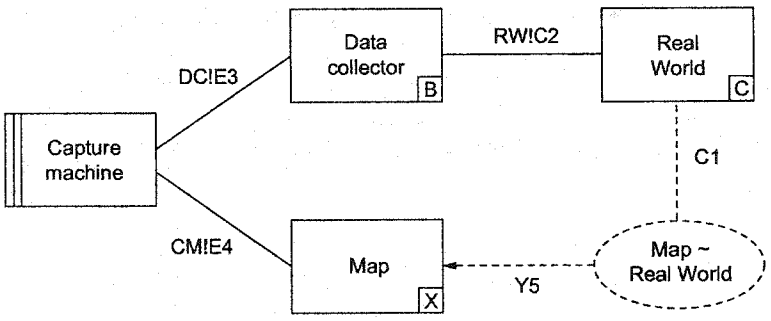


Figure 2.5: A sample frame diagram

The frame diagrams follow the notation presented by Michael Jackson [Jac00b]. Each rectangle is a domain, a collection of individuals that are relevant to the problem. In the data capture problem frame, the real world, the data collector, and the map are domains.

A rectangle with a double line at the left is the machine domain, the system to be built. A dashed ellipse is a set of requirements, propositions to be satisfied by the machine. In the case of the data capture frame, the requirement is that the map should correspond to the real world.

A line connecting two domains represents shared phenomena that belong to the two domains.

A dashed line connecting an ellipse to one or more domains indicates that the requirements are related to those domains. It is called a requirement reference. The arrowhead indicates the domain being constrained by the requirement. In the case of the data capture frame, the requirement constrains the map by making it correspond to the real world.

Phenomena and their classification are marked at the interface of the domains or at the requirement reference: (C) is used for causal phenomena, (Y) for symbolic, and (E) for events. In the sample frame, C1, C2, E3, E4 and Y5 are shared phenomena. For identification purposes, numbers are assigned starting at one. The names for shared phenomena at the interface of the domains are also prefixed with the initials of the domain that controls them. In the sample diagram, DC!E3 means that E3 events are caused by the Data Collector (DC!) domain. The exclamation mark after

the initials of the domain name indicates the domain that controls the shared phenomena.

Each domain in the diagram is also marked with the type of the domain: (C) for causal domains; (B) for biddable domains; and (X) for lexical domains. In the sample frame, the real world domain is causal, the map is lexical and the data collector is biddable.

Chapter 3

Classes of geographic problems

This chapter presents an overview of geographic applications, identifying classes of common subproblems that these applications usually address. These classes of problems form the basis of the problem-oriented approach for describing and analyzing requirements of geographic applications [NCA01].

A previous, simpler set of classes of geographic problems was presented in [Mag91]. The classes described in this chapter are based on this simpler set, which was expanded to include a wider range of problems found in the geographic area.

Geographic applications are computer-based systems that deal with geographic maps. They have evolved significantly during the last 20 years. In the past, the computer was used only as a way to store information and display it graphically to geographers. Today, geographic applications are also tailored to end users and may serve not only the purpose of visualization, but also analysis, synthesis, simulation, and decision support [ACG+97].

Geographic applications are used in a wide range of areas. Cartography, environmental sciences, urban planning, transportation, emergency response, social studies, agriculture, and community development are a few examples of areas where the advances in geographic applications have had an impact.

In all of these different areas, the data that are used have some type of explicit or implicit associated geographic reference (see Appendix B.5). An explicit geographic reference might be a Cartesian or latitude/longitude coordinate, and an implicit reference might be the name of a city or a physical feature such as a mountain or a lake. In many cases, it is possible to derive an explicit reference from an implicit reference.

This geographic reference has proved to be an effective means of linking diverse data sets.

This principle is what makes geographic information attractive to so many different areas.

3.1 Geographic applications

Geographic applications draw relationships between independent facts in a particular region of interest in the world, provided that the facts are georeferenced. A georeference is a way to identify a fact in the world according to its location.

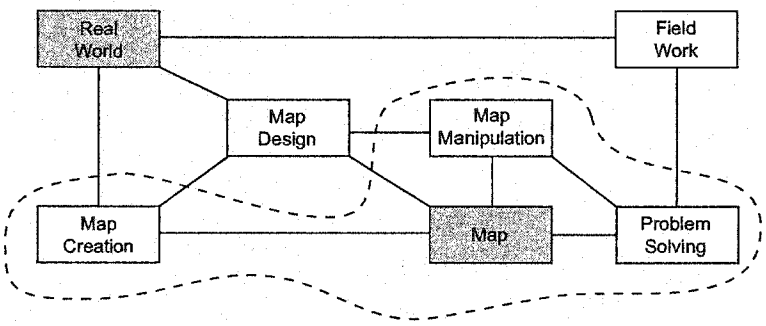


Figure 3.1: A typical geographic application. The tasks inside the dashed line can be automated with the use of a computer system. The ones outside the dashed line are not part of a computer-based system, since they require human labour and creativity.

The diagram in Figure 3.1 shows a general view of a typical geographic application. A geographic application usually addresses a problem from the real world using a map (see Appendix B.3). In the diagram, this is referred to as a *problem-solving* task. Examples of problems dealing with maps include finding a route between two locations, analysis of farmland usage, and tracking bird migration patterns. However, in order to deal with the specific problem-solving task, the map must be *designed, created,* and *manipulated.* Some *field work* may also be required to ensure that the real world reflects the decisions made using the map. The field work task and the map design task are usually not automated by a computer-based system, since they require human labour and creativity.

Figure 3.2 details the types of problems that each main task in a geographic application may involve. Map design usually involves *abstraction* and *symbolization.* Since a map is a representation of the real world, it is necessary to abstract the relevant facts from the real world that should appear in the map. A decision about which symbols are going to represent the various facts or phenomena from the real world must also be made. The tasks of deciding what goes into the map (*abstraction*) and how it is going to be represented (*symbolization*) are usually not automated.

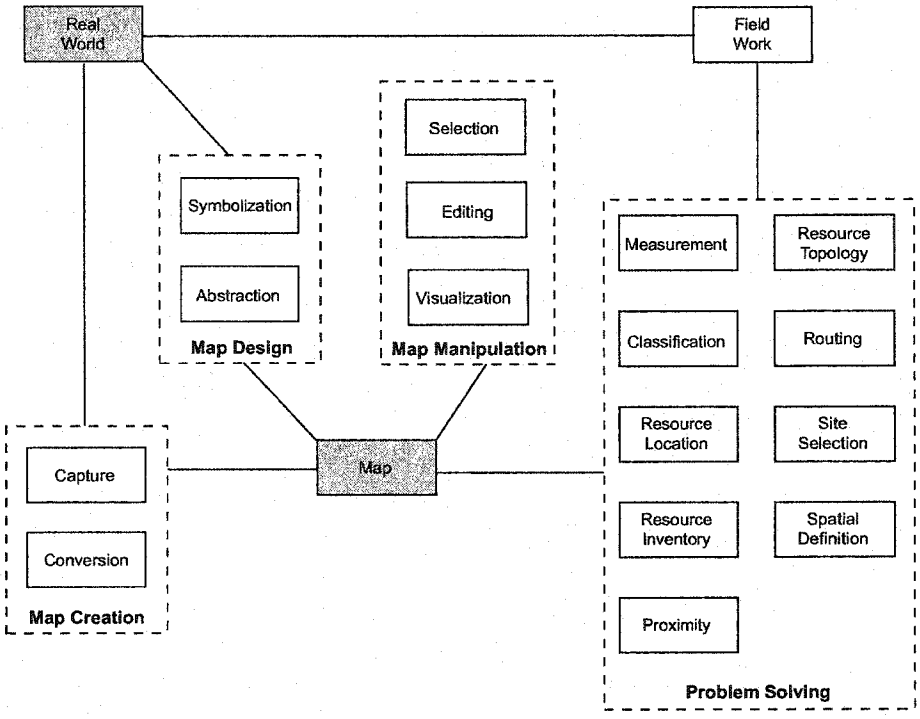


Figure 3.2: A breakdown of the specific kinds of problems that constitute each main task category shown in Figure 3.1

To create a faithful representation of the real world, it is necessary to *capture* data from the environment in order to build a map. In some cases, parts of the data already exist, but in different projections, datums, coordinate systems, or levels of detail (see Appendix B.5). Changing the level of detail is called map generalization, which may involve feature simplification, elimination, aggregation, or exaggeration to make the representation appropriate to the scale and problem at hand. Data *conversion* is the process of transforming diverse data sets so that they are compatible with each other, and suitable for problem-solving tasks. Data conversion tasks are not concerned with the meaning of the features in a map, but with their geometric and topological characteristics.

A map that will be used in a problem-solving task must also be manipulated. Most applications need to display the map and the results of the problem-solving tasks. This is referred to as the *visualization* task in the diagram. The results may need to be added to the map through *editing*. The editing task, like the data capture task, may serve the purpose of creating a map. The difference is that entities added through editing, such as political boundaries, do not necessarily have corresponding physical, real-world phenomena. In addition, most geographic applications

require the user to select points or regions in the map through direct manipulation. This task is called *selection*.

There are several kinds of problems whose solutions require a map. These are: *measurement* of lengths and areas; *classification* of geographic phenomena; *resource inventory*; *resource location*; determining the *proximity* among entities; determining *resource topology*; *routing* in a network; *site selection*; and *spatial definition*. In order to create the map, the relevant problem-solving tasks should first be analyzed and fully understood so that the map contains the appropriate phenomena.

Note that in this diagram, the person responsible for each task has been omitted, but there are different people involved in the different tasks. For example, data from the real world are usually acquired by an individual or by an automated system controlled by an individual.

The diagram of figure 3.2 shows common tasks of geographic applications. However, some of these tasks may not be necessary, depending on the specific problem-solving task at hand. For instance, if the data is already available from a library, data capture and conversion tasks can be eliminated.

3.2 Geographic problems

Geographic problems can be divided into three categories:

- **Problems that make use of a map**
  In the diagram in Figure 3.2, these are referred to as the problem-solving tasks. Examples are finding measurements of distances, finding locations of resources, classifying entities according to some criteria, or determining routes.

- **Problems concerning the creation and manipulation of maps**
  In the previous diagram, the following tasks fall into this category: capturing data from the real world; converting, editing, and visualizing maps; and selecting points and regions through direct manipulation.

- **Problems that require human labour or creativity**
  There are a few tasks in the previous diagram that cannot be automated. Abstraction, symbolization (map design), and field work fall into this category.

3.2.1 Problems that require human labour and creativity

The main concern of this work is to analyze and describe problems for which a solution based on a computer system can be built. The problems in this category, although important, will not be detailed in this dissertation because a computer system cannot be built to automate these tasks. They are briefly described here to provide a better sense of all the tasks involved in building a geographic application.

1. **Abstraction**

   Abstraction deals with the decisions about the contents of the map that needs to be created to reflect the real world and to serve as a basis for the problem-solving task.

   The abstraction task can be summarized by four questions:

   - **Where?** - deciding on a region of concern in the world
   - **When?** - deciding on a time frame of interest, so that data reflecting the appropriate time frame can be gathered
   - **Why?** - deciding the purpose of the map
   - **What?** - deciding the phenomena that need to be mapped

2. **Symbolization**

   Symbolization is concerned with the decisions about assigning symbols to represent real world phenomena in a graphical map that is going to be visualized. The symbolization task depends on:

   - the level of measurement (qualitative vs. quantitative) used when acquiring the data for the map.
     To show different classes of qualitative data, symbols with different shapes, colors, pattern arrangements, or orientations can be used. To express the difference in quantitative data, symbols with a magnitude connotation are more appropriate. Size, texture, and intensity of the color emphasize the quantitative differences.
   - the dimensionality of the discrete phenomena.
     If the phenomena to be represented is discrete, choosing the dimensionality from point, line, or area representations is also part of the symbolization process. Depending on the purpose and scale of the graphical map, the same phenomena in the real world may be emphasized as an area feature or simplified to a point representation.

3. **Field Work**

   One of the problem-solving tasks is to select a site on the map that is suitable for building or placing a resource. The field-work task is to change the real world in order to build or place the particular resource at the selected site. This task usually requires human labour because it is rarely the case where a computer system can fully automate this task. Examples of field work may vary from placing a pole for the electric power network, to building a hospital in a strategically selected site. The goal of the field-work task is to make the real world reflect the decisions made using the map.

3.2.2 Problems that make use of a map

The problems in this category are concerned with the meaning of the symbols in the map as opposed to simply treating them as geometries. The problems also involve spatial relationships between the symbols. They all rely on the existence of a map that reflects the real world. They are all specializations of the problem-solving task in the diagram presented in Figure 3.2.

1. **Measurement:** problems concerned with the calculation of measurements that can be derived from locations, such as distance or length, angle, or area. Calculating any of these values involves a precise definition of the term. Euclidean distance, distance along a great circle of the earth, and distances of locations in different Universal Transverse Mercator (UTM) zones are examples that require different calculations. Please see Appendix B.5 for an explanation of different coordinate systems and map projections, as well as of their effects over the calculation of measurements.

2. **Classification:** problems concerned with grouping discrete or continuous phenomena that present similar quantitative or qualitative characteristics. The number of classes, the range of each class, and the use of constant vs. variable intervals for the classes are factors that influence the result of grouping.

3. **Resource Location:** problems involving the geographic position of resources in the map. These problems address questions of the form "Where is it?". Given a resource, the task is to find the location of the resource in the map.

4. **Resource Inventory:** problems concerned with the identification of the resources that satisfy a topological relation with a given region in the map. Examples of this problem may involve determining which resources are inside, overlap, meet or cover the given input region.

5. **Proximity:** problems concerned with resources constrained by a certain *distance*. The task is to find resources that satisfy a distance requirement from a given resource or location. Examples of this problem include finding the nearest or farthest entities.

6. **Resource Topology:** problems involving boundaries and interior parts of resources and their relationship with other resources. The task is to determine existing resources that satisfy a topological relationship with a given set of resources in a map represented according to the connected object model (see Appendix B.6.2). Examples of topological relationships are overlap, adjacency, inside, or enclosed.

7. **Routing:** problems concerned with a special case of topology involving networks where nodes are connected through edges. Examples are found in the utilities industry, which involves distribution and management requirements such as finding the shortest path.

8. **Site Selection:** problems concerned with selecting on a map regions constrained by a certain *distance*. The task is to find regions that satisfy a distance requirement from the given resources.

9. **Spatial Definition:** problems involving the definition of a new region or field on the map in terms of existing map entities. For example, the task may be to calculate geometries to define a new region for the map in terms of the intersection, union, or difference of the given regions. The same idea applies to defining new fields in terms of the values of existing ones.

3.2.3 Problems concerning the creation and manipulation of maps

The problems in this category are mostly concerned with the geometries and formats of maps. They relate to the construction and management of maps.

1. **Data Capture:** problems that involve creating an accurate representation of the real world by gathering data from the environment and storing them in a computer. The necessary data and method of acquisition both depend on the problem-solving task at hand. There are basically three methods of data capture:

   - Digitizing paper maps
     In this case, the data from the environment have already been collected but are not stored in an electronic digital format. The digitizing process can be done manually,

where a person operating a digitizing table that holds the paper map records coordinates for each geometry[1] in the map. Digitizing consists of placing a precise cursor over each point of each geometry in the map and clicking a cursor button to capture the coordinate for that point, creating a vector map [RMM+95]. Automatic digitizing involves the use of precise scanners that create a digital image of the map in raster form (see Appendix B.6.2 for explanations about raster and vector maps).

- Surveying the environment
  Ground surveys gather the position of environmental features by determining their distance and directional relations with respect to other features [MM98]. This process uses a reference datum (see Appendix B.5.1) to locate the features precisely; this is important when surveying areas large enough that the earth's curvature has to be considered. Modern survey methods use electronic devices, such as the Global Positioning System (GPS), that can determine the coordinates of a location on earth. By receiving signals from several satellites, a mobile GPS receiver can compute its latitude and longitude coordinates. Multiple receivers are used to increase positioning accuracy when surveying. The maps created with ground surveying are vector maps.
- Remote sensing the environment
  There are basically two techniques of remote sensing used to capture data from the environment. One is based on a photographic camera. Aerial photographs and photogrammetry are examples of this technique. The second is based on satellite scanners that record electromagnetic energy and are capable of sensing waves that are not visible by photographs or the human eye. Many scanner images are produced in the thermal infrared and microwave portions of the spectrum, allowing better perception of the environment. Some scanners are passive, in that they merely observe the energy that exists in the environment. Others are active sensors, which first transmit an electromagnetic signal and then measure the energy reflected back from the landscape. Radar, for example, is an active sensor. All remotely sensed images are rasters.

When data is gathered directly from the environment and stored in digital form, it is called digital geographic data. Surveying and remote-sensing techniques produce digital geographic data. If geographic data are first used to make a map, and the map is later digitized, the result is called digital cartographic data. Digital cartographic data is usually

[1] here the definition of geometry refers to " the relative arrangement of objects or constituent parts, as specified by geometrical quantities." (Oxford English Dictionary)

more abstract than geographic data, because it ignores any element that was not relevant or impossible to capture given its size, map scale, and resolution.

Resolution is also an important concept when dealing with data capture. For all three techniques, the concept of resolution is related to the precision of the instruments and devices used. For paper maps that are digitized, the nature of the cartographic data presents another limit in the resolution. Considering that the smallest mark a cartographer can physically draw in a paper map is 0.5 millimeters, the resolution of the paper map at each scale is limited to this value. For example, at a scale of 1:50,000, the resolution is 25 meters ($0.5mm \times 50,000 = 25m$). This means that any real world feature smaller than 25m is shown in the map symbolically with its size exaggerated. The size of objects that can be detected by remote sensing techniques depends on the resolution of the devices. "The rule of thumb is that the smallest dimension of a real world feature must be at least twice as great as the system's resolution before there is a good likelihood of detection" [Cam98].

Usually, when data is captured in vector format, the resulting map follows the isolated model (see Appendix B.6.2). However, when the problem-solving task requires a map following the connected model, topological relationships may be captured along with the geographic data. This is only possible when digitizing paper maps or surveying the environment.

All three of these techniques have different precisions and costs. The decision of which technique to use depends on the problem-solving task for which the map is being created. First it is necessary to know the problem that needs to be solved so that the necessary data is captured in an appropriate manner.

2. **Data Conversion:** problems concerned with the conversion of diverse data sets into a single, compatible format so that they can be combined. Some problem-solving tasks may involve dealing with data that have previously been captured, but are not in the appropriate format for the problem-solving task. In this case, the data need to be converted before they are used. This task may require conversion of:

   - datum (see Appendix B.5.1)
   - map projection (see Appendix B.5.3)
   - coordinate system (see Appendices B.5.2 and B.5.4)
   - resolution (in the case of raster maps)
   - data-model format (vector to raster, raster to vector, isolated to connected, and connected to isolated) (see Appendix B.6.2)

- data-encoding format (e.g., between Drawing Exchange Format (DXF) from AutoDesk and Spatial Data Transfer Standard (SDTS) from the U.S. Geological Survey (USGS) or between DXF and the Topologically Integrated Geographic Encoding and Referencing System (TIGER) from the U.S. Bureau of the Census)
- level of detail

In many of these cases, the meaning of the map is not considered because the conversion is mostly involved with the geometry, positioning, or format of the features.

Concerning the data model used to capture data and store the map, it is common to first build a map following the isolated model and then convert it to the connected model. This conversion process is commonly referred to in the literature as the process of *building the topology* of the map [RMM+95]. In other words, it is the process of discovering the topological relationships among the map objects.

Digital maps do not have an embedded scale — they can be visualized at any scale. The concept of scale is relevant in the visualization task. However, there are cases when visualizing a map in different scales may require data conversion by changing the level of detail of the map.

Data visualized at a very large scale may look overly simplified if the resolution or precision used to capture the data did not detect features at that level of precision. On the other hand, data visualized at a very small scale must be converted to a more abstract level of detail.

This problem is called map *generalization* and may involve the following tasks:

- simplification

  Simplification is the task of reducing the amount of information in a map without compromising the essential geographic characteristics of the mapped phenomena. It can be achieved by eliminating features according to their relative importance, size or density; smoothing lines and areas by straightening minor fluctuations; abstracting shapes by replacing complex natural shapes with more compact and easier-to-construct objects; aggregating features; or changing the dimensionality of the feature's representation (e.g., an area feature becomes a point in the generalization).

- exaggeration

  Exaggeration is the process of enlarging or altering a feature in order to capture its real-world essence. It is necessary to prevent important features from disappearing as the map scale decreases.

Although the tasks in map generalization can be automated, they require human intervention in deciding which tasks should be performed and in judging the quality of the results. Map generalization may also be required when combining geographic and cartographic data.

3. **Visualization:** problems involving the display of a graphical map on a computer screen or printer. The digital map is a simplified model of the real world, and lies on a computer. The visualization task transforms the digital map into a graphical map that can be presented on a computer screen or printer. The graphical map resembles the traditional paper maps. The differences are in the presentation media — if the computer screen is used — and in the process of creating new graphical maps. Having the digital map stored in a computer allows for easy prototyping of graphical maps until a satisfactory result is achieved.

   The graphical map differs from the digital map mainly because the graphical map has a scale. Deciding on the scale of the map is part of the visualization task.

   The task also involves determining what information needs to be displayed, since only part of the digital map may be considered to appear in the graphical map. The decisions made during the symbolization task are also used here to guide the appearance of the graphical map.

4. **Editing the Map:** problems involving changes and updates to the map. Editing the map may be necessary to:

   - correct errors made during the data capture process made through manual digitizing;
   - add abstract geographic features that are not visible (e.g., political divisions), and therefore, not captured by survey or remote sensing methods;
   - change or update non-geographic attributes.

5. **Selection:** problems involving direct manipulation of the map by a user through an input device such as a mouse, keyboard, digitizing table, stylus, or touch screen. The task essentially involves taking input from a device and converting the input into meaningful geographic coordinates. This is necessary when the user needs to select a point or region on the map to perform some operation.

3.3 An example of a geographic application

In this section, a natural-language description of a geographic problem is presented. A client may provide this description as the basis for the development of a geographic application. This problem description will be used in the following chapters to illustrate the geographic problem frames.

The Algonquin Park Administration wants to improve its fire detection and its help-dispatch process. They want to install emergency-alert transmitters in camping sites and trails to allow passersby to notify the central station in case of a fire emergency. The park authorities believe that fires that are detected at an early stage are easier to control, do not spread to a large area, and kill fewer animals and plants. The park is partitioned into many regions with independent fire stations. Each fire station is responsible for patrolling and dealing with the fires that happen in its region of coverage.

A computer system is needed at the central station to determine which fire station is responsible for providing emergency help when it receives an alert signal from the transmitters. The system notifies the appropriate fire station with the location of the alert signal. Each station has a computer terminal with the park map, in which the user can zoom in and out of the map and select the viewed region. The system must be able to receive input from a mouse, whereby the user can select a point on the map shown on the screen as the center of zooming or viewing. Fire notification is done through a blinking orange circle displayed on the station's terminal indicating the location of the alert signal on the park map.

A digital map of the park needs to be created and should contain the boundaries of the park and of the regions covered by each fire station, as well as the locations of the fire stations, the emergency transmitters, the roads, main trails, and bodies of water. Figure 3.3 shows a map of the Algonquin Park containing the required resources. The park region must be surveyed by a team using GPSs or similar devices. The locations should be recorded using UTM coordinates. The park boundaries can be obtained from the Survey and Mapping Branch of the Ontario Ministry of Natural Resources in raster digital format. The boundaries of the fire-station-coverage regions can be input after the rest of the data capture is done, because these boundaries can be placed at any location inside the park that is more convenient for the fire station patrols.

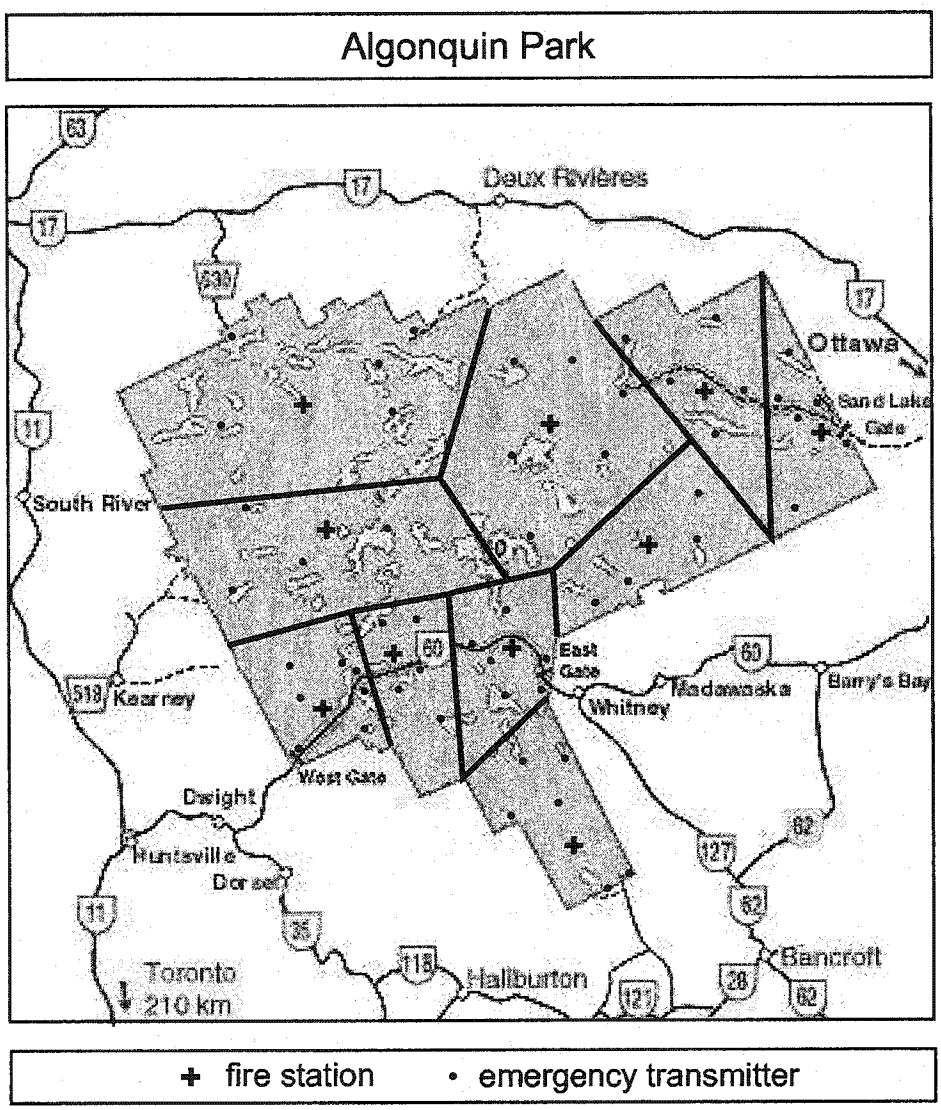


Figure 3.3: Algonquin Park map

Chapter 4

Informal description of geographic domains

This chapter contains an overview of map and real-world phenomena, which are necessary to describe and analyze geographic problems. All geographic problems deal with maps. Therefore, the map domain always needs to be appropriately described. The map phenomena necessary to solve a problem may vary from problem to problem. This chapter presents the different map phenomena and refers to some existing geographic notations that are adequate to describe these phenomena.

Since geographic applications use a map to address a problem from the real world, the world itself is also an important domain that needs to be described. This chapter also presents the phenomena that are usually present in real world descriptions.

4.1 Describing map domains

A map is a model of the real world. Some problem frames are concerned with the creation and manipulation of this map, and some only use the map as a way to find out information about the real world.

A map problem-domain is a lexical domain because it consists of data about the geographic phenomena of the real world. A map is a data model which is created by "discretizing the complex natural and man-made environment so that it can be examined within the computer" [Kem93]. A map is an abstraction, generalization, or approximation of the real world constrained by the finite and discrete nature of computing devices. The map is an *inert* domain. It will change its state only

in response to external events. It does not initiate any events or change its own state.

A map has several important characteristics like datum, projection, coordinate system, scale, (see Appendix B for the definitions of these terms) and the region of the earth's surface being represented. These characteristics of the map are described as state phenomena. They usually do not change with time but they characterize the state of the map.

Appendix B also presents a classification of maps according to the type of model used to perceive the real world and to gather, store and manipulate geographic information. Map phenomena either refer to fields or objects, and objects may be isolated or connected to each other.

4.1.1 Describing map fields

If the real world is perceived as continuous fields, the map should be a model, an approximation of these continuous fields.

"The linkage between continuous reality and its representation in the computer is achieved by:

1. dividing continuous space into discrete locations (cells, lines or points) for which discrete values can be measured and recorded, and

2. establishing a rule for interpolating unknown values between these locations." [Kem93]

Map fields are finite, discrete relations (functions) from sampled locations to measured values. A field is a total function $f : R \rightarrow V$, where R is the finite set of sampled locations (expressed as coordinates) that cover a region and V is the set of values that each location may assume.

Determining the value of the field for a location that has not been sampled requires an approximation. This approximation depends on the method by which the region R is partitioned in different cells or by which the phenomena are sampled in the region R.

This method also defines a more refined classification to describe the type of the field [Goo92, Kem93]. For each type, the discrete locations may have different names, shapes and characteristics:

- Grid of Cells (figure 4.1a): the region R is partitioned in rectangular cells. The value of the field for each location within one cell is constant. The value of the field may change abruptly at the edge of the cells.

- Adjacent Polygons (figure 4.1b): the region R is partitioned into irregularly shaped polygons. The value of the field within a single polygon is defined as a constant and changes abruptly at polygon edges.

- Triangular Irregular Networks (TINs) (figure 4.1c): the region R is partitioned into triangles of different sizes. The value of the field is defined by measurements only at triangle nodes. For any other location on a triangle face, the value can be calculated directly from the values at the nodes. The value of the field is defined by a linear function. There is no abrupt change in value at the edges of the triangles. However, there is an abrupt change in slope at the edges, because each planar triangle face has a different linear function.

- Grid of Points (figure 4.1d): the phenomenon is measured at the intersections of a rectangular grid over region R. Unlike the previous field types, the interpolation function is not clearly defined. The map designer is responsible for defining the interpolation function for any given location that does not belong to the sampled points.

- Irregular Points (figure 4.1e): the phenomenon is measured at some selected locations of region R. The interpolation function should also be defined by the map designer. The locations may be selected according to their relevance for the phenomenon (e.g., representative locations for measuring rainfall) or they may be selected not accounting the phenomenon under consideration but other factors (e.g., locations of airport weather stations).

- Contour Lines (figure 4.1f): this type of field differs from the others, in that the value of the phenomenon is what determines the locations represented by the field. Adjacent locations which have the same value are connected by a line called a contour line. All contour lines are continuous and they do not intersect each other. Each contour line explicitly identifies all adjacent locations that exhibit its constant and desired value. The value of the phenomenon is defined only along the contour lines. The locations of the contour lines are determined by the phenomenon (e.g., rainfall) and the selected value for drawing the contour line (e.g., 5mm). The interpolation function for locations lying between two contour lines is not necessarily linear. However, its range is constrained by the values of the two bounding contour lines.

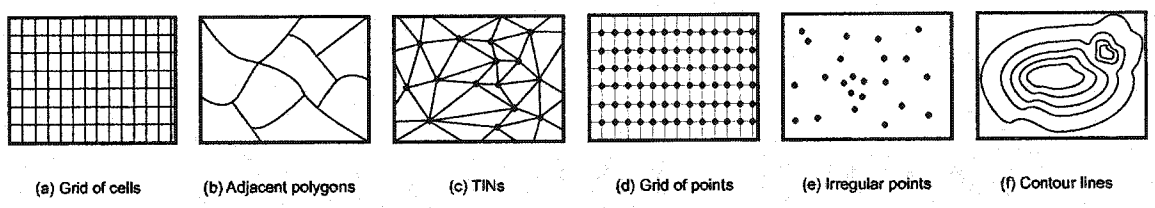


Figure 4.1: The different types of map fields

For example, a field called *temperature A*, may be defined as a function from locations in a grid of regular square cells to values ranging from -60 to +60 degrees Celsius. A different field *temperature B* may be defined as a function from locations on contour lines to the same values. Each contour line assumes only one value in the range set.

Fields have an important property which is a consequence of the fact that they are defined as a function: each location of the region has only one value.

Grid of Cells, adjacent polygons, and TINs have an important property called planar enforcement [Goo92]:

- a region described by a field is partitioned in non-overlapping cells
- every location must be within exactly one cell, or on a boundary.

There are a few specialized notations that are appropriate to describe map fields. OMT-G [BLD99, BDL01] is a model and a notation for describing the conceptual model of a geographic database. It is based on the Object Modeling Technique (OMT) [RBP+91] and extended for geographic applications. GeoFrame is "a conceptual framework that serves as a starting point for the conceptual modeling of geographic databases" [FI99]. It is based on the Unified Modeling Language (UML) [BRJ98]. Both are graphical notations and include special types of object classes to express the different types of fields presented earlier.

4.1.2 Describing map objects

If the phenomena in the real world are perceived as distinguishable objects, then the map should describe these objects using appropriate map phenomena and terminology.

The real world domain provides a description of each class of objects. This description is concerned with the recognition of an object as belonging to a class and with the geo-reference of each object. For example, if the objects of interest in the real world are the buildings with more than 5 floors, a rule to recognize these buildings should be provided in the description. The location of each of these buildings in the region of interest should also be provided. These are the necessary phenomena that the real world description needs to contain, but depending on the problem being analyzed, other phenomena may also be required. Section 4.2 considers the description of the real world in more detail.

The map-objects description also needs to make a distinction between the different classes of objects. Their location is also crucial for the map contents. However, there is another important phenomena that should be described about the map objects: their geometry. Despite the complex

entity class	geometry	location
fire station	point	(x, y)
emergency transmitter	point	(x, y)
road segment	line	$(x_1, y_1), (x_2, y_2)$
fire station coverage region	polygon	sequence of coordinates

Table 4.1: Entity classes for Algonquin Park map following the isolated model

shapes and geometries that the objects in the real world have, in the map the details of their geometries are described by one of the basic geometric primitives which consist of the point, line, and polygon. For the purpose of problem analysis and description, these basic geometric primitives are sufficient. If desirable, more complex geometries may be built upon the basic primitives.

Each of the geometric primitives assumes that the location of the object is provided through a number of coordinates. A *point* representation assumes that *one* coordinate is provided for the object. A *line* primitive assumes that a *pair* of coordinates is given for the endpoints of each line object. Finally, a polygon primitive assumes that a *sequence* of coordinates is given for each object. It also assumes that all polygons are closed and that the last and first coordinates in the sequence are connected with a line.

If the map objects are described by the three phenomena — class of the object, location, and geometry — then the map is said to follow the isolated object model (see Appendix B.6.2). No relationship between the objects is required in the description.

For example, in the Algonquin Park problem presented in Section 3.3 a map description appropriate to solve one subproblem may follow the isolated objects model. Table 4.1 shows what the entity classes look like.

This map description may be enough for one subproblem but may be too simplistic or may not talk about the necessary phenomena when considering another subproblem. In the connected model, objects need to be described as distinguishable entities as in the isolated model, but their relationships with other entities are also required.

There are important geographic relationships between entities that may need to be described in the map in order to solve a particular problem. There is a difference between what is given as input to a problem and what is required to solve the problem. Inputs to most geographic problems are usually the coordinates of objects in the world. However, each problem may require different types of relationships among the objects. These relationships can usually be computed from the

coordinates.

- **topological whole-part relations** are a special type of geographic relations where each part has a non-empty intersection with the whole entity. These relations are further divided into: [KPS97]
  - *covering* (figure 4.2a): in a covering relation the geometry of the whole is covered by the union of the geometries of the parts. The parts may overlap and may exceed the boundary of the whole. The whole and the parts must belong to the same geometric type.
  - *containment* (figure 4.2b): in a containment relation the geometry of the whole contains the geometry of all its parts. The parts may overlap but must not exceed the boundary of the whole. There may be regions of the whole that are not covered by any of the parts.
  - *partition* (figure 4.2c): in a partition relation the geometry of the parts form a partition of the geometry of the whole. In this case the parts must not overlap and there is no region of the whole that is not covered by a part. The whole and the parts must belong to the same geometric type.

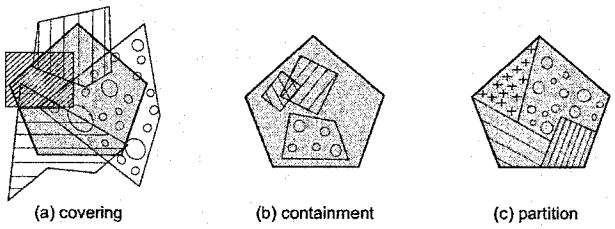


Figure 4.2: Topological whole-part relations

- **topological network relations** are concerned with two distinct classes of objects called nodes and edges. A network or graph is a collection of nodes and edges where objects of the node class are connected to each other through objects of the edge class; in other words, each edge is incident to two nodes.
- **other topological relations** may describe how two or more entities may be related to each other geographically. For example, given any two polygons, X and Y (closed and with no holes), the following relations are defined:

- X is *disjoint* from Y if the intersection between their boundaries is empty and the intersection between their interiors is also empty.
- X *is inside* Y (Y *contains* X) if X is a subset of Y and X, Y do not share a common portion of their boundaries.
- X *equals* Y if their boundaries are the same.
- X *meets* Y if X and Y touch externally in a common portion of their boundaries and their interiors do not intersect.
- X *covers* Y (Y is *covered by* X) if Y is a subset of X.
- X *overlaps* Y if the intersection between the interior of X and the interior of Y is non-empty.

The definition of these topological relations depends on the dimensionality (0,1 or 2) of the objects involved in the relations. However, they may all be defined in terms of the intersection between the objects' boundaries, interiors and exteriors [EF91, CFvO93]. Any other topological relation may be defined, as long as its semantics are provided.

- **metric relations** are concerned with the distance and direction between two entities.

  Given a set of points S, a function which takes ordered pairs (x, y) and returns a real number $d(x, y)$ is called a metric or distance on S if it satisfies the following axioms [Wor95]:

  1. $d(x, y) > 0$ if $x \neq y$ (non-negativity) and $d(x, x) = 0$; (reflexivity)
     the distance must be a positive number unless the points are the same, in which case the distance will be zero.
  2. $d(x, y) = d(y, x)$; (symmetry)
     the distance between two points is independent of which way around it is measured.
  3. $d(x, y) \leq d(x, z) + d(z, y)$; (triangle inequality)
     it must always be at least as far to travel between two points via a third point rather than to travel directly.

  For example, the Euclidean distance is a metric relation because it satisfies these axioms.

Note that all the relations presented here can be calculated and built from simple relation rules and the location (coordinates) of the objects. However, it is important to describe how the classes of objects in a map are related to each other because this is the core of problem analysis.

There are many geographic notations specialized to describe map objects, their geometries and locations, and the relationships among them. Most of the contributions in this area come from the database community. Some notations are specializations of the Entity-Relationship model, others rely on some object-oriented modeling language. Depending on the type of phenomena that are required in the map description, some notations may be more appropriate than others. In Section 1.6, a list of specialized notations is presented.

GeoOOA [KPS96] is used here to give an example of a map description for the Algonquin Park problem following the connected object model. GeoOOA extends the Object-Oriented Analysis method from Coad and Yourdon [CY91] by introducing geographic primitives. It is based on the notion that a geographic description should distinguish between spatial and conventional classes, and between different kinds of geometric primitives (points, lines and regions). Figure 4.3 was extracted from [KPS96] and shows the different class symbols that were incorporated in GeoOOA.

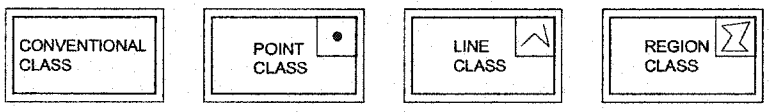


Figure 4.3: Class symbols in GeoOOA

GeoOOA allows the description of topological whole-part relations using the graphical notation depicted in Figure 4.4:

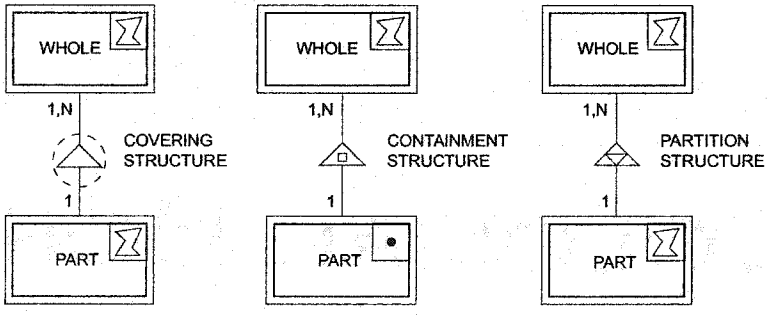


Figure 4.4: Topological whole-part structures in GeoOOA

Figure 4.5 shows the description of the Algonquin Park map using GeoOOA's graphical notation. Note that the purpose here is to illustrate map descriptions, not to be complete. When describing each problem frame, relevant map descriptions will be provided.

The Algonquin Park is partitioned into many fire station coverage regions. Each region contains one fire station and one or more emergency transmitters. Bodies of water are related to the

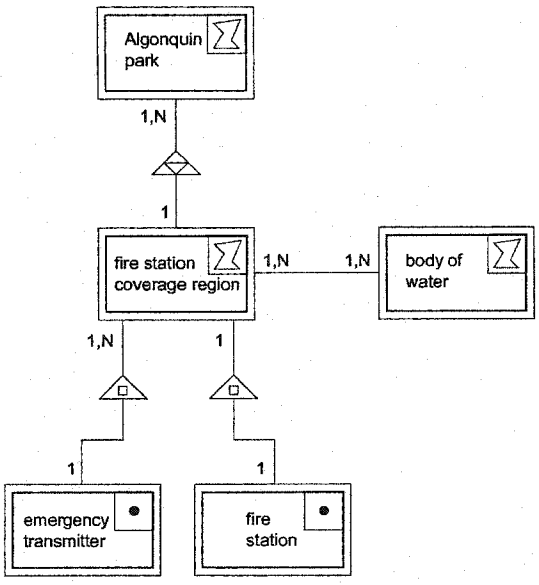


Figure 4.5: Algonquin Park map description using GeoOOA

regions. However, from this description, the topological relationship between these two entities is not clear. Do they cover, meet, or overlap each other? GeoOOA does not support this type of topological relation as described earlier in this section. If they are relevant for the problem at hand, another notation may be more appropriate. Only the expression of topological whole-part and network relations is possible in GeoOOA.

This description does not provide the attributes of each class. However, it is assumed that each geographic class provides appropriate attributes to describe the location of each object.

4.2 Describing the real world domain

The real world is an informal domain rich in phenomena. It is hard to describe the real world using a notation that assumes certain rules because in reality there are exceptions to most rules. Therefore, a description of the real world for geographic problems uses *designations* [Jac98]. The following are roles of a designation of geographic phenomena.

1. Pinpoint the phenomena of the world that are necessary to describe a particular problem. For example, according to table 4.2, the following phenomena can be identified in the RW description: $FireStation$, $Transmitter$, $RoadSegment$, $TrailSegment$, $BodyOfWater$, $CoverageRegion$.

2. Provide a clear recognition mechanism that allows one to decide if a phenomenon belongs to a certain class in the description or not. Table 4.2 presents a designation that allows, for example, the recognition of whether or not a particular path is a road or a trail segment.

3. Describe the geographic *location* of the phenomenon so that it can be uniquely identified. For example, in table 4.2, l is a location within the limits of Algonquin Park that can be unambiguously identified.

Unlike the map, which uses a coordinate system to refer to locations, in the real world there are many different ways of uniquely identifying a location. For example, in an urban setting, the address of a house is sufficient to determine its location without resorting to a coordinate system. In some cases, however, there is no way of designating the location of phenomena without using a coordinate system. Designations for such phenomena will be very similar to their corresponding map-domain descriptions. For example, a province boundary may fall into either of two cases. It may be possible to describe its location using natural phenomena (e.g., the St. Lawrence river to the south) or it may be necessary to specify a set of coordinates where no natural phenomena determine the boundary.

In the Algonquin Park problem, the relevant phenomena are emergency transmitters, fire stations, roads and fire station coverage regions. Table 4.2 shows an appropriate set of designations for these phenomena.

entity class	**designation**
$Location(l)$	l is a place within the limits of Algonquin Park that can be unambiguously identified
$FireStation(f, l)$	f is a building equipped with fire fighting gear at $Location(l)$
$Transmitter(t, l)$	t is a fixed emergency radio transmitter at $Location(l)$ that can be used to inform the central station in case of a fire
$RoadSegment(r, a, b)$	r is a continuous stretch of road connecting $Location(a)$ to $Location(b)$ that is accessible by automotive vehicles
$TrailSegment(t, a, b)$	r is a continuous trail connecting $Location(a)$ to $Location(b)$ that is not accessible by automotive vehicles
$BodyOfWater(w, l)$	w is a lake or river at $Location(l)$
$CoverageRegion(f, l)$	$FireStation(f)$ is responsible for dealing with all fires that happen in $Location(l)$

Table 4.2: Designations for the real world description of Algonquin Park

Generally, topological information does not appear in a real world description. However, there may be cases when this information is necessary. For instance, when the method used for data capture involves entering topological information directly rather than through computation. In this case, the data collector must obtain the topological relationships from the phenomena shared with the real world (see data capture problem frame in Section 5.1).

If the relevant phenomena in the real world are better described as fields instead of objects, the description should contain definitions that specify the function that relates locations to field values. These are different from the functions specified in the map domain in that they are continuous functions, whereas the map-domain fields are discrete approximations of the real-world fields.

Finally, the real-world domain should contain information in addition to the descriptions of phenomena, such as the region of interest to the problem. In the case of the Algonquin Park problem, the real-world-domain description should mention that the region of interest is that located within the boundaries of Algonquin Park.

Chapter 5

Informal problem description and analysis — creating and manipulating maps

In this chapter, the problem frames for map creation and manipulation are described. These problem frames assist in informal problem analysis and description. The common problems in the geographic area are discussed and outlined in Chapter 3. For each of those classes of problems, a problem frame is presented as a way to capture the parts involved in the problem. Each problem frame is also accompanied by an example representing that class of problems. The examples are used to explore the problem-oriented approach.

Fitting problems into frames helps to identify and describe customer needs, domain properties, and machine specifications in a more systematic way than using only natural language. The frame parts and the phenomena they share serve as prompts to remind the analyst to explore, to discuss, and to describe the problems carefully. They also provide hints and indicate patterns of similar problems in the class.

The focus of this chapter is on one of the three main categories of problems presented in Chapter 3: problems concerning the creation and manipulation of a map. The problems that make use of a map comprise the second category, which is presented in Chapter 6. Problems in the third category mostly require human labour and creativity. Since a machine cannot be built to solve these problems, they will not be further explored in this dissertation.

Some of the problem frames discussed in this chapter are based on Jackson's general purpose problem frames and are tailored here for the geographic area. Chapter 2 contains an overview of

problem frames with definitions of the terminology used in the approach. There are five different classes of problems associated with creating and manipulating maps: building a map that is a reasonable approximation of the real world — data capture; displaying a map on a computer screen — visualization; editing a map to add/remove entities; selecting points on a map displayed on a screen using an input device — selection; and converting maps to different formats to allow their integration — data conversion. In this chapter, each of these classes is explored as a problem frame.

5.1 Data-capture problem frame

Problem description

The problem is to build a model (map) that corresponds to the real world. Every phenomenon present in the map must have a real world equivalent. In this respect, the map is constrained by the real world. The problem frame described here covers two general methods of data acquisition, namely surveying and remote sensing. Digitizing paper maps, which is the third data acquisition method, is a special case and is presented at the end of this section.

The data-capture problem frame is based on the *model building* problem frame [Jac00b] with the addition of a connection domain, and further tailored to the geographic area. The data-capture frame concerns the acquisition of geographic data and the attachment of required non-geographic data to the spatial elements.

A sample problem from this class

The Algonquin Park problem description contains a subproblem that can be described as an instance of a data-capture frame. The description of this subproblem says:

A digital map of the park needs to be created and should contain [...] the location of the fire stations, the emergency transmitters, the roads, main trails and bodies of water. The park region must be surveyed by a team using GPSs or any other surveying devices. The locations should be recorded using UTM coordinates.

Before developing the description of this sample problem, the general frame diagram for the class of data capture problems will be presented in detail.

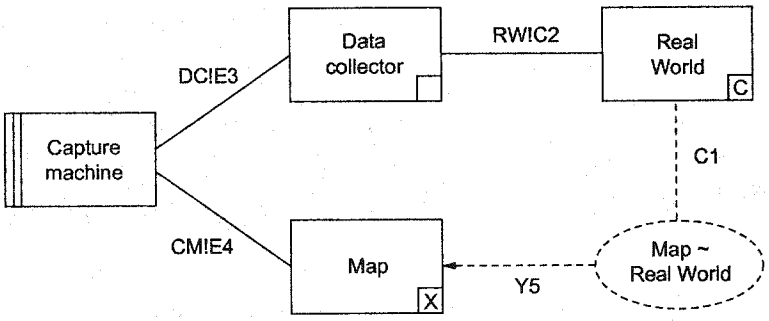


Figure 5.1: The data capture frame diagram

Frame diagram, domains, and shared phenomena

Figure 5.1 presents the data-capture frame diagram. The capture machine is the machine to be built. The real world is the domain to be modeled as a map. The requirement constrains the map being created to correspond to the real world.

The world is a causal domain, since it is responsible for the changes in its own state. However, for the scope of creating one map, or performing one data capture task, the real world is considered static. This means that no changes in the real-world states are perceived by the data collector. The real world phenomena may be described as in Section 4.2 with the main emphasis on the location, classification, and recognition rules of the geographic entities. Topological relations may or may not be part of the real world phenomena depending on the data-capture method and purpose of the creation of the map. The map is a lexical domain, given that it basically contains data describing the real world.

The world shares the states of its entities with a data collector. The data collector is a connection domain between the real world and the machine, since the phenomena in the world cannot be experienced directly by the machine. The data collector is purposefully left without a domain mark indicating the type of domain, since depending on the problem it can be a biddable (a person surveying) or a causal (a satellite) domain.

The data collector is responsible for understanding the phenomena C2 caused by the real world and generating events E3 that are shared with the machine. The machine is responsible for understanding and handling the events E3 and generating corresponding events E4 which change the state of the map domain. Thus, the customer can check if the state C1 is represented accordingly in the map as state Y5.

Note that C1 and C2 may be different types of phenomena. C1 is the part of the real world

that is required to be represented in the map. C2 is the part of the world that the data collector is capable of sensing. If the data collector is not capable of measuring and collecting the data with the required precision, the customer will be able to see that the map does not satisfy the requirements since the phenomena C1 in the real world will not correspond to phenomena Y5 in the map.

If the data collector is a biddable domain, it may be, for example, a human being using a GPS to record the position of entities in the world, or it may be a human being taking aerial photographs. A biddable data collector is an autonomous and spontaneous domain that is not controlled by another domain; it is unpredictable. When using this frame to describe data-capture problems with biddable data collectors, the procedure to be followed by the data collector should also be included. For example, in the Algonquin Park data capture, the following procedure could be included to instruct the data collector on how to capture the location of a lake: *start at any point on the lakeshore and walk around the lake until the starting point is reached, measuring and recording the coordinates every 20 meters.*

The data collector may be a causal domain, such as, for example, a cell phone base station or a scanner in a satellite. It has predictable behaviour and its description should contain a state transition diagram showing this behaviour. It may be an active domain that causes events or changes in its own state without any external stimulus. For example, a scanner in a satellite changes its own position without any external stimulus. It may also cause events, such as sharing the data that was collected with the machine without any external stimulus.

Changes in state of the real world may cause a reaction in a data collector. For example, when a cell phone enters the area covered by a base station, the base station reacts by generating an event to inform the machine that the particular cell phone is now under its coverage. In this case, the data collector domain is reactive and passive, since it only initiates events in response to state changes in the real world. This type of data collector is not further explored in this dissertation since the real world domain is considered to be static — without any changes in state — for the purpose of creating a map.

Whether the data collector is biddable or causal, its description should contain the correspondence between the phenomena C2 and the events E3. The role of the data collector is to connect the real world to the machine, so it is essential to show how phenomena from the real world are translated to the data collector events that are also experienced by the machine.

The relationship between phenomena C1 and C2 depends on properties of the real world. For example, if the customer requires the map to show a distinction between regions with vegetation and regions with water (phenomena C1) and the data collector is capable of registering regions of

blue or green colors (phenomena C2), the different colors need to be matched to the desired real world phenomena. The colors are not independent phenomena from the occurrence of water and vegetation — water is blue and vegetation is green. Therefore, blue is related to watery regions and green to vegetation regions.

Requirements

The requirements should contain a description of the criteria used to constrain the map. The description should show what phenomena C1 from the real world correspond to the symbolic phenomena Y5 in the map. There are different aspects of this correspondence that the customer may be interested in:

correctness of the map

Does every entity on the map have a corresponding real world entity? A map of the Algonquin Park, for example, should not show emergency transmitters, fire stations, or roads that do not exist in the world. The requirement description should make clear which real world phenomenon C1 corresponds to each entity Y5.

Correctness of the map in relation to the real world also concerns the location of the map entities. The coordinates of the map entities must be inside the region of interest in the real world. The map description should contain the coordinates of this region of interest (see Section 4.1).

completeness of the map

Does every entity of concern in the real world have a corresponding entity on the map? A map of the Algonquin Park, for example, should not leave out any of the emergency transmitters, fire stations, or roads that appear in the world. The relevant phenomena of the world that need to be captured are described by C1. The real world domain should describe the limits of the geographic region of interest so that all the required C1 entities in the real world can be checked for a corresponding Y5 map entity. The real world region of interest must be contained inside the map region of interest.

consistency of the map

Do all entities in the map conform to certain rules that are true in the real world? For example, in the real world two transmitters do not occupy the same location; this rule can be used to check the consistency of the map. Are all the transmitters within radio broadcast range of the central office? This is another way to check the correspondence between the

model and the real world, in this case checking to see if the properties that hold in the world also hold on the map.

If the topology is captured together with the data, topological consistency rules can also be checked. For example, in a subway network map, the stations and the connecting rails may be captured as nodes and edges of a graph. A topological consistency rule may be that every rail must connect two stations.

Specification

The specification should contain a description of the relationship between E3 and E4 events. The machine is responsible for translating each event E3 from the data collector into one or more corresponding E4 events that will modify the state of the map accordingly.

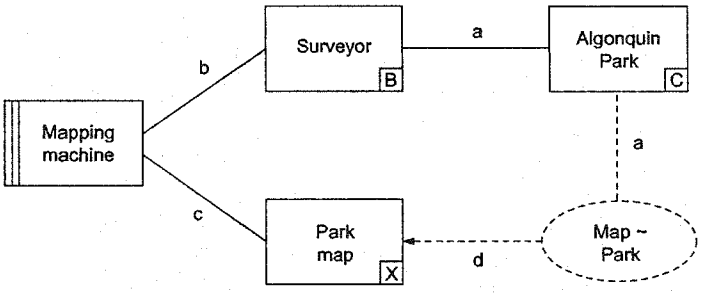
The specification of the machine will depend on the characteristics of the data collector. If the data collector is a biddable domain, the machine has to discard events which are not sensible or viable that may be generated by the data collector. It also has to validate the events generated by the data collector. This can be done by using the consistency rules to check the map being created in correspondence to the real world.

If the data collector is a causal domain, the machine in the data-capture frame may consider it a reliable domain. The events controlled by the data collector are caused in an ordered fashion as described by the data-collector domain properties. The machine experiences these events and generates its own events in turn, to change the state of the map. If the machine does not experience the events as expected according to the data collector properties, a failure in the data collector domain may have occurred. Monitoring the causal data collector may be quite difficult and may need to be a problem frame by itself. The concern here is only on data capture.

Framed sample problem

The sample problem from the beginning of this section will now be presented fitted to the data capture problem frame. This is the problem statement:

A digital map of the park needs to be created and should contain [...] the location of the fire stations, the emergency transmitters, the roads, main trails and bodies of water. The park region must be surveyed by a team using GPSs or any other surveying devices. The locations should be recorded using UTM coordinates.



a: AP! { FireStation(f, l), Transmitter(t, l), RoadSegment(r, a, b), TrailSegment(t, a, b), BodyOfWater(w, l) } [C2]

b: SV! { AddFireStation(f, x, y), AddTransmitter(f, x, y), AddRoadSegment(r, x_1, y_1, x_2, y_2), AddTrailSegment(t, x_1, y_1, x_2, y_2), AddBodyOfWater(w, Seq(x_1, y_1, ..., x_n, y_n)) } [E3]

c: MM! { AddPoint(p, e, x, y), AddLine(l, e, x_1, y_1, x_2, y_2), AddRegion(r, e, Seq(x_1, y_1, ..., x_n, y_n)) } [E4]

d: PM! { Point(p, e, x, y), Line(l, e, x_1, y_1, x_2, y_2), Region(r, e, Seq(x_1, y_1, ..., x_n, y_n)) } [Y5]

Figure 5.2: The Algonquin Park map creation fitted to the data capture frame

Figure 5.2 shows how the parts of the problem of creating a map of the Algonquin Park fit into the data capture frame. In this case, the surveyor is the biddable data collector. The entity classes described in Section 4.2 characterize the real world phenomena of interest — the park domain — that need to be captured, as well their location. The mapping machine is responsible for building the model — the map domain — of these phenomena. The park domain shares causal phenomena with the surveyor, indicating the location of the entities of interest. The surveyor should follow the procedure outlined in Table 5.1 to capture the location of each entity and generate an event to the machine to add each of the entities to the map.

The machine responds to each event from the surveyor by creating appropriate events to add the entities to the map as points, lines, and polygons, which are map phenomena. Table 5.2 shows the machine specification as a list of events generated by the machine in response to the events received from the surveyor. Note that all (x, y) coordinates should be expressed in the UTM coordinate system.

This example includes all three kinds of requirements: correctness, completeness, and consistency. For each phenomenon, each of the three types of requirements should be detailed. For example, the three types of requirements involving the phenomenon *transmitters* are:

entity class	**capture procedure and corresponding event**
$FireStation(f, l)$	a single coordinate measured at any point inside the fire station building
	$AddFireStation(f, x, y)$
$Transmitter(t, l)$	a single coordinate measured at the transmitter's exact position
	$AddTransmitter(t, x, y)$
$RoadSegment(r, a, b)$	a pair of points with coordinates measured at each end of the road segment
	$AddRoadSegment(r, x_1, y_1, x_2, y_2)$
$TrailSegment(t, a, b)$	a pair of points with coordinates measured at each end of the trail segment
	$AddTrailSegment(t, x_1, y_1, x_2, y_2)$
$BodyOfWater(w, l)$	a sequence of coordinates obtained by starting at any point on the shore and walking around the body of water until the starting point is reached, measuring and recording the coordinates at every 20 meters
	$AddBodyOfWater(w, Seq(x_1, y_1, \ldots, x_n, y_n))$

Table 5.1: Data collector procedure outline for the Algonquin Park example

corresponding events	
generated by the surveyor	**generated by the machine**
$AddFireStation(f, x, y)$	$AddPoint(p, f, x, y)$
$AddTransmitter(t, x, y)$	$AddPoint(p, t, x, y)$
$AddRoadSegment(r, x_1, y_1, x_2, y_2)$	$AddLine(l, r, x_1, y_1, x_2, y_2)$
$AddTrailSegment(t, x_1, y_1, x_2, y_2)$	$AddLine(l, t, x_1, y_1, x_2, y_2)$
$AddBodyOfWater(w, Seq(x_1, y_1, \ldots, x_n, y_n))$	$AddRegion(r, w, Seq(x_1, y_1, \ldots, x_n, y_n))$

Table 5.2: Specification for the Algonquin Park example

- **Correctness**: for every transmitter (an entity represented by a point) found on the map, there is a transmitter in Algonquin Park at the corresponding location
- **Completeness**: every transmitter in Algonquin Park must be represented on the map by a point entity at the corresponding location
- **Consistency**: two transmitters must not occupy the same location on the map

Table 5.3 shows the correspondence between real world and map phenomena as well as the correspondence of the geo-references. This correspondence is necessary when evaluating whether or not the requirements are satisfied.

real world	corresponding map phenomena and geo-reference
$FireStation(f, l)$	$Point(p, f, x, y)$
	(x, y) is the coordinate of a point inside $Location(l)$
$Transmitter(t, l)$	$Point(p, t, x, y)$
	(x, y) is the exact $Location(l)$
$RoadSegment(r, a, b)$	$Line(l, r, x_1, y_1, x_2, y_2)$
	(x_1, y_1) is the coordinate of a point within $Location(a)$ and (x_2, y_2) is the coordinate of a point within $Location(b)$
$TrailSegment(t, a, b)$	$Line(l, t, x_1, y_1, x_2, y_2)$
	(x_1, y_1) is the coordinate of a point within $Location(a)$ and (x_2, y_2) is the coordinate of a point within $Location(b)$
$BodyOfWater(w, l)$	$Region(r, w, Seq(x_1, y_1, \ldots, x_n, y_n))$
	$Seq(x_1, y_1, \ldots, x_n, y_n)$ are a sequence of coordinates following the perimeter of $Location(l)$

Table 5.3: Correspondence between real world and map phenomena for the Algonquin Park

Variation — Digitizing paper maps

The method of data capture through digitizing paper maps does not rely on the real world in order to create a digital map. Instead, it relies on paper maps which are already an abstract representation of the real world.

The parts involved in this problem are the same as in the general data-capture problem frame, with the exception of the real world. Instead of the causal-real world domain, a lexical paper-map domain becomes the domain to be modeled. The data collector can also be biddable, as in the case of a person capturing data manually with a digitizer device, or causal, when it is done automatically by a scanner connected to the machine. The requirements in this situation should be checked against the paper map and not the real world. A correspondence with the real world is not established in this case. Figure 5.3 presents the problem-frame diagram for data capture when the chosen method is digitizing paper maps.

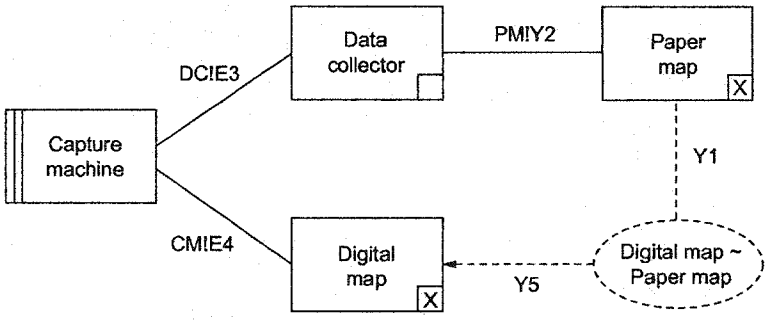


Figure 5.3: The data capture frame diagram for digitizing paper maps

5.2 Data-conversion problem frame

Problem description

The problem is to convert a map from one digital format to another. The source and the target phenomena may differ in terms of datum, map projection, coordinate system, data model, data encoding, or level of detail. The problem frame described here covers any conversion that can be completely automated since this is the most common case. However, changing the level of detail of a map may require map generalization, which is a special case of conversion since it is an interactive process with a user. This variation will be presented at the end of this section.

The data conversion problem frame is based on the *transformation* problem frame [Jac00b, Kov98] but tailored to the geographic area.

A sample problem from this class

The Algonquin Park problem description contains a subproblem that is an instance of the data-conversion class. The description of this subproblem says:

A digital map of the park needs to be created and should contain the boundaries of the park, ... The park boundaries can be obtained from the Survey and Mapping Branch of the Ontario Ministry of Natural Resources in raster digital format.

Since the system that needs to be built requires a map where topological relations between its entities can be explored (for example, transmitters are inside the park boundary, coverage regions form a partition of the park), the map should only contain objects in the vector format to allow for building its topology. The raster map containing the park boundary should be converted to a vector map. Figure 5.4 illustrates the map in raster and vector format.

(a) Raster format (b) Vector format

Figure 5.4: Converting the park boundary from raster to vector format

Before developing the description of this sample problem, the general frame diagram for the class of data-conversion problems will be presented in detail.

Frame diagram, domains, and shared phenomena

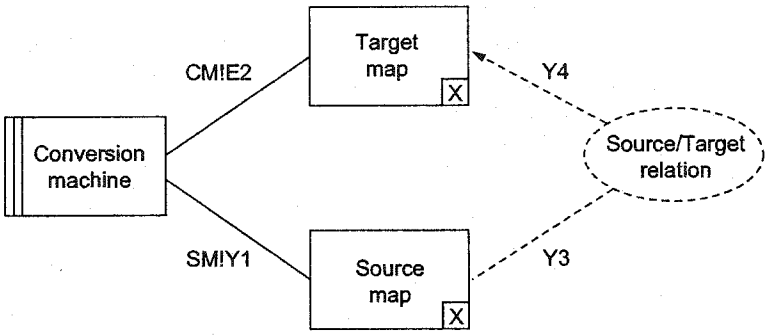


Figure 5.5: The data-conversion frame diagram

Figure 5.5 presents the data-conversion frame diagram. In this problem frame, the conversion machine is what needs to be built. The source map is the input map to be converted or transformed into the target map, which is the output map. The requirement constrains the target map, which is being created to maintain a certain relation with the source map. This relation is described as part of the requirement.

Both the source and target map domains are data and are, therefore, lexical domains. In this case, both are stored in digital format. However, their phenomena differ significantly. For example, in the sample problem of converting the boundaries of Algonquin Park, the source map is in raster format and the target map is in vector format. The phenomena that should describe the source map are pixels while the target map should be described in terms of points, lines and polygons.

The source map shares its states Y1 with the machine, which is responsible for generating events E2 to create the appropriate state in the target map. Thus, the customer can check whether state Y4 in the target map maintains the required relation with state Y3 in the source map.

Y1 and Y3 may be the same set of phenomena. However, they are usually different. Y1 represents the source map phenomena as perceived and understood by the machine, and are thus, machine-oriented types of phenomena. Y3 represents the source map phenomena as perceived by the customer at a higher level of abstraction. For example, in a raster source map the phenomena Y1 may be described as pixels that assume black or white values, and Y3 may be perceived as the boundary buffer: the set of all coordinates that are close to the actual boundary.

Requirements

The requirement of the data-conversion problem frame has to describe the relation between the source and the target map. It has basically to provide the relation between Y3 and Y4 phenomena.

In the example of converting the park boundaries from raster to vector format, the Y4 phenomenon is a *boundary of the park* entity which is represented by a polygon in the target map state. As mentioned above, Y3 is the boundary buffer defined as the set of coordinates close to the actual boundary between the black and white regions. The requirement should describe how the park boundary on the vector map is related to the boundary buffer phenomenon. The full description is presented later in this section.

Specification

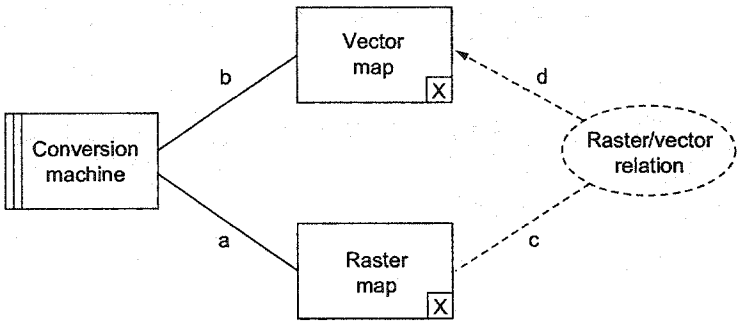
The specification of the conversion machine has to describe the relation between the Y1 symbolic phenomena and the events E2 generated by the machine. The E2 events should produce the appropriate target map that can be perceived by the customer as Y4 phenomena. The specification should indicate how the source map should be processed or traversed in order to trigger the generation of the E2 events in the appropriate order to culminate in a target map that satisfies the requirements.

Framed sample problem

The sample problem from the beginning of this section is presented here fitted to the data-conversion problem frame. Informally the problem is stated as:

A digital map of the Algonquin Park needs to be created and among other entities, the map should contain the boundaries of the park. The park boundaries can be obtained from the Survey

and Mapping Branch of the Ontario Ministry of Natural Resources in raster digital format. However, the park map should only contain objects in the vector format. Therefore, the raster map containing the park boundary should be converted to a vector map.



a: RM! { Grid of pixels, Size, Resolution } [Y1]
b: CM! { AddPolygon(p, boundary, Seq(x, y)), AddVertexToPolygon(p, x, y) } [E2]
c: RM! { Boundary buffer, *d* } [Y3]
d: VM! { Park boundary entity } [Y4]

Figure 5.6: The Algonquin Park boundary creation fitted to the data-conversion frame

Figure 5.6 shows how the parts of the problem of converting a map with the Algonquin Park boundary from raster to vector format fit the data conversion frame. In this case, the source map is the raster boundary and the target map is the vector boundary. The machine is responsible for converting or transforming the raster boundary into the vector boundary. The domain descriptions, shared phenomena, requirements, and specification are further explored below.

The raster map is an instance of a grid of cells as presented in Section 4.1.1. Here, each cell is denominated a *pixel*. The raster map description is divided into two sets:

1. phenomena shared with the machine (Y1)
   - the raster map is formed by a grid of $w \times h$ pixels representing the region of interest; in this case, $w = h = 1000$ pixels
   - each pixel has size $p \times p$ meters (resolution); in this case, $p = 100m$
   - each pixel represents a set of coordinates
   - each pixel has a value: *black* or *white*

2. phenomena referred to by the requirement (Y3)

   - distance d defined as: d $\hat{=}$ $p/2 = 50m$
   - boundary buffer $\hat{=}$ set of coordinates x such that the value of each coordinate x is different from the value of some coordinate at a distance smaller than d from coordinate x.

The boundary buffer includes all coordinates that are close to the actual boundary. If the coordinate being considered has value *black*, and is close to (distance $\leq d$) another coordinate that has value *white*, then it belongs to the buffer.

The vector map description also has two groups of phenomena:

1. private phenomena

   - the vector map is formed by a set of points, lines, and polygons representing the region of interest
   - a polygon is a sequence of points (vertices) connected by line segments (edges). The last point in the sequence is connected by an edge to the first point.

2. phenomena constrained by the requirement (Y4)

   - park boundary entity $\hat{=}$ a single polygon on the map.

The requirements can be described as:

1. every coordinate in the park boundary entity on the vector map is also part of the boundary buffer defined in the raster map.
2. the distance between every coordinate in the boundary buffer and the vector boundary is smaller than d
3. there can be no other entity satisfying requirements 1 and 2 with fewer vertices than the park-boundary entity.

The following definition is necessary in order to describe the specification:
boundary pixel $\hat{=}$ a pixel with at least one neighbour of a different value
The following is the specification of the machine:

1. from all boundary pixels, select a black pixel with the fewest black neighbours and generate a *CreatePolygon* event to create a polygon with a single vertex on the vector map located at the central coordinate of the selected pixel

2. traverse through the sequence of black boundary pixels that are neighbours of the above selected pixel in any chosen direction

3. for every change in direction of traversal, generate an *AddVertex* event to add a vertex to the polygon on the vector map at the central coordinate of the pixel in the sequence where the direction of traversal changes.

This specification describes one possible traversal of the pixels that will, together with the vector and raster domain properties, satisfy the requirements described above. However, there may be other traversals that are more efficient and may also satisfy the requirements.

Variation — data conversion with user intervention

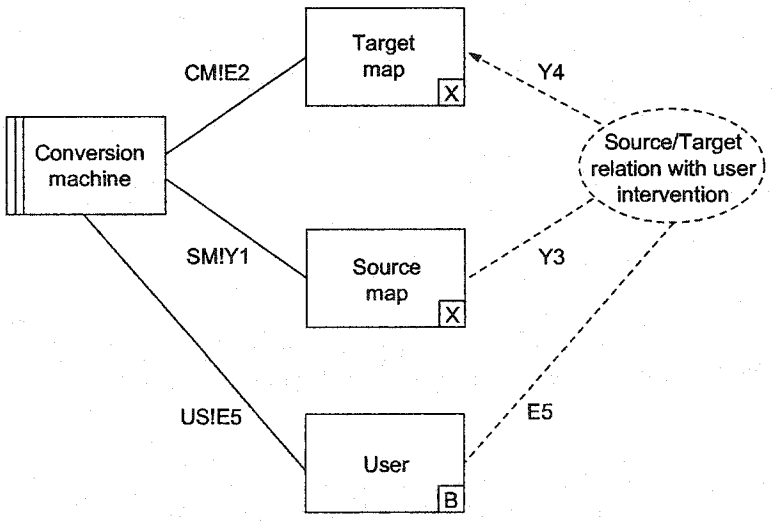


Figure 5.7: The data conversion with user intervention

Changing the level of detail of a map may require a generalization task, which in turn requires a user to judge which map objects are involved in each generalization step. For example, roads consisting of many segments may be simplified to a single segment that is an approximation of the path of the road. Other map objects may be left in their original form. Although it may be

possible to automate this task, usually there is a person responsible for guiding this process and deciding about the quality of the results.

Figure 5.7 presents the problem-frame diagram for this case. A user initiates a conversion step by sending an event E5 to the conversion machine that carries some form of identification of the entities in the source map that should be converted. The machine rejects the request if it is not viable (for example, if the desired entities don't belong to the source map). The machine then generates an event E2 that creates the converted entities in the target map. The requirement reflects the source/target relationship but takes into account any user interventions.

5.3 Editing-the-map problem frame

Problem description

The problem is to modify the map following the instructions of a user. These modifications include adding, removing, or modifying map phenomena (objects or fields). Unlike the data-capture frame, the map phenomena of an editing frame need not necessarily have real world correspondents. For example, political boundaries may need to be added to a map but may not have a physical manifestation in the real world.

The editing problem frame is similar to the *workpieces* problem frame [Jac00b, Kov98]. Editing a map is no different than editing any other digital workpiece. The difference lies in the description of the domains and shared phenomena. Therefore, the description of the editing frame will be brief.

A sample problem from this class

The following subproblem comes from the Algonquin Park problem description. It is an instance of the editing class:

The park is partitioned into many regions with independent fire stations. [...] The boundaries of the fire station coverage regions can be input after the rest of the data capture is done since these boundaries can be placed anywhere inside the park that is more convenient or appropriate for the fire station patrols.

Frame diagram, domains, and shared phenomena

Figure 5.8 presents the editing frame diagram. The *user* is a biddable domain responsible for generating the operation requests to add, to delete, or to modify map phenomena.

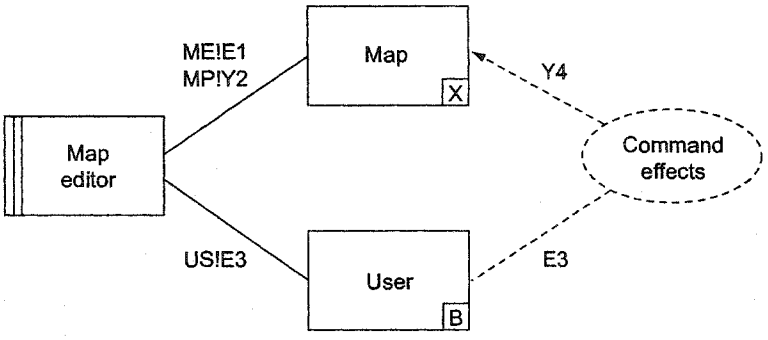


Figure 5.8: The editing frame diagram

The *map* is a lexical domain. Any kind of map may fit this problem - field or object, connected or isolated — as long as the machine is specified to deal with the appropriate phenomena.

The map shares its states and values Y2 with the machine. The user initiates events E3 that will be obeyed if they are recognized by the editing machine as requests to edit the map. The events identify the type of editing operation desired and the entities affected by the operation. The machine generates events E1 to modify the map given what it knows about the state of the map through Y2. The symbolic phenomena Y4 are the requirement references. They are the phenomena that the user would examine on the map to verify that the requested operations were performed correctly. Y2 and Y4 are different in that Y2 contains machine-oriented phenomena such as points, lines, and polygons, and Y4 contains user-oriented phenomena with added meaning such as houses, roads, and lakes.

Requirements

The requirement of the editing frame must describe the effects that the user's commands have on the map phenomena. It must state the relation between the user events E3 and the map symbolic phenomena Y4.

For example, when adding the fire station coverage regions, the requirement states that for each *AddCoverageRegion* event generated by the user that contains a region that lies within the map's region of interest, a fire station coverage region exists in the map.

The requirements must also describe which commands are sensible — that is, which events from the user make sense in the context of preceding events and therefore should be recognized and obeyed. The requirements must also describe which commands are viable — that is, which events are valid given the current state of the map. For example, if a user tries to add an object

outside the region of interest on the map, the command is not viable.

Specification

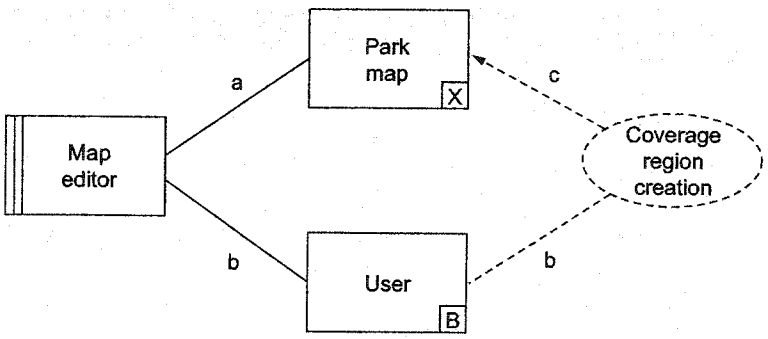
The specification of the map-editor machine needs to describe the events E1 that are generated in response to the events E2 received from the user.

When a user event is sensible and viable, the specification must state which events are generated to modify the map. Events that are not sensible or viable are ignored by the machine. This may sound overly simplistic. A solution to the problem may deal differently with commands that are not sensible or viable. For example, the machine may display a message for the user explaining the nature of the difficulty.

Framed sample problem

The sample problem from the beginning of this section can be described using the editing frame. The problem is stated as:

The park is partitioned into many regions with independent fire stations. [...] The boundaries of the fire station coverage regions can be input after the rest of the data capture is done since these boundaries can be placed anywhere inside the park that is more convenient or appropriate for the fire station patrols.



a: ME! { AddRegion(r, CoverageRegion, Seq(x_1, y_1, ..., x_n, y_n)) } [E1]
PM! { Point(p, e, x, y), RegionOfInterest } [Y2]

b: US! { AddCoverageRegion(r, Seq(x_1, y_1, ..., x_n, y_n)), Finished } [E3]

c: PM! { FireStation, CoverageRegion, RegionOfInterest } [Y4]

Figure 5.9: The Algonquin Park coverage region creation fitted to the editing frame

Figure 5.9 presents the problem of creating fire-station coverage-regions on the Algonquin Park map. Some requirements involve topological relationships and need a separate *data conversion* frame in order to be handled. For example, the requirements that the coverage regions form a partition of the park, and that there must be a fire station inside each coverage region are topological in nature and must be described in a separate subproblem. This subproblem is an instance of the data conversion problem frame since the conversion from the isolated object model to the connected object model is necessary to build the topology of the map. There are some requirements that do not involve topology and lie within the scope of the editing frame. For example, there must be the same number of coverage regions as fire stations, and the coverage regions must be in the region of interest of the map.

The park map shares with the machine the phenomena concerning the points where fire stations are located and the region of interest of the map. The user generates events to add coverage regions indicating the sequence of points at the boundary of each region. The machine validates the input region, checking whether it is inside the *RegionOfInterest* of the map. If it is not, the request is not viable and is rejected. Otherwise, the machine generates an *AddRegion* event to create the appropriate object on the map. When the user issues a *Finished* event, the machine checks whether the number of previously issued *AddRegion* events is the same as the number of fire stations in the map. If this is not true, the *Finished* event is not viable and is rejected. *AddRegion* events are also rejected if there are already as many coverage regions as fire stations. Commands from the user that are not sensible are rejected by the machine.

The requirement description states that the sensible commands are *AddCoverageRegion(r, Seq(x1, y1, ..., xn, yn))* and *Finished*. The *AddCoverageRegion* command takes two parameters: a unique identifier for the coverage region and a sequence of geographic coordinates delimiting the region. The *AddCoverageRegion* command is viable when the number of coverage regions in the map is smaller than the number of fire stations. The *Finished* command is viable when the number of coverage regions on the map is equal to the number of fire stations.

The requirements also state that the map should contain a coverage region in the appropriate location after a sensible and viable *AddCoverageRegion* user request. The number of regions added by the user must be equal to the number of fire stations in the map after the user issues a viable *Finished* event.

5.4 Visualization problem frame

Problem description

The problem is to display a digital map on a device such as a computer screen or printer, according to some request. The task involves transforming the digital map into a graphical map by establishing a display scale and a set of symbols for representing the digital map objects or fields.

The visualization problem frame is a composite problem consisting of two smaller subproblems. The first is similar to the *model building* frame [Jac00b] with an added *requestor* domain which is responsible for specifying what parts of the digital map should be displayed and/or how they should appear. This subproblem involves building the graphical map, a model of the digital map that includes information about how the map should be displayed, such as its scale, center, symbols, and so on. The second subproblem is similar to the *information display* frame [Jac00b] and involves rendering the graphical map model on a display medium such as a terminal or printer.

A sample problem from this class

There are two instances of visualization subproblems in the Algonquin Park problem. The first involves allowing a user to select the scale of the viewed map and the region to be viewed. The second involves a request from the central station to display an alert signal on the map. The problems are stated as follows:

Each station has a computer terminal with the park map, where the user can zoom in and out and select the viewed region. Fire notification is done through a blinking orange circle displayed on the station's terminal indicating the location of the alert signal on the park map.

Frame diagram, domains, and shared phenomena

Figure 5.10 presents the visualization frame diagram. The graphical-modeling machine is responsible for taking the information from the requestor and from the current state of the digital map, and building a graphical map model that corresponds to the digital map and contains information that is necessary for rendering the map according to the requestor's instructions. The *requestor* is a biddable or causal domain, depending on whether it is a user that is issuing the requests (biddable) or another computer system (causal).

The *digital map* domain may be described following the object or the field model. The domain represents the map, which is the source of information. The *graphical map* domain contains state phenomena that are related to how the map is to be displayed (for example, the color and

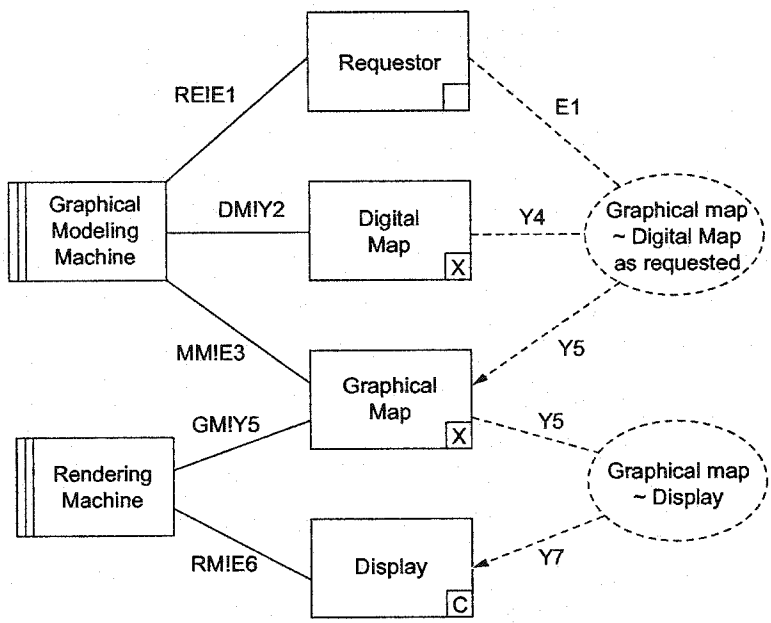


Figure 5.10: The visualization frame diagram

thickness of each line, the scale, the point in the map that should be at the center of the display). The graphical map domain is shared between the two subproblems and provides the connections between them. The role of the graphical modeling machine is to build the graphical map, and the role of the rendering machine is to display it. The *display* domain describes the causal phenomena that govern its behaviour (for example, *draw* events). The display domain is passive, and reacts to external events caused by the rendering machine by changing its own state.

Graphical-modeling machine

The requestor issues events E1 containing information about what should be displayed and about the appearance of the graphical map. The machine discards any E1 events that are not sensible or viable (see editing frame). The digital map shares symbolic phenomena Y2 about its state with the machine. The machine combines the request with the map state and generates events E3 which create graphical map entities corresponding to the digital map constrained by the request. These events may be, for example, *CreatePoint(symbol, color, location)*, *CreateLine(symbol, color, location)*, or *ChangeScale(scale)*.

The E1 events issued by the requestor and Y4 symbolic phenomena from the digital map are the requirement references that constrain the symbolic phenomena Y5 in the graphical map. The symbolic phenomena Y5 consist of points, lines, and polygons, along with ren-

dering information such as colors, styles, symbols, and scale. The graphical map does not include any information about spatial relations – it is only a graphical representation of the digital map for display purposes. Y2 and Y4 may be the same phenomena, but are usually different since Y2 refers to machine-oriented phenomena such as points, lines, and polygons, and Y4 to user-oriented phenomena.

Rendering machine

The rendering machine observes the state of the graphical map through phenomena Y5, and generates events E6 to the display that draw the corresponding map phenomena. The same Y5 phenomena are the requirement reference that constrain the Y7 phenomena that can be observed in the display. The events E6 are *draw* events that contain information about the appearance and position of the displayed elements. The phenomena Y7 are the graphical elements that can be observed in the display, such as a blue line.

The correspondence between the displayed elements (Y7) and the digital map (Y4) is transitive through the graphical map (Y5). That is, a blue line on the screen represents a river in the digital map if there is a Y5 phenomenon representing both at the corresponding location.

Requirements

Graphical-modeling machine

The requirement for the graphical modeling machine should list the graphical map phenomena Y5 that correspond to the digital map phenomena Y4 and requestor events E1. The requirement should describe all the symbols that represent the various digital map phenomena. For example, given the requestor event *View* and map phenomenon *FireStation(f, l)*, the corresponding graphical map phenomenon is *Cross(red, solid, x, y)*.

Rendering machine

The requirement for the rendering machine should list the display phenomena Y7 that correspond to the graphical map phenomena Y5. It must also establish the correspondence between graphical map coordinates and display coordinates. For example, given graphical map phenomena *Cross(red, solid, x, y)*, *Scale(s)*, *CenterX(cx)*, and *CenterY(cy)*, a red cross should be visible at screen coordinates *(sx, sy)*, where *sx* is a function of the values of *x*, *s*, and *cx*, and *sy* is a function of the values of *y*, *s*, and *cy*.

Specification

Graphical-modeling machine

The specification of the graphical modeling machine should describe the events E3 that are issued in response to events E1 from the requestor, given the phenomena Y2 from the digital map. For example, given the requestor event *View* and map phenomenon *Point(p, FireStation, x, y)*, a *CreateCross(red, solid, x, y)* event is issued.

Rendering machine

For the rendering machine, the specification should describe the events E6 that are issued given the Y5 graphical map phenomena. For example, given graphical map phenomena *Cross(red, solid, x, y), Scale(s), CenterX(cx),* and *CenterY(cy)*, the corresponding display phenomenon is *DrawPoint(cross, red, solid, sx, sy)*, where *sx* is calculated as a function of the values of *x, s,* and *cx* and *sy* is calculated as a function of the values of *y, s,* and *cy*.

Framed sample problem

The first sample visualization subproblem is to display an alert from the central station on the map:

Fire notification is done through a blinking orange circle displayed on the station's terminal indicating the location of the alert signal on the park map.

Figure 5.11 presents the subproblem of displaying the alert signal on the Algonquin Park map framed as a visualization problem. The *Graphical Modeling Machine* senses *FireAlert* events from the *Central Station*. The machine then examines the *Point* phenomena from the digital park map to locate the transmitter specified in the *FireAlert* event. If the transmitter is not located on the map, the request is not viable and is rejected. Otherwise, the machine issues a *CreateCircle* event to the *Graphical Map*. The requirement is that a *Circle* with orange color and blinking style should be part of the *Graphical Map* at the alert location reported by the *Central Station* if a transmitter exists at that location in the *Digital Park Map*.

The *Rendering Machine* observes any change of state in the *Graphical Map* through its shared phenomena. If a *Circle* with orange color and blinking style appears in the *Graphical Map*, the machine maps the geographic coordinates (x, y) to screen coordinates (sx, sy) that are dependent on the current scale of the map. It then issues a *DrawCircle* event to the *Fire Station Terminal* at the corresponding screen coordinates. The requirement is that if there is an orange blinking *Circle* on the *Graphical Map*, the *Fire Station Terminal* should display such a circle.

By transitivity, the implicit requirement of the visualization frame is that if there is an event *FireAlert(l)* from the *Central Station*, there must be an orange blinking *Circle* on the *Graphical*

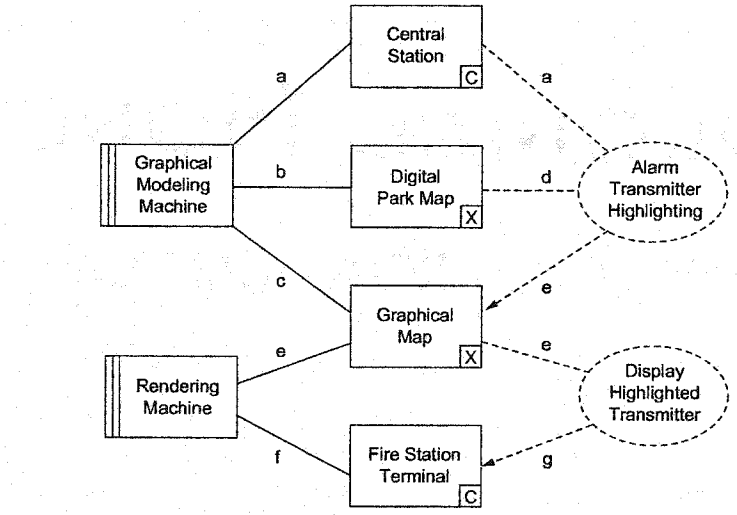


Figure 5.11: Fire notification framed as a visualization problem

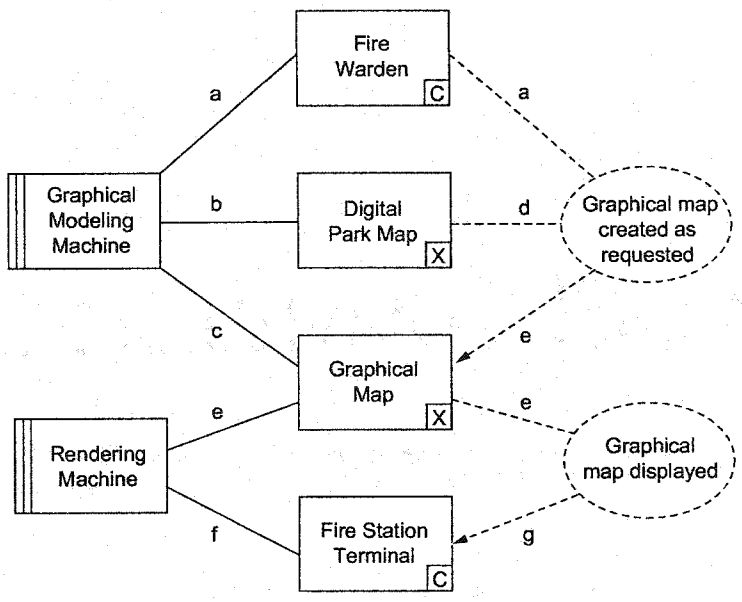
Map displayed in the *Fire Station Terminal* at the corresponding screen coordinates.

The second instance of the visualization subproblem is related to showing the map and allowing the patrols at the station to modify the graphical map by zooming and panning the image:

Each station has a computer terminal with the park map, where the user can zoom in and out and select the viewed region.

Figure 5.12 presents this subproblem. The machine senses *View*, *Zoom* and *Pan* events from the *Fire Warden*. In the case of *View* events, the graphical modeling machine must build the graphical map and the rendering machine must display it at the default scale which fits the complete map on the terminal. The requirements for this event are shown in Table 5.4 and the specifications of the machines for this event are shown in Tables 5.5 and 5.6. The specification of the machine contains a correspondence between the map entities shared with the machine and the symbols to be displayed for each entity.

In the case of *Zoom* and *Pan* events, if the events refer to locations that are not in the map, the machine rejects the events as not viable. Otherwise, the *Graphical Modeling Machine* issues *ChangeScale* events to the *Graphical Map* in response to *Zoom* events or *ChangeCenter* events in



a: CS! { View, ZoomIn(l), ZoomOut(l), Pan(l) } [E1]

b: DPM! { Point(p, e, x, y), Line(l, e, x_1, y_1, x_2, y_2), Region(r, e, Seq(x_1, y_1, ..., x_n, y_n)) } [Y2]

c: GMM! { CreateCross(color, style, x,y), CreateDot(color, style, x,y),
CreateLine(color, style, x_1, y_1, x_2, y_2),
CreatePolygon(color, style, Seq(x_1, y_1, ..., x_n, y_n)),
ChangeScale(s), ChangeCenter(x, y) } [E3]

d: PM! { FireStation(f, l), Transmitter(t, l), RoadSegment(r, a, b), TrailSegment(t, a, b),
BodyOfWater(w, l), CoverageRegion(c, l), ParkBoundary(l) } [Y4]

e: GM! { Cross(color, style, x,y), Dot(color, style, x,y), Line(color, style, x_1, y_1, x_2, y_2),
Polygon(color, style, Seq(x_1, y_1, ..., x_n, y_n)), Scale, CenterX, CenterY } [Y5]

f: RM! { DrawPoint(symbol, color, style, sx, sy), DrawLine(color, style, sx_1, sy_1, sx_2, sy_2),
DrawPolygon(color, style, Seq(sx_1, sy_1, ..., sx_n, sy_n)) } [E6]

g: FST! { Cross(sx, sy), Dot(sx, sy), GrayLine(sx_1, sy_1, sx_2, sy_2), BrownLine(sx_1, sy_1, sx_2, sy_2),
BluePolygon(Seq(sx_1, sy_1, ..., sx_n, sy_n)), WhitePolygon(Seq(sx_1, sy_1, ..., sx_n, sy_n)) } [Y7]

Figure 5.12: Zooming the Park Map framed as a Visualization problem

PM!Y4 phenomena	GM!Y5 phenomena	FST!Y7 phenomena
FireStation(f, l)	Cross(red, solid, x, y)	RedCross(sx, sy)
Transmitter(t, l)	Dot(white, solid, x, y)	WhiteDot(sx, sy)
RoadSegment(r, a, b)	Line(gray, solid, x_1, y_1, x_2, y_2)	GrayLine(sx_1, sy_1, sx_2, sy_2)
TrailSegment(r, a, b)	Line(brown, dashed, x_1, y_1, x_2, y_2)	BrownLine(sx_1, sy_1, sx_2, sy_2)
BodyOfWater(w, l)	Polygon(blue, filled, Seq(x_1, y_1, ..., x_n, y_n))	BluePolygon(Seq(x_1, y_1, ..., x_n, y_n))
CoverageRegion(c, l)	Polygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))	WhitePolygon(Seq(x_1, y_1, ..., x_n, y_n))
ParkBoundary(l)	Polygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))	WhitePolygon(Seq(x_1, y_1, ..., x_n, y_n))

Table 5.4: Requirements for the *View* event in the park visualization problem

DPM!Y2 phenomena	GMM!E3 events
Point(p, FireStation, x, y)	CreateCross(red, solid, x, y)
Point(p, Transmitter, x, y)	CreateDot(white, solid, x, y)
Line(l, RoadSegment, x_1, y_1, x_2, y_2)	CreateLine(gray, solid, x_1, y_1, x_2, y_2)
Line(l, TrailSegment, x_1, y_1, x_2, y_2)	CreateLine(brown, dashed, x_1, y_1, x_2, y_2)
Region(r, BodyOfWater, Seq(x_1, y_1, ..., x_n, y_n))	CreatePolygon(blue, filled, Seq(x_1, y_1, ..., x_n, y_n))
Region(r, CoverageRegion, Seq(x_1, y_1, ..., x_n, y_n))	CreatePolygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))
Region(r, ParkBoundary, Seq(x_1, y_1, ..., x_n, y_n))	CreatePolygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))

Table 5.5: Specification for the *View* event in the park graphical modeling machine

GM!Y5 phenomena	RM!E6 events
Cross(red, solid, x, y)	DrawPoint(cross, red, solid, sx, sy)
Dot(white, solid, x, y)	DrawPoint(dot, white, solid, sx, sy)
Line(gray, solid, x_1, y_1, x_2, y_2)	DrawLine(gray, solid, sx_1, sy_1, sx_2, sy_2)
Line(brown, dashed, x_1, y_1, x_2, y_2)	DrawLine(brown, dashed, sx_1, sy_1, sx_2, sy_2)
Polygon(blue, filled, Seq(x_1, y_1, ..., x_n, y_n))	DrawPolygon(blue, filled, Seq(sx_1, sy_1, ..., sx_n, sy_n))
Polygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))	DrawPolygon(white, outline, Seq(sx_1, sy_1, ..., sx_n, sy_n))
Polygon(white, outline, Seq(x_1, y_1, ..., x_n, y_n))	DrawPolygon(white, outline, Seq(sx_1, sy_1, ..., sx_n, sy_n))

Table 5.6: Specification for the *View* event in the park rendering machine

response to *Pan* events. The requirement of the graphical modeling machine is that the *Scale*, *CenterX*, and *CenterY* phenomena from the graphical map are modified in accordance to the Warden's requests.

The rendering machine must calculate the screen coordinates of each graphical map element considering the center and scale. Further, the machine must issue *Draw* events to the *Fire Station Terminal* so that the elements that should be visible on the viewing window are drawn in the appropriate positions.

5.5 Selection problem frame

Problem description

The problem is to select a point or region on the map to perform some operation. The selection is done through direct manipulation with the aid of an input device such as a mouse, keyboard, digitizing table, stylus, or touch screen. The user picks a set of coordinates on the video display device which the machine must convert into geographic coordinates.

A sample problem from this class

The second example shown in the previous problem frame involved visualization of a portion of the map according to the user's commands. The problem was stated as follows:

Each station has a computer terminal with the park map, where the user can zoom in and out and select the viewed region.

However, the problem description for the visualization frame did not include the procedure on how to obtain input from the user to determine where to zoom in or out, or how to select the viewed region. The visualization frame simply assumes that input is available, perhaps by typing in the numeric values for the geo-coordinates.

The full problem description, however, states that the input must be made through direct manipulation:

The system must be able to receive input from a mouse, allowing the user to select a point on the map shown on the screen to determine the center of zooming or viewing.

This is an instance of the selection problem.

Frame diagram, domains, and shared phenomena

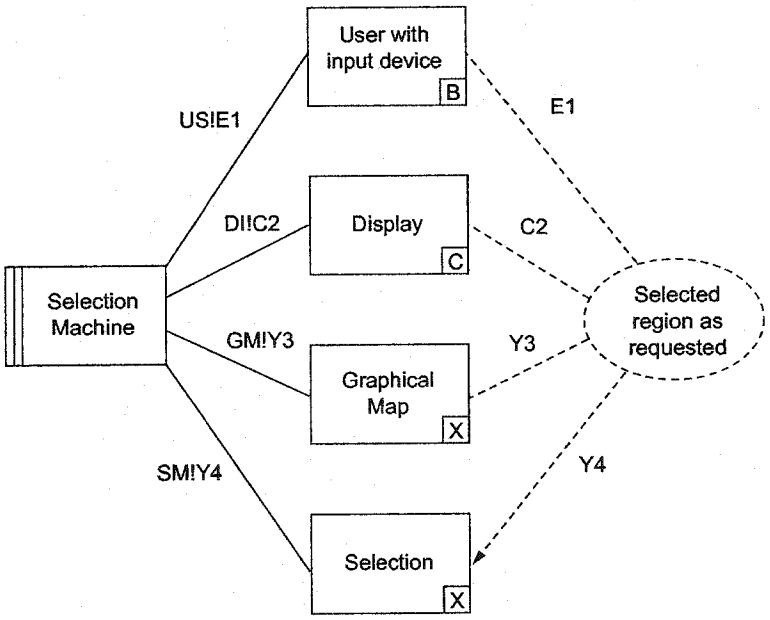


Figure 5.13: The selection frame diagram

Figure 5.13 presents the selection frame diagram. The *graphical map* and *display* domains in the selection frame have the same roles as in the visualization frame. The *user with input device* domain is a biddable domain. The user controls an input device and uses it to issue events to the machine. *Selection* is a lexical domain which contains the geographic coordinates corresponding to the display coordinates selected by the user. This is the domain constrained by the requirements.

The user issues input commands E1 to the machine containing the desired screen coordinates. These commands depend on the input devices used in the problem. Examples of such commands are clicking a mouse button, pressing a key on a digitizer keypad, and touching the screen. The display shares its resolution and other relevant information with the machine through causal phenomena C2. The graphical map shares its state through symbolic phenomena Y3. Given C2 and Y3, the machine transforms the screen coordinates in E1 into geographic coordinates which are shared with the selection domain through symbolic phenomena Y4.

This frame is related to the Model/View/Controller pattern [KP88]. The graphical map can be considered the model, the display is the view, and the machine is the controller which reacts to the user input.

Requirements

The requirement should describe the range of acceptable results. In general, a pixel on the screen has many geographic coordinates, depending on the scale of the map and the resolution of the display. The coordinates in the *selection* domain, when mapped back to screen coordinates, should be equivalent to the coordinates given in event E1.

Specification

The specification should contain the mapping from screen coordinates to geographic coordinates. This depends on the coordinate system used in the map. For each type of coordinate system (for example, UTM or latitude-longitude) there is a different procedure for calculating the mapping. The mapping is a function of properties of the display domain (for example, resolution and size) and graphical map domain (for example, scale and center).

Framed sample problem

The diagram in Figure 5.14 shows the following problem fitted to the selection frame:

The system must be able to receive input from a mouse, allowing the user to select a point on the map shown on the screen to determine the center of zooming or viewing.

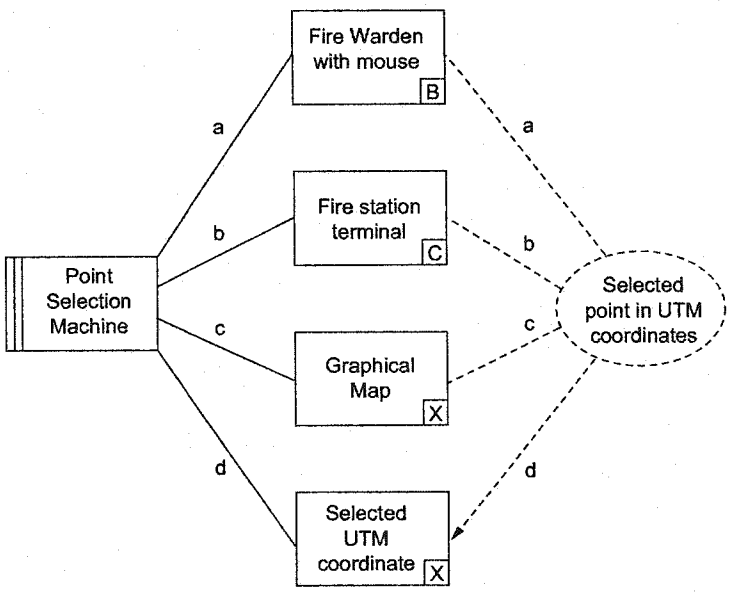


Figure 5.14: Selection frame for the task of picking a point on the map using a mouse

The user clicks on the map shown on the screen, generating a *MouseClick* event that contains the current screen coordinates (sx, sy), measured in pixels with the origin at the bottom left corner of the display. The display shares with the machine its current *resolution* rx, ry (number of pixels in the x and y axis) plus the *size* of the display (szx, szy) in meters, for each dimension. The graphical map shares the UTM coordinates of the point (cx, cy) currently at the *center* of the display, and the current *scale* of the displayed map. The machine calculates the UTM coordinates (utmx, utmy) of the selected screen point (sx, sy) as follows:

$$utmx = cx - \frac{|rx/2 - sx| * szx}{rx * scale}$$
$$utmy = cy - \frac{|ry/2 - sy| * szy}{ry * scale}$$

A pixel may contain many UTM coordinates depending on the scale and resolution. The requirement for this problem is that the (utmx, utmy) coordinates, when mapped back to the display screen, must lie inside the pixel that was selected with the mouse.

Chapter 6

Informal problem description and analysis — using maps to address real-world problems

In the previous chapter, informal descriptions and analysis of problems that deal with the creation and manipulation of maps were presented. In contrast, all problems described in this chapter are based on the existence of a map. They use the map as a source of information about the real world. In Chapter 3 nine different classes of problems that make use of a map were introduced: measurement, classification, resource location, resource inventory, proximity, resource topology, routing, site selection, and spatial definition. In this chapter, each of these classes is described as a problem frame. The frames in this chapter were developed in this dissertation as they are specific to geography.

Although many of these frames resemble each other at first glance, they differ in the input and output problem domains, the characteristics of the model — the map — that is required to solve each problem, the phenomena shared with the machine, and the information relation provided in the requirement. The problems are quite distinct from each other.

6.1 Measurement problem frame

Problem description

The problem is to calculate measurements such as distance, length, angle, or area, that can be derived from geographic coordinates. A complete description of the problem should contain a clear definition of the measurement in order to allow its correct calculation.

A sample problem from this class

In the Algonquin Park problem, in order to dispatch fire rescue teams, the fire warden at the designated fire station may want to calculate the distance in kilometers from the fire station to the transmitter that emitted the alert. For this purpose, *the system needs to be able to calculate the distance in kilometers between two locations given as UTM coordinates.*

Frame diagram, domains, and shared phenomena

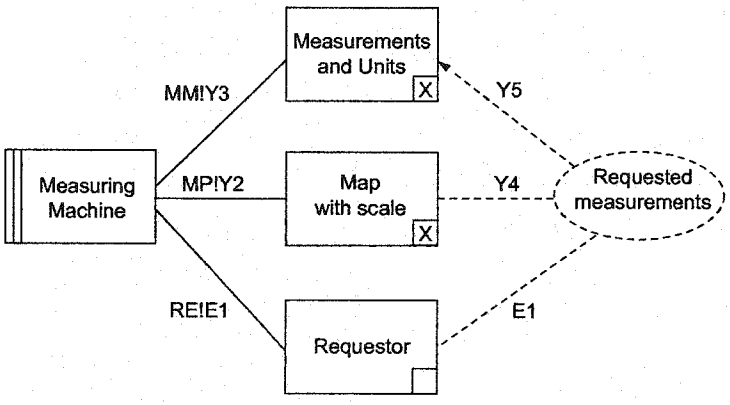


Figure 6.1: The measurement frame diagram

Figure 6.1 presents the measurement frame diagram. The *Requestor* may be a biddable domain such as a user interacting with the machine, or a causal domain such as another machine, a computer system requesting measurements. In both cases, the requestor generates events E1 that are shared with the machine. These events carry the geographic coordinates required to calculate the requested measurement, as well as the units of the measurement, depending on the behaviour of the machine.

The *Measurements and Units* domain is a lexical domain consisting of numeric values and their respective units of measurement. This domain makes the units used to calculate the measurements explicit so that the units are taken into consideration later, during the composition of frames. This is an attempt to stress important aspects of this problem, which although apparently simple and basic, have been overlooked many times during requirements analysis causing fatal errors in the final systems. [1]

The *Map* shares its states Y2 with the machine. This state consists of datum, coordinate system, and projection phenomena. It is used by the machine to compute the measurement correctly.

The requestor issues events E1 to the machine containing the measurement request. The machine processes the request considering the state Y2 of the map and calculates the measurements Y3 that are shared with the measurements and units domain.

The requirements refer to the events E1 and the scale of the map Y4, and constrain the measurements and units domain through phenomena Y5.

Requirements

The requirements for the measurement frame describe the relationship between the values determined by the machine and the values of the measurement determined manually using the map. For example, when calculating distance, the map is used to measure the distance between the two points using a ruler or a mechanical distance measuring device. The map distance is converted to the approximate ground distance taking into account the map scale. The requirement must establish the acceptable error margin for the calculation.

The requirement must also fully describe what measurement is being taken. Consider, for example, a distance measurement. It is not always clear, given the inputs, what type of distance measurement is being sought. If the required measurement is the distance between the coordinates of two polygons, the requirement must state whether the distance is between the two nearest pair of points, or the centers of the two polygons, or some other set of points.

Defining sensible and viable commands is also a part of the requirements for this problem. Sensible commands are those that the machine is able to sense and can react to. For example, a request to calculate the angle between two points makes no sense and is therefore not sensible. Viable commands are those that the machine is expecting and should respond to given its current

[1] In a classic example of failure in requirements, one machine expects the values in one unit and another machine generates the measurements in another unit, causing a failure in the final system. This was the case with the Mars Climate Orbiter, since one team used Imperial units (feet, inches) and the other used metric units [Boa99].

state. A request to calculate distances in which the given UTM coordinates do not represent valid points on the Earth's surface is not viable.

Specification

The specification should contain a description of the method used to compute the desired measurement given the input locations and the map state. This is generally done using mathematical formulae and/or a sequence of steps describing the process. Only sensible and viable commands provide a valid result.

Framed sample problem

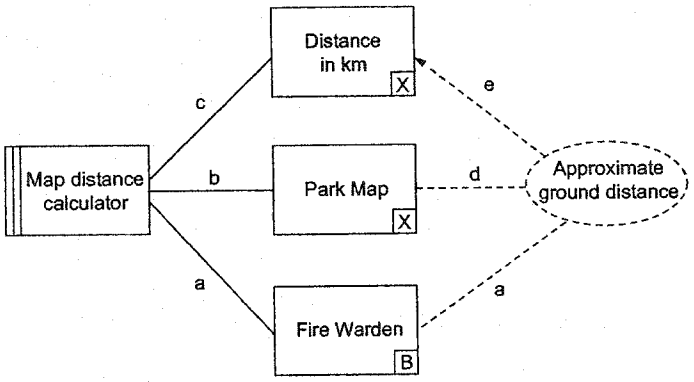
The problem stated previously requires a system able to calculate the distance in kilometers between two locations given as UTM coordinates. The Algonquin Park examples of previous frames have been simplified for clarity, by using (x, y) coordinates. In fact, to identify fully a geographic location in UTM coordinates, three values are required: easting, northing, and zone. The Algonquin Park lies in two different UTM zones: 17T and 18T. This makes the distance calculation more complex than if the park were completely inside a single zone. Therefore, the calculation procedure must be described for this problem. Figure 6.2 presents the problem fitted to the measurement frame.

The machine specification details the procedure to determine the distance. If the UTM coordinates are in the same zone, the Euclidean distance is given by $\sqrt{(e_2 - e_1)^2 + (n_2 - n_1)^2}$. However, if the UTM coordinates are in different zones, there is no mathematical relationship between the coordinate values and the distance. In order to calculate the distance, the UTM coordinates are first converted to the latitude-longitude coordinate system. Then, the spherical distance (value of the great circle arc between the two points) is calculated by approximating the Earth's surface to a sphere of radius equal to 6370 km (determined by the map's datum, which uses the Clarke spheroid [SMB85]). The formula is:

$$\begin{aligned} &\arccos(\cos(long_1) * \cos(lat_1) * \cos(long_2) + \\ &\cos(lat_1) * \sin(long_1) * \cos(lat_2) * \sin(long_2) + \\ &\qquad \sin(lat_1) * \sin(lat_2)) * 6370 \end{aligned}$$

where $(lat_1, long_1)$ and $(lat_2, long_2)$ are the latitude-longitude coordinates of the two points.

The requirements for this problem are that the distance value computed by the machine is within a 5.0% range of the distance measured in the map.



a: FW! { Distance(e_1, n_1, z_1, e_2, n_2, z_2) } [E1]
b: PM! { datum, coordinate system, projection } [Y2]
c: MDC! { NonNegativeRealValue } [Y3]
d: PM! { Locations A and B in the map, scale } [Y4]
e: AGD! { Measured distance from A to B } [Y5]

Figure 6.2: Algonquin Park distance calculation fitted to the measurement frame

6.2 Classification problem frame

Problem description

The problem is to group together discrete or continuous map phenomena that present similar quantitative or qualitative characteristics. The criteria for classification, the number of classes, the range of each class, and the use of constant vs. variable intervals for the classes are important aspects that should be described in this problem since they all influence the result of grouping.

A sample problem from this class

Pursuing the goal of improving fire detection and prevention in the park, the fire patrols and wardens want to study the occurrences of fire in the park. They have been storing a history of how many times each emergency transmitter has been in an alert state since the system was first activated. A system is needed to *classify the transmitters according to the number of times they have been on alert, grouping them in three classes: low, medium and high frequency. The low class includes transmitters with less than 3 alerts, medium varies from 3 to 6 and high includes those with 6 or more alerts. The classified map should also contain the map boundary and the coverage regions.*

Frame diagram, domains, and shared phenomena

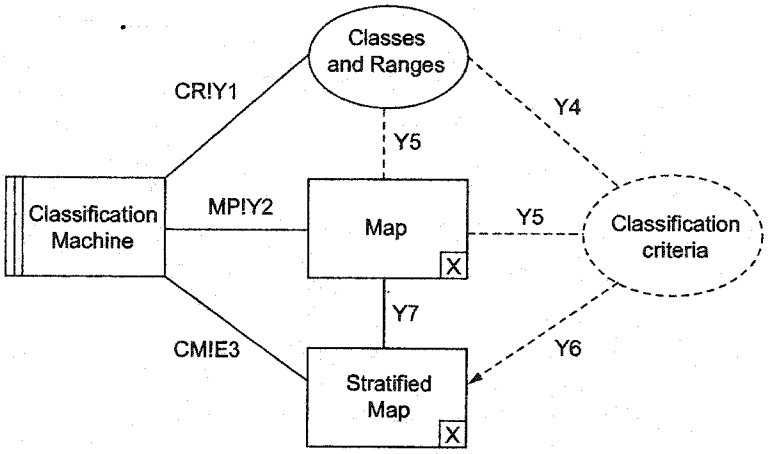


Figure 6.3: The classification frame diagram

The diagram in Figure 6.3 shows the classification frame. It contains a solid oval that represents a description domain, which places constraints on the requirements. This is introduced in the problem-frames approach as a variant of a basic frame [Jac00b]. In the basic frame, all the constraints are embedded in the requirement. This implies that the machine always performs the same task over the shared phenomena from the domains. However, at times it may be necessary to describe a part of the requirements that changes from one use of the machine to another. The core of the frame is still the same, but the machine is more flexible. In this case, the part of the requirement that is allowed to change is separated from the main requirement as a description domain. Description domains are lexical in nature.

Classes and Ranges is a description domain that contains information about the different classes involved in the problem, as well as the range of each class or its membership criteria. If the requirement describes quantitative classification criteria, then this domain should describe the range of each class with a relation such as *ClassRange(c, from, to)*. For example, *ClassRange('Low',3,6)* may describe the class in which transmitters originated emergency alerts with low frequency. If the requirement describes qualitative criteria, then this domain should present a description of the membership criteria for each class as a relation such as *ClassMember(c,m)*. For example, if a map is going to be grouped according to the species of trees, *ClassMember('deciduous', 'oak')* and *ClassMember('deciduous', 'maple')* may describe that oak and maple are tree species that belong to the class of deciduous trees. The *Classes and Ranges* may be cre-

ated by a user. However, its creation is not the responsibility of this machine. Creating such a description is an instance of Michael Jackson's *workpieces* problem frame [Jac00b].

The *Source map* and the *Stratified map* are lexical domains which are similar to each other. The *Source map* can be of any type as long as the classification criteria can be recognized in its objects or fields. The *Stratified map* contains a subset of phenomena from the *Source map* including the result of the classification. The *Source map* shares its phenomena with the *Stratified map*. Any phenomena from the *Source map* that needs to be classified, such as trees or transmitters, should be shared with the *Stratified map*. Any other phenomena from the *Source map* that are also required to be part of the *Stratified map*, usually for geo-reference purposes, should also be shared. For example, when grouping transmitters, the result map must also contain the coverage regions and the boundary of the park. Since these phenomena do not make any difference in the classification, they simply come from the *Source map*.

In the case of field maps, performing a classification means creating a new and more abstract field in the *Stratified map*. For example, a *Source map* containing various numeric temperature values, when grouped into 'cool' and 'warm' classes, will result in a *Stratified map* containing a field mapping each location to 'cool' or 'warm', depending on the location's temperature in the *Source map* and on the ranges established for 'cool' and 'warm'.

In the frame diagram in Figure 6.3, Y7 are the *Source map* phenomena that are shared with the *Stratified map*. This sharing can happen independently of the machine, for example through shared memory in the computer, or it may be the responsibility of the machine to ensure that the sharing takes place.

The machine issues events E3 to create the classification phenomena in the *Stratified map* according to the *Classes and Ranges* and *Source map* domains through shared phenomena Y1 and Y2. The requirement describes the classification criteria. It refers to Y4 phenomena which are shared with the *Classes and Ranges* and to Y5 which are shared with the *Source map*, as perceived by the client. The classification criteria constrains the Y6 phenomena of the *Stratified map*. The dashed line connecting the *Classes and Ranges* to the *Map* indicates that the description domain refers to the map domain.

Requirements

The requirements should describe which phenomena from the *Source map* should be used as criteria for the classification into the *Classes and Ranges*. For instance, in the Algonquin Park example, the classification criteria is the frequency of emergency alerts emitted by each transmitter. Therefore the transmitters (Y5) should be divided into three classes consisting of low, medium and high

frequency (Y4) constraining the membership of each transmitter to a specific class (Y6).

In the case of a *Source map* described by objects, the requirement should also describe if the result of classification must:

- be a partition of the original set
- include only a subset of the original set, leaving some objects out of all classes
- allow objects to participate in more than one group, in which case the classes are not disjoint.

Specification

The specification should describe which events E3 should be issued by the machine to create the groups in the *Stratified map* given the symbolic phenomena Y1 (classes and ranges) and Y2 (map state).

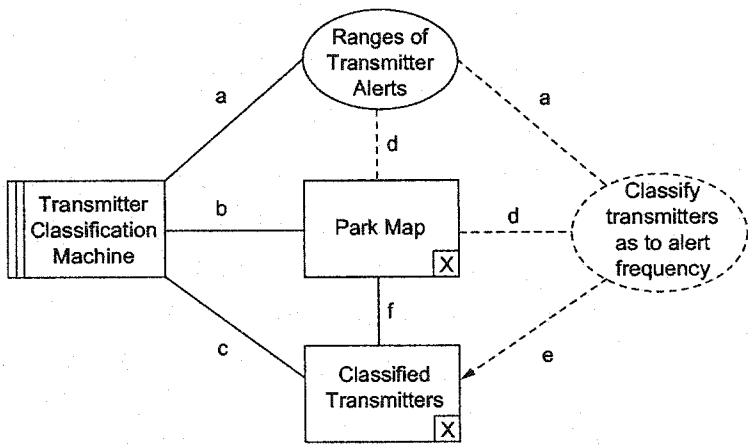
Framed sample problem

The sample problem from the beginning of this section can be fitted to the classification frame. The problem is stated as: *a system is needed to classify the transmitters according to the number of times they have been on alert, grouping them in three classes: low, medium and high frequency. The low class encompasses transmitters with less than 3 alerts, medium varies from 3 to 6 and high includes those with 6 or more alerts. The classified map should also contain the map boundary and the coverage regions.*

Figure 6.4 presents the problem fitted to the classification frame. The *Ranges of alerts* description domain shares the ranges of the three classes with the machine. The same phenomena are also the requirement reference. The *Park map* shares with the machine the part of its state that is relevant for the classification. In this case these state phenomena are the transmitters and their alert history. The specification is presented here as a small program in pseudocode describing what the machine should generate for each class and transmitter.

```
for each class c from Y1
    AddTransmitterClass(c)
for each point p with transmitter t from Y2 {
    determine class c according to AlertHistory(t,n)
    AddTransmitterToClass(t, c)
}
```

The classification criterion is the frequency of emergency alerts emitted by each transmitter. The requirement is that all transmitters should be divided into three classes of low, medium and



a: RTU! { Range(low, 0, 3), Range(medium, 3, 6), Range(high, 6, max) } [Y1]

b: PM! { Point(p, t, x, y), AlertHistory(t, n) } [Y2]

c: TCM! { AddTransmitterClass(c), AddTransmitterToClass(t, c) } [E3]

d: PM! { Transmitter(t, l), AlertHistory(t, n) } [Y5]

e: CT! { TransmitterClass(c), TransmittersInClass(c, t_1, ..., t_n) } [Y6]

f: PM! { Transmitter(t, l), CoverageRegion(r, l), ParkBoundary(l) } [Y7]

Figure 6.4: Grouping park transmitters according to the number of alerts, fitted to the classification frame

high frequency in such a way that the membership of each transmitter in a specific class is constrained by its alert history. Each class should contain all the transmitters with an AlertHistory in that class range. All transmitters should be assigned to one and only one class so that they form a partition of the original set.

6.3 Resource-location problem frame

Problem description

The problem is to find the location of resources or entities in the map given an identification of the resource. It addresses questions of the form "Where is it?".

A sample problem from this class

In Algonquin Park, the fire patrols want to inspect emergency transmitters that have never produced an alert. They want to make sure that these transmitters are working properly. *The patrols need to be able to determine the location (geographic coordinates) of the transmitters that have a history of alerts equal to zero.* Having the geographic coordinates and a GPS, the fire patrols can locate and test these transmitters. This is an instance of the resource-location problem class.

Frame diagram, domains, and shared phenomena

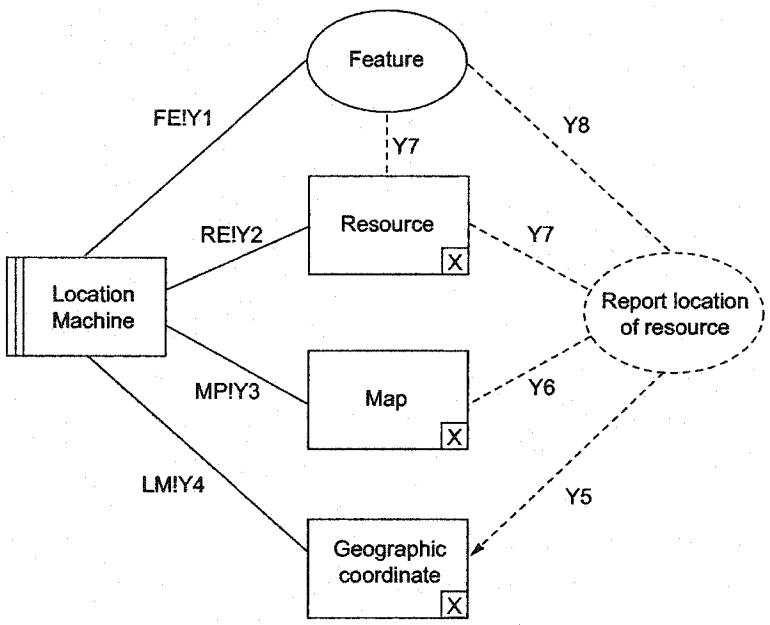


Figure 6.5: The resource-location frame diagram

The diagram in Figure 6.5 shows the resource-location problem frame. The *Map* shares with the machine its state Y3 — the position (geographic coordinates) of each of its resources or, in the case of field maps, the value of the field for each position. The requirement — report location of resources — refers to map state Y6, since the location is given (is found) in the map.

The *Resource* domain identifies all the resources on the map that are candidates to be located. From all candidate resources, only the ones that have the desired features will be located by the machine. The *Feature* description domain contains characteristics or features to select resources for the machine to locate. The description domain refers to the *Resource* domain. Phenomena

Y7 identify all candidate resources, while phenomena Y8 describe the features that the resources must have in order to be located. The feature domain must describe the following:

- the type of resource to locate; for example, transmitters
- the criteria or features to select specific resources of the desired type; for example, the alert history
- the required relationship between the desired values and the actual values of the resource features; for example, the actual value must be less than or equal to the desired value
- the desired values or ranges for each criterion; for example, zero for the alert history.

If the Y6 map-state phenomena consist of fields instead of objects, then the Y8 phenomena must describe:

- the type of resource or field to locate; for example, temperatures or precipitation
- the required relationship between the desired values and the actual values of the field; for example, the actual value must be less than the desired value
- the desired values or ranges for each type of field; for example, 30deg celsius for temperature.

Since the Y8 phenomena for both types of maps should include similar constraints, the explanations about this problem frame focus on object resources. However, note that the map in this problem may consist of field resources.

In addition, the description of the feature domain includes the acceptable shared phenomena: the values or ranges for a specific criterion, the relationship between actual and desired values for each criterion, the possible criteria for each type of resource, or the possible types of resources to locate.

The alphanumeric codes that identify the resources are shared with the machine through Y2. The strings containing feature descriptions are also shared with the machine through Y1. The machine determines from the map state Y3 the geo-coordinates of the resources Y2 that have the specific features Y1, and shares the result with the geo-coordinate domain through Y4 phenomena.

Requirements

The requirement refers to the Y6 map state, Y7 resource identifiers, Y8 resource features and constrains the Y5 phenomena.

The requirement must describe which coordinates should be output as the resource locations. Since the geometries of resources vary in complexity ranging from points to polygons, different problems may require different outputs. For example, the location of a lake may be given as a single coordinate inside the lake, as the smallest bounding rectangle, or as a polygon with a sequence of coordinates that bound its shoreline. The requirement must state which approach is desired.

The correctness and completeness criteria should also be part of the requirement:

- **correctness**: the geo-coordinates reported by the machine must be locations of the resources that have the features and belong to the map.
- **completeness**: there is no other geo-coordinate in the region of interest of the map that is not in the result set and is also a location of one of the resources that have the desired features and belong to the map.

Specification

The specification of the machine should describe the procedure to find the coordinates of the desired resources. The specification depends on the type and complexity of the resources.

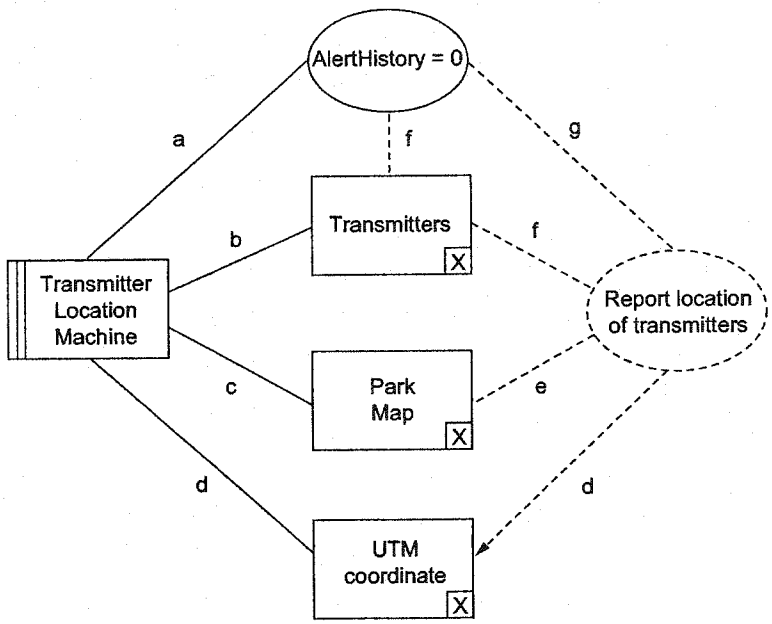
For example, the following is a pseudocode specification of a machine to locate fire stations and transmitters. The expected output for the fire stations is the center point of their delimiting rectangles.

```
for all candidate resources cr
  if cr is in the map
    if cr is a fire station
      get the four coordinates of cr
      find the center c of the rectangle cr
      place c in the target coordinate set
    if cr is transmitter
      get c, the coordinate of cr
      place c in the target coordinate set
```

Framed sample problem

The sample problem from the beginning of this section can be fitted to the resource-location frame. The problem is stated as:

The patrols need to be able to find out the location (geographic coordinates) of the transmitters that have a history of alerts equal to zero.



a: AH! { type 'transmitter', criteria 'alertHistory = 0' } [Y1]
b: TR! { AlphaNumeric codes } [Y2]
c: PM! { Point(p, t, x, y), AlertHistory(t, n) } [Y3]
d: TLM! { utmx, utmy } [Y4]
e: PM! { Transmitter(t, l), AlertHistory(t, n)} [Y6]
f: TR! { identification } [Y7]
g: AH! { transmitters with alertHistory equals to zero } [Y8]

Figure 6.6: Algonquin Park transmitter location fitted to the resource-location frame

Figure 6.6 presents the problem fitted to the resource-location frame. The *Transmitters* domain shares with the machine the alphanumeric codes that identify each transmitter. The map shares its Point-type objects that are transmitters with the machine. The map also shares with the machine the state of the alerts for each transmitter. The machine first determines which transmitters satisfy

the criteria of never having produced an alert. It then finds the coordinates of these transmitters and shares the coordinates with the *UTM coordinates* domain.

The requirements for this problem are that the coordinates for all transmitters with alert history equal to zero that belong to the map are in the *UTM coordinates* domain (completeness) and that all coordinates in that domain refer to desired transmitters (correctness). Since transmitters are represented on the map as point entities, the expected output for each transmitter is a single UTM coordinate.

Variations

The resource-location problem frame can be presented in more than one form. The problem does not change but the focus of the problem and the focus of how information is provided to the machine may change. Three different forms of the problem can be identified:

1. **The machine finds the location of only one type of resource**:

   In this case the feature description-domain does not exist as a separate domain as presented in the Figure 6.5. The information about which resource to locate is embedded in the requirement and in the machine specification. This is the simplest and most specific type of machine.

2. **The machine requires the user to provide details of which resource to locate**:

   In this case the focus of the requirement and the machine specification is the user interaction of soliciting locations and providing information about which resources to locate. The feature description-domain does not exist as a separate domain, but instead the resource-location problem frame has the addition of a user domain. The role of the user is to provide some of the information required to find the desired resources. In the previous form of this frame, the description of the desired resources lay entirely in the requirement. In this case, part or all of this description may be provided to the machine by user commands.

   The information that the user can provide may be related to the value or range of a specific criterion, the relationship between actual and desired values, the criteria themselves, or the types of resources that are desired.

   The more control the user has, the more general-purpose the machine. That is, the machine must be able to deal with any kind of user input that has been defined (for example, finding the location of transmitters with a user-specified value for alert histories), instead of dealing

only with a specific case (for example, finding the location of transmitters with zero alert history). Highly general-purpose machines for resource location are provided in most GIS packages. These machines are not geared towards end users, but towards programmers who must provide all of the information required by the machine (for example, through SQL queries). The programmer's job is to tailor these general-purpose machines to specific problems by restricting the amount of information that the user will provide, thus creating problem-specific machines for resource location.

3. **The domain that describes the resources to be located by the machine is explicitly represented in the frame**:

   In this form of the problem, which is presented in Figure 6.5, the focus is not on the user but on describing the characteristics or features of the resources to be located. When more flexibility is given to the user, the information the user has to provide to the machine is more complex. It is also harder to specify the machine and the requirement in terms of user actions. It is clearer in this case to specify the machine in terms of the description of the features of the desired resources. The user is responsible for creating this description in a file. This is an instance of the workpieces frame [Jac00b]. However, in the resource-location frame, the concern is the description itself and not how the user edits it. The original subproblem is further divided into two frames: workpieces and resource location. The diagram in Figure 6.5 uses the description of the features, as opposed to their creation.

   The role of the features domain is to provide part or all of the information required to determine which resources to locate. The rest of the information is described as part of the requirement. In the previous form of this frame, the description of the features that the resources must have lay mostly in the user actions. Here, part or all of this description may be provided to the machine through the features domain.

Depending on the problem at hand, one of these three forms is more appropriate to frame the description. The third form was chosen to present the problem frames in this chapter, since it makes explicit the information necessary to describe the problem in terms of independent domains.

6.4 Resource-inventory problem frame

Problem description

The problem is to find the resources that have a given set of features and satisfy a given topological relation with a given input region. The common manifestation of this problem is to determine resources that are inside or overlap with a given region. However, other topological relations may also be present in instances of this class of problems. For example, finding resources that cover, meet or are disjoint from the input region are all instances of the resource inventory problem.

A sample problem from this class

The fire fighters in the Algonquin Park use an amphibian water-bomber which lands on water and deploys a scoop while still "taxiing", allowing it to load its tanks with six tons of water in 12 seconds and immediately take off. *The fire warden needs to find the bodies of water that overlap with a given region and that are deep enough to allow an amphibian water-bomber to refill its tanks. For this purpose, the bodies of water must have an average depth of 2 meters or more.*

Frame diagram, domains, and shared phenomena

The diagram in Figure 6.7 shows the resource-inventory problem frame. The requirement is to report the resources that have the described features and that satisfy a topological relation with the input region. Therefore the requirement refers to map state Y7, to the location of the input region Y8, and to the features Y6. It also constrains the resource domain through phenomena Y5.

The *Map* shares with the machine its state Y2. The map can be of any type as long as the required topological relation can be identified in the map phenomena.

The *Input Region* is a collection of contiguous locations. It is a designated region since it is given or described in the problem and it is not defined in terms of other phenomena. The input region shares its boundary with the machine through phenomena Y1.

The *Feature* description domain contains characteristics or features of the resources that are being sought. The description domain refers to the *Resource* domain through Y5. The feature description may contain any of the following: the value or range of a specific criterion, the relationship between actual and desired values, the criteria themselves, or the types of resources that are desired.

The *Resource* domain is the output domain and identifies all the resources in the map that have the described features and satisfy the topological relation with the input region.

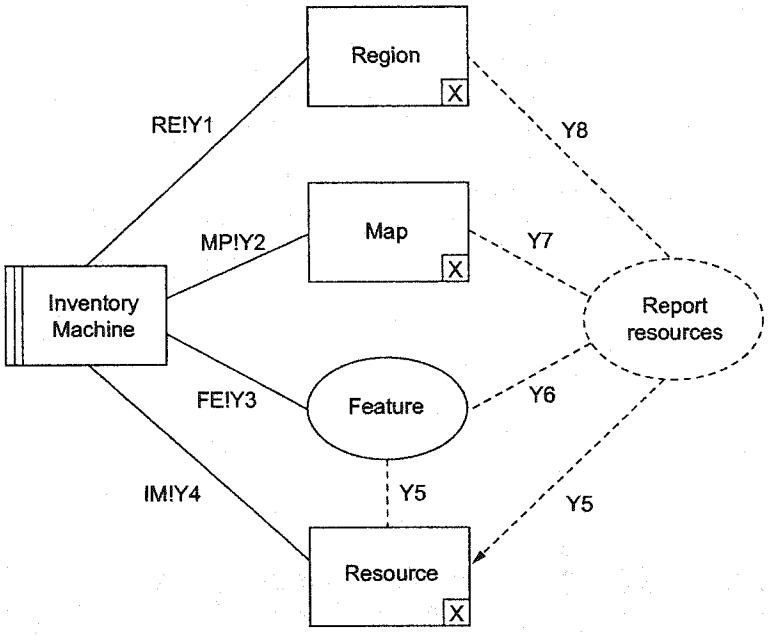


Figure 6.7: The resource-inventory frame diagram

The machine determines which resources from the map Y2 have the features described in Y3 and satisfy the topological relation with the input region described in Y3. The result is shared with the *Resource* domain through phenomena Y4. The result may be empty if there is no resource in the map that satisfies the features and the topological relation with the input region.

Requirements

The requirement for this problem should describe the topological relation between the input region and the resources that are being sought. The requirement is to report the resources that have the necessary features and that satisfy the topological relation with the input region. For example, common requirements for this class of problems are to report specific types of resources that are inside, outside, overlap, cover or meet a given input region. Therefore, the requirement refers to map state Y7, to the location of the input region Y8, and to the features Y6, and constrains the resource domain through phenomena Y5.

There are two important parts in the requirement:

correctness: the resources reported by the machine must satisfy the topological relation with the input region, have the desired features, and belong to the map.

completeness: there is no other resource that satisfies the topological relation with the input region, has the desired features, belongs to the map, and is not in the result resource set.

Specification

The specification of this machine first has to build the topological relation between the input region and the resources located in that region. The topology of the input region is built considering its geographic location, using geometric algorithms. The topology is built upon the position of the boundary of the region. Once the topology is determined, it is straightforward to find which map resources satisfy the topological relation with the input region. Note that only the input region topology needs to be determined, not the topology of the whole map.

For example, the following is a pseudocode specification of a machine used to report all transmitters located inside a given bounding box.

for all transmitters t in the map with coordinates $(tx,\ ty)$
  if $tx > x1\ and\ tx < x2\ and\ ty > y1\ and\ ty < y2$
    place t in the target resource set

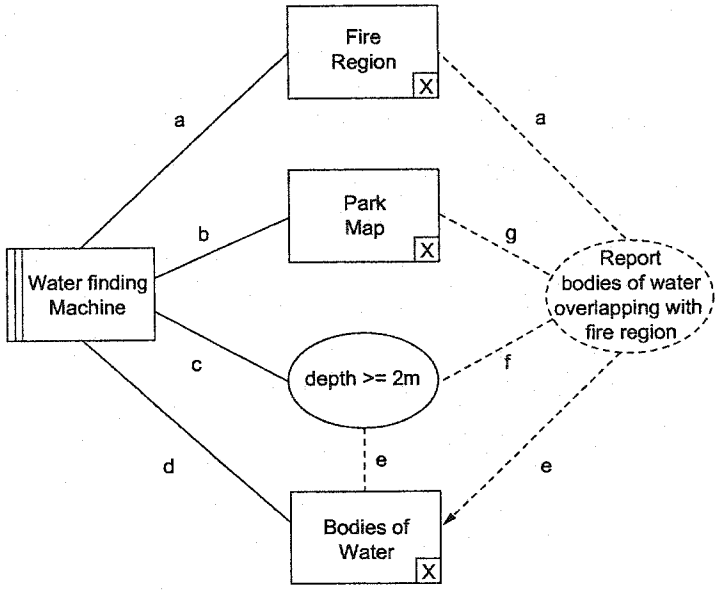
Framed sample problem

Figure 6.8 shows a diagram framing the water-bomber problem to the resource-inventory frame. The sample problem is stated as:

The fire warden needs to find the bodies of water that overlap with a given region and that are deep enough to allow an amphibian water-bomber to refill its tanks. For this purpose, the bodies of water must have an average depth of 2 meters or more.

The location of the input region, or the *Fire Region*, is shared with the machine. The *Park Map* shares the location and average depth of all bodies of water with the machine. The feature domain shares the description that the average depth of bodies of water needs to be greater than 2 meters. In the machine specification only bodies of water with an average depth greater than 2 meters will be considered. The machine specification describes a geometric algorithm to determine if a body of water overlaps with the given region.

The requirements for this problem are 1) all bodies of water in the map with an average depth of 2 meters or more that overlap with the fire region are identified in the *Bodies of Water* domain; 2) all resources identified in the *Bodies of Water* domain belong to the map, overlap with the fire region — there is an intersection between the interior parts of the resource and the fire region — and are bodies of water with an average depth of 2 meters or more.



a: FR! { Region(r, Seq(x_1, y_1, ..., x_n, y_n)) } [Y1]
b: PM! { Region(r, w, Seq(x_1, y_1, ..., x_n, y_n)), averageDepth(w,d) } [Y2]
c: DE! { type 'bodyOfWater', criteria 'averageDepth >= 2m' } [Y3]
d: WFM! { AlphaNumeric codes } [Y4]
e: BW! { identification of body of water } [Y5]
f: DE! { bodies of water with average depth of two meters or more } [Y6]
g: PM! { BodyOfWater(w, l), averageDepth(w,d) } [Y7]

Figure 6.8: The Algonquin Park water finding problem fitted to the resource-inventory frame

6.5 Proximity problem frame

Problem description

The problem is to find resources that satisfy a distance requirement from a given set of resources or locations. Examples of this problem include finding nearest or farthest entities, as well as entities that are within a specific distance.

A sample problem from this class

In the Algonquin Park problem, in order to dispatch fire rescue teams, the fire warden at the designated fire station wants to find the access point that is closest to the fire's location. Access points are the nodes in the network of roads and trails in the park. They consist of road/trail intersections and branching points, fire stations, park gates, and park facilities (restaurants, information booths, washrooms, campsites). Given a point describing the fire location, the fire warden needs to find the nearest node resource — an access point in the roads and trails network.

Frame diagram, domains, and shared phenomena

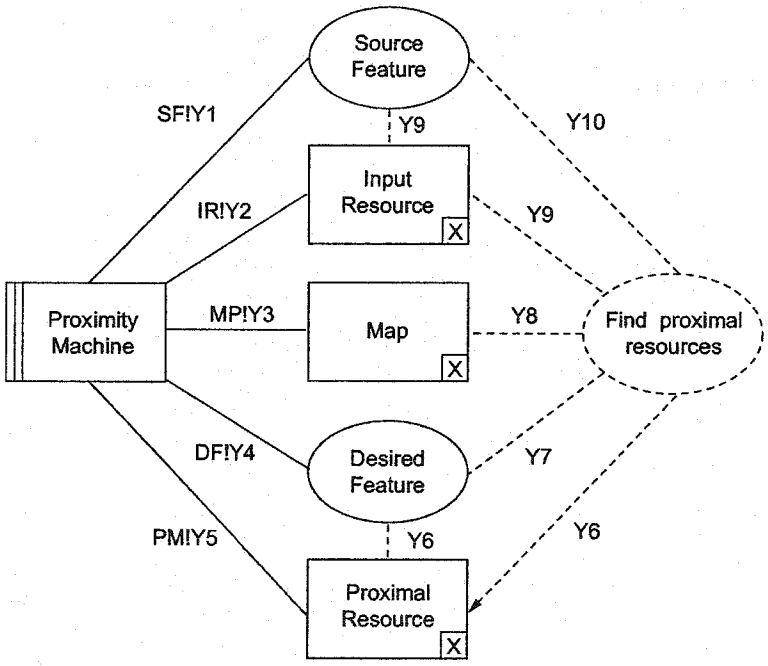


Figure 6.9: The proximity frame diagram

The diagram in Figure 6.9 shows the proximity problem frame given a set of resources as input to find proximal[2] resources. A variation of this problem takes a location as source to determine the proximal resources. The *Map* shares with the machine its state Y3 — the position (geographic coordinates) of each of its resources. The requirement — find proximal resources - refers to map

[2]Lying near or close to something (Oxford English Dictionary).

state Y8, since such resources must be on the map.

The *Input Resource* domain identifies the resources that are candidates for distance evaluation. The machine will consider only the input resources that have the desired features. Their distances to other resources will be evaluated to determine the target resources. The *Source Feature* description domain contains characteristics or features of the resources that the machine will consider. The description domain refers to the *Input Resource* domain. Phenomena Y9 identifies the input resources while phenomena Y10 describe the features that the resources must have to be taken into account when the proximal resources are determined. The feature domain has been discussed in detail in the Resource-location frame (Section 6.3). It may include the type of the resource, the criteria to select resources from the map, the values or ranges for a specific criterion, and the relationship between actual and desired values for each criterion.

Proximal Resource is the domain constrained by the requirement and generated by the machine as a result of machine execution. It contains the resources that satisfy the distance relation expressed in the requirement, taking into account the input resources. The *Desired Feature* domain describes the features or characteristics that the proximal resources must possess. Phenomena Y7 describes these features.

The requirement refers to the input resources, the source features, the map, and the desired features. It constrains the *Proximal Resource* domain.

The alphanumeric codes that identify the resources are shared with the machine through Y2. The strings containing feature descriptions are also shared with the machine through Y1. The machine determines from the map state Y3 the proximal resources that also have the specific features Y4, and shares the result with the proximal resources domain through Y5 phenomena.

Requirements

The requirements must describe the distance constraint that the proximal resources must satisfy in relation to the input resources. Common distance constraints include determining nearest or farthest resources, determining resources that are at a specific distance, or resources that are within a specific distance. In any case, it should be clear what the definition of distance is, and how the distance should be calculated when considering resources with two dimensional geometry: measured from the center, from the boundary of the resource, or perhaps following a different convention.

Specification

The specification of the machine should also contain the distance constraint so that the machine can determine the proximal resources that satisfy the constraint in relation to the input resources. However, the specification should concentrate on describing the procedure to calculate which resources do satisfy the constraint. For example, suppose that the requirement says that the proximal resources should be within x meters of distance from at least one transmitter from a given input set of resources. The specification should describe the procedure to determine the proximal resources as follows:

// all resources that are candidates for distance evaluation
$input\_resources$ is the set of all input resources
// the source feature says that resources considered by the machine should be transmitters
$source\_feature$ = isTransmitter
// resources considered for distance evaluation must have the source feature
$valid\_resources$ = all $input\_resources$ that have $source\_feature$
for all resources r in the map
  if r has the Target Feature
    for all $valid\_resources$ t
      d = distance (r, t)
      if ($d <= x$)
        place r in the $proximal\_resources$ set

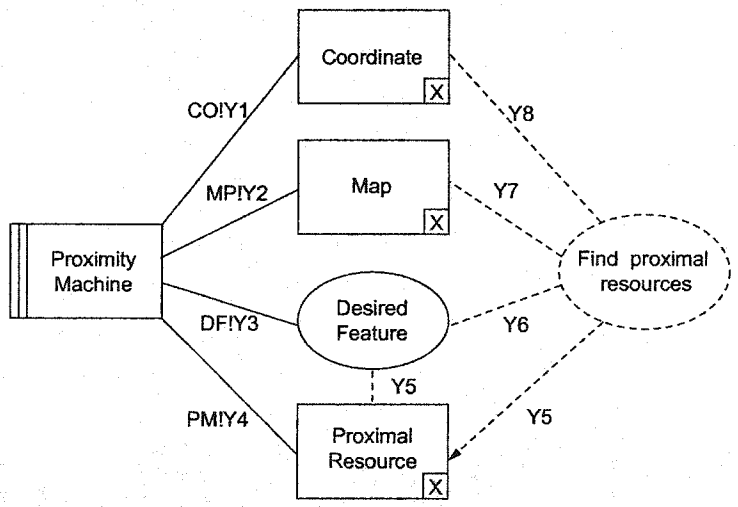


Figure 6.10: Variation of the proximity frame with source coordinate

Variation — Input coordinate

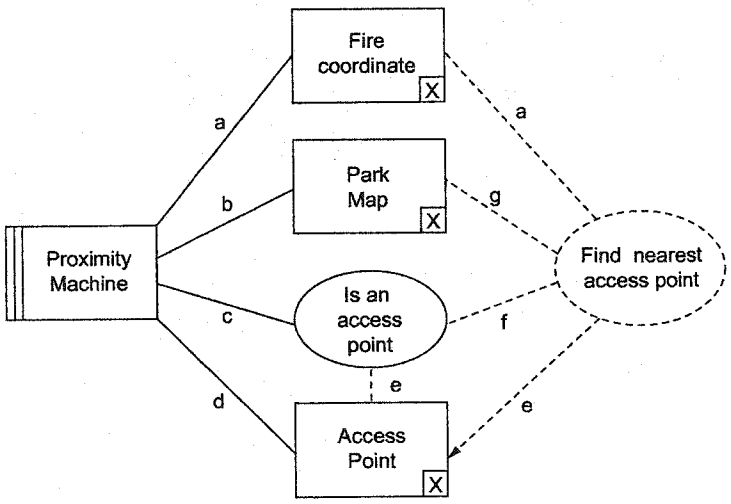
A common variation of this problem frame is to have a coordinate as the source point from where proximal resources are determined. This coordinate is usually a way to identify a point in the map where some action is happening, or simply a point of interest. Figure 6.10 shows the proximity frame diagram in this case.

Framed sample problem

Figure 6.11 presents the following problem fitted to the variation of the proximity frame:

Given a point describing the fire location, the fire warden needs to find the nearest node resource — an access point in the roads and trails network.

The fire location coordinate $utmx$, $utmy$ is shared with the machine. The *Park Map* shares



a: FC! { utmx, utmy } [Y1]
b: PM! { Point(p, x, y), Location(o, p), Type(o, t) } [Y2]
c: DF! { type 'access point' } [Y3]
d: PM! { AlphaNumeric code } [Y4]
e: AP! { identification of nearest access point } [Y5]
f: DF! { objects that are access points } [Y6]
g: PM! { AccessPoint(ap, x, y) } [Y7]

Figure 6.11: Algonquin Park nearest access point problem fitted to the proximity frame

with the machine the location of each resource in the map as well as the type and features of each resource. The *Desired Feature* domain shares with the machine the fact that proximal resources must be access points. The machine specification calculates Euclidean distances from the fire coordinate to all access points (taking any coordinate inside the boundary of a resource) and determines the access points with the smallest distance. This result is shared with the *Access Point* domain.

The requirement says that from all resources in the map, the *nearest* access point(s) to the fire coordinate should be identified in the *Access Point* domain.

6.6 Resource-topology problem frame

Problem description

The problem is to determine existing resources that satisfy a topological relation with the given resources. Examples of topological relations are overlaps, adjacent, inside, or enclosed. This problem may look similar to the resource-inventory problem. The difference lies in the kind of map that is used to describe each problem. Resource-inventory problems are based on maps where the location of each resource is part of its state. Here, the maps are based on topological relations between the regions occupied by each resource. The absolute position of each resource is not so important. In topological maps, every resource has a defined region associated with the points that it occupies. Therefore, a topological map contains topological relations among resources. Topological maps follow the connected object model.

The problems in this class are concerned with the boundaries and interior parts of regions occupied by the resources, and their relationships with other resources.

A sample problem from this class

A computer system is needed at the central station to determine which fire station is responsible for providing emergency help when the station receives an alert signal from the transmitters. Given a transmitter that is emitting an alert, the system should find the coverage region that contains the transmitter in emergency alert. Having found the coverage region, the system should then find the fire station that is inside the coverage region and is therefore responsible for dispatching help.

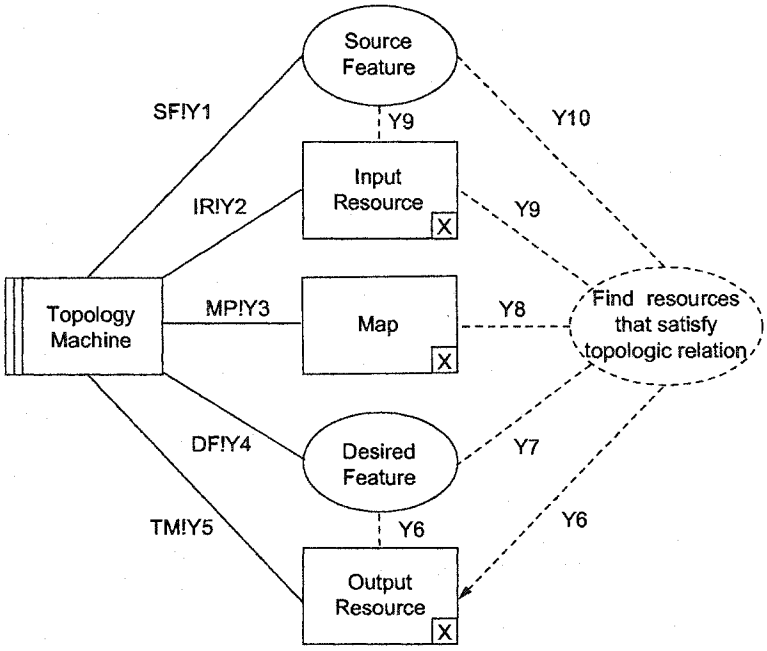


Figure 6.12: The resource-topology frame diagram

Frame diagram, domains, and shared phenomena

Figure 6.12 shows the resource-topology frame diagram. The domains and shared phenomena in this problem are very similar to the proximity problem (see Section 6.5) frame except for the map. The *Map* shares with the machine its state Y3 — the topological relations among the resources. topological relations are usually represented in a map using node, edge and polygon primitives. Nodes represent the beginning, the ending, and any intersection of linear features, which are, in the connected-object model, represented by edges. Nodes bound edges and edges bound polygons. From these simple relationships, more complex ones can be built. For example, if two polygons share a common edge, then they are said to be adjacent. The requirement — find resources that satisfy topological relations - refers to map state Y8, since such resources must be on the map.

There are a few specialized notations that can be used to represent topological information. As mentioned previously, GeoOOA [KPS96] provides specialized graphical symbols to describe topological whole-part and network relations. OMT-G [BLD99, BDL01] and GeoFrame [FI99] provide aids to describing whole-part, network and other topological relations (e.g., coverage, adjacency, overlap). However, these notations focus on creating a conceptual database schema

and are more concerned with integrity constraints for the database than with the definition of the relation itself. When describing maps for the problem frames, the concern is with the definition of these relations.

A map that follows the isolated object model and does not have topological relations established can be converted to a map following the connected-object model. This problem is common in geographic applications and is referred to as the *topology building* problem, which is an instance of the conversion problem frame presented in Section 5.2. The more complex the topological relations required in a problem, the more difficult the process of topology building.

Requirements

The requirements for the resource-topology problem must describe the topological relation that the output resources must satisfy, given the input resources that are constrained by the source features. The topological relation must be clearly described in order to allow for a complete description of the problem. An informal description of the topological relation is acceptable, as long as it has well-defined semantics. For example, suppose the requirement is to determine which coverage regions are crossed by a road in the park. The topological relation *line crosses region* needs to describe how the a line is related to the interior and boundary of the region. Here is the informal definition of the *line crosses region* relation:

- the line and the interior of the region must intersect, and
- the two endpoints of the line must not intersect with the interior of the region.

A formal description of this relation can be found in Appendix D.6.

Specification

The specification of the machine should also contain the topological constraint, so that the machine can determine the output resources that satisfy this constraint, given the input resources and features. However, the machine specification focuses on describing the procedure to determine the output resources. For example, the following description is a high level machine specification to determine the coverage regions crossed by Road 60 in Algonquin Park:

// assuming that the map has nodes, edges, polygons and a relation
// describing which nodes are inside which polygons
// a road is a sequence of edges with different start and end nodes
$internal\_nodes$ = all nodes that belong to the road and are not the start or end node
$external\_nodes$ = start + end node of the road
set1 = polygons that are coverage regions and that have at least one $internal\_node$
inside its interior or in its boundary
set2 = polygons that do not have the $external\_nodes$ inside its interior
output_resources = set1 ∩ set2

This machine specification differs from previous ones with respect to the kind of map phenomena to which it refers. This description is based on the topological relation *node inside polygon* and on the topological representation of the objects on the map (*node, edge, polygon*). Previous descriptions referred to metric relations of the objects on the map (e.g., distance) and to their specific geographic coordinates.

Framed sample problem

Figure 6.13 presents the problem of determining the fire station that should respond to an alert from a transmitter. The map shares its state with the machine providing the nodes, edges, and polygons as well as the relation of nodes that are inside each polygon. The transmitter in alert also shares its alphanumeric id codes with the machine. The machine specification first determines which coverage region contains the transmitter in alert. It then determines which fire station is inside that same coverage region. This fire station is reported to the output resource domain.

The requirement for this problem states that there exists a coverage region that contains both the transmitter in alert and the fire station reported as a result.

6.7 Routing problem frame

Problem description

The problem is to find the node and edge resources that form a route that connects the given node resources. The problems in this class are concerned with a special case of topology involving maps that have *networks*, where nodes are connected through edges. The characteristics of the route (e.g., shortest, longest) are described as part of the requirements.

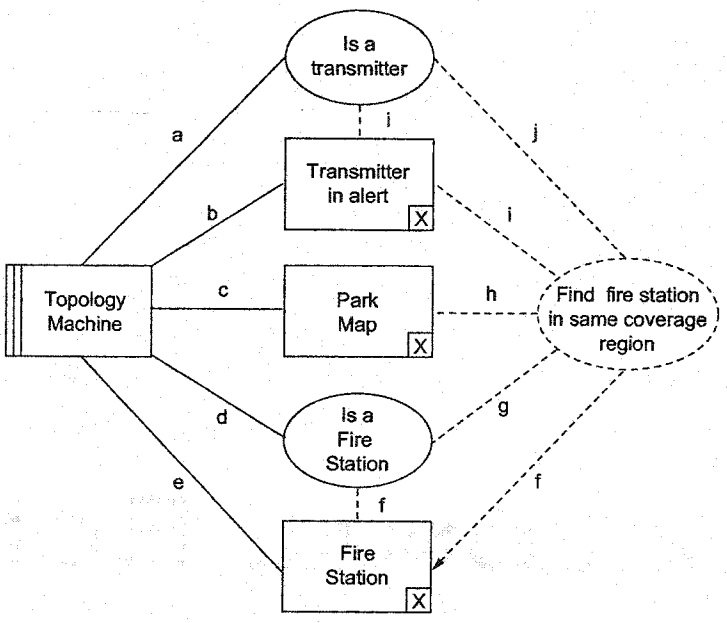


Figure 6.13: Determining the fire station responsible for responding to an alert fitted to the resource-topology frame

A sample problem from this class

In the Algonquin Park problem, in order to dispatch fire rescue teams and equipment, the fire warden at the designated fire station wants to find the shortest route that connects the fire station to an access point close to the fire location. Access points are the nodes in the network of roads and trails in the park. They consist of road/trail intersections and branches, fire stations, park gates, and park facilities (restaurants, information, washrooms, campsites). The edges of this network are the roads and trails. Given an access point (supposedly the closest to the fire location, see

proximity problem in Section 6.5) and a fire station, the fire warden wants to find the shortest route connecting both nodes of the network.

Frame diagram, domains, and shared phenomena

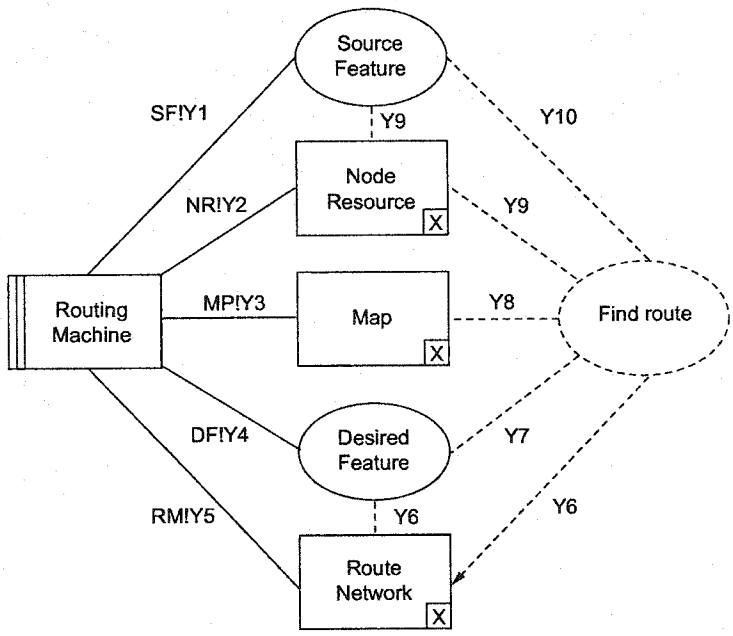


Figure 6.14: The routing frame diagram

Figure 6.14 shows the routing frame diagram. The *Node Resource* domain shares with the machine the alphanumeric codes identifying the nodes that should be in the route (Y2). This domain may provide to the machine all nodes that should be in the route, or only the start and end nodes, or just the start node, depending on the problem. The *Source Feature* domain shares with the machine the characteristics that the node resources must have in order to be valid inputs to the problem (Y1).

The *Map* in this problem is similar to the map in the resource-topology problem. The same type of phenomena (nodes, edges, polygons) are used to describe this map. However, for the routing problem, the map contains only a single type of topology called a *network*. Whole-part topologies or other topological relations such as coverage, adjacency, overlap, etc., are not necessary to solve a routing problem. The network topology is based on a graph, and describes the nodes that are adjacent to each other and the resources that connect them. The map shares the

network with the machine (Y3).

The *Desired Feature* domain shares with the machine the characteristics of the nodes and edges of the resulting route (Y4). For example, suppose the problem is to find a path to the first information booth that can be reached from a campsite in the park, following the roads and trails network. The *Desired Feature* domain would state that the end node of the route must be an information booth. Since the specific information booth will only be known after the machine executes, the characteristic of the end node is described as a desired feature and not as a source feature. The *Desired Feature* domain may also constrain the edges in the route.

Given the input, the features, and the map network, the machine determines a subset of the network that contains the desired route and reports it to the *Route Network* domain (Y5).

The requirement constrains the *Route Network* domain providing the optimization criteria (Y6) to determine the route. It refers to map phenomena Y8 since the route must be a subset of the original map network. It also refers to the source (Y10) and desired (Y7) features of the route, and to the input node resources (Y9).

Requirements

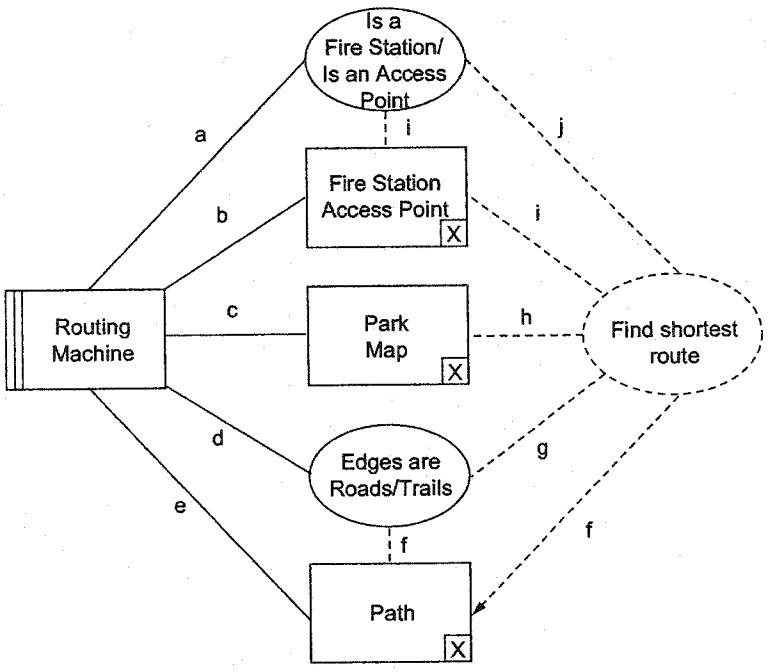
The requirements describe the optimization criteria for the desired route. Common optimization criteria are the shortest or longest distance path, shortest or longest travel time path, and path with the fewest or most nodes or edges of a certain type. For example, find the path that passes through the most tourist attractions.

Specification

The machine specification examines the possible routes on the map that satisfy the desired features. It then selects the resulting route so that it meets the optimization criteria. For example, to find a path that passes through the most tourist attractions, the machine may first look for a path that passes through all attractions, then, in case of failure, reduce the number of attractions iteratively until a path is found.

Framed sample problem

Figure 6.15 presents the problem of finding the shortest path from the fire station to the access point which is closest to the fire, fitted to the routing frame. The park map shares with the machine its network made of roads and trails connecting access points. The inputs to the problem are the alphanumeric code of the start node, which should be a fire station, and the code of the end node,



a: SF! {start node type 'fire_station', end node type 'access_point'} [Y1]
b: SEN! { AlphaNumeric codes of start node and end node} [Y2]
c: PM! { Node(n, type), Edge(e, n1, n2, length) } [Y3]
d: DF! { edge type 'road' or 'trail' } [Y4]
e: RM! { Node(n, type), Edge(e, n1, n2) } [Y5]
f: PA! { path from fire station to access point } [Y6]
g: DF! { edges in the path are roads or trails } [Y7]
h: PM! { roads and trails network } [Y8]
i: SEN! { identification of start node and end node } [Y9]
j: SF! { start node is a fire station and end node is an access point } [Y10]

Figure 6.15: Finding the shortest path from a fire station to an access point, fitted to the routing frame

which should be an access point. The only desired feature of this network is that the edges are roads or trails. The machine determines if the input is valid and calculates all possible routes from the given fire station to the given access point. It then selects the shortest route considering the length of each edge and reports it to the path domain.

The requirement states that no route from the start node to the end note has a shorter length than the one reported as the resulting path.

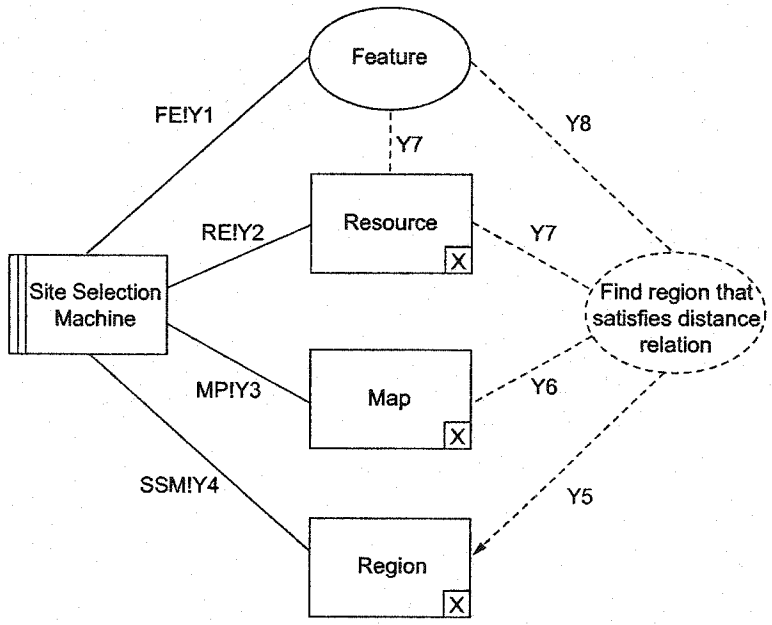


Figure 6.16: The site-selection frame diagram

6.8 Site-selection problem frame

Problem description

The problem is to determine a region that satisfies distance requirements from the given resources. A region is a set of points that do not necessarily have resources. This is the main difference between the site selection and the proximity problems. The proximity problem is concerned with finding proximal resources.

A sample problem from this class

Road 60 is a provincial road that crosses the park bringing accessibility to visitors. Most of the park development and facilities to visitors are situated along the road. This area along the road is of particular interest to the fire patrols and warden.

In order to be more specific about the impact of Road 60 in the park, the fire warden wants to find and define a region that contains all points that are at most 1 km away from the road.

Frame diagram, domains, and shared phenomena

Figure 6.16 shows the site-selection frame diagram. The *Resource* domain shares with the machine the alphanumeric codes of the input resources (Y2). The *Feature* domain describes the characteristics of the input resources. These characteristics are shared with the machine through phenomena Y1. The *Map* shares its state Y3 with the machine. Since this problem is concerned with distances, the map must be described in terms of geographic coordinates or other forms of metric information. Both object and field maps are suitable for this problem. The machine determines if the input resources are valid, according to the feature, and calculates which points follow the distance constraint from the valid input resources and reports the points (Y4) to the *Region* domain. In the object model, a region is a set of points. In the field model, a region is a field that maps locations to booleans — if the value of a location is true, the location is in the region.

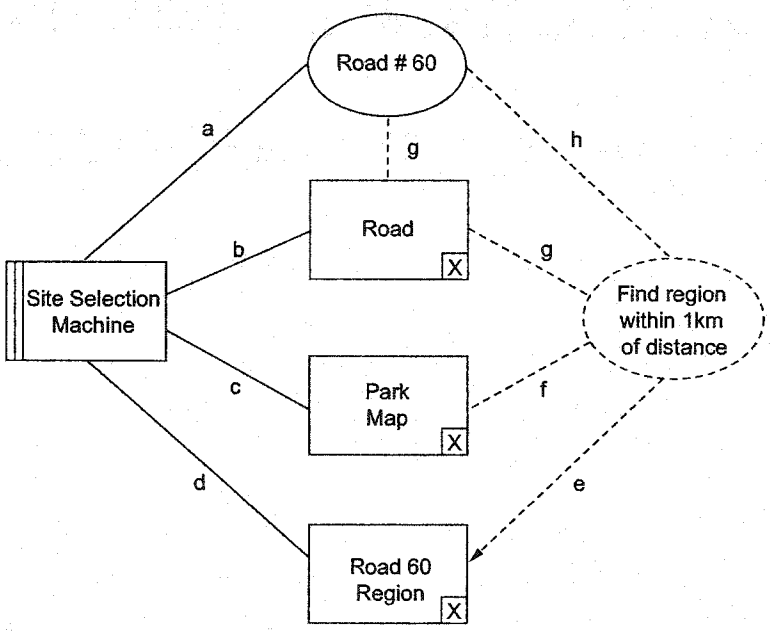
The requirements constrain the *Region* domain (Y5) describing the distance relation between the points in the *Region* domain and the input resources. The requirements refer not only to the input resources (Y7) but also to the map phenomena (Y6) that describe the geographic location of resources or metric relations between points. The requirements also refer to the *Feature* domain (Y8) that describes the characteristics of the input resource.

Requirements

The requirements for this problem should describe the distance relation that the points in the result region must have with the input resources. In general this is the range of distance values that are acceptable between points in the region and the input resources, or even between points in the region and other resources on the map. For example, determining a 10 km buffer of empty regions around a lake involves finding the points that are between 0 and 10 km from the lake and have a distance greater than zero from all other resources in the map.

Specification

The machine specification examines the map state and determines the points on the map that satisfy the desired distance relation with the input resources. The machine reports the resulting points to the *Region* domain. The resulting region may or may not be continuous. If there are no valid input resources, the resulting region may be either empty or equal to all points on the map, depending on the problem. For example, if the distance relation is "all points within 10 km of a hospital," and there are no hospitals on the map, the resulting region should be empty. However, if



a: SF! { type 'road', road_number = 60 } [Y1]

b: RO! { AlphaNumeric code of road } [Y2]

c: PM! { Point(p, x, y), Location(o, p), Type(o, t), road_number (r, n) } [Y3]

d: SSM! { Point(p, x, y) } [Y4]

e: RR! { coordinates of points in Road 60 Region } [Y5]

f: PM! { All points in the map, locations and features of all objects } [Y6]

g: RO! { identification of Road 60 } [Y7]

h: SF! { objects that are road with number equals to 60 } [Y8]

Figure 6.17: The problem of determining the region within 1 km of Road 60, fitted to the site-selection frame

the relation is "all points farther than 10 km from a hospital," and there are no hospitals, then the resulting region should be all points on the map.

Framed sample problem

Figure 6.17 presents the problem of finding the region within 1 km of Road 60, fitted to the site-selection frame.

The *Road* domain shares the alphanumeric codes for the input resources. The feature states that only segments of road with the number 60 should be considered. The park map shares with

the machine all points in the map, the locations of the resources, the type of each resource, and the number of each road segment. Given all of this information, the machine computes the points that form the buffer along Road 60 and reports them to the *Road 60 Region* domain.

The requirement states that all points in the *Road 60 Region* domain are within 1 km of Road 60, and that no other point on the map is within 1 km of the same road.

6.9 Spatial-definition problem frame

Problem description

The problem is to define a new region or field on the map in terms of existing map resources. For example, the task may be to calculate geometries to define a new region for the map in terms of the spatial intersection, union, or difference of the given regions. The same idea applies for defining new fields in terms of the values of existing ones.

A sample problem from this class

Since Road 60 is a major corridor through the park, the warden wants to define a new spatial region on the map in terms of the union of all coverage regions that are crossed by Road 60. Calculating the union of the coverage regions is an instance of the spatial-definition problem. However, finding which coverage regions are crossed by Road 60 is an instance of the topology problem (see requirements and specification of the resource-topology frame in Section 6.6).

Frame diagram, domains, and shared phenomena

Figure 6.18 presents the spatial-definition frame diagram. The *Regions* domain shares with the machine the set of points belonging to each of the input regions (Y1). The input may be a set of resources. Every resource has a defined region which is the set of points occupied by the resource. When the input is a set of resources, their regions are shared with the machine.

The *Map* domain shares all of its points with the machine (Y2). The points are used to determine the validity of the inputs and output, since all points belonging to the regions must be on the map.

The machine creates the *Defined Region* domain by applying the desired computations over the input regions. The new region is reported to the *Defined Region* domain (Y3). In the case of field maps, the input and output are also fields on the map. The values of the defined field are calculated based on the values of the input fields.

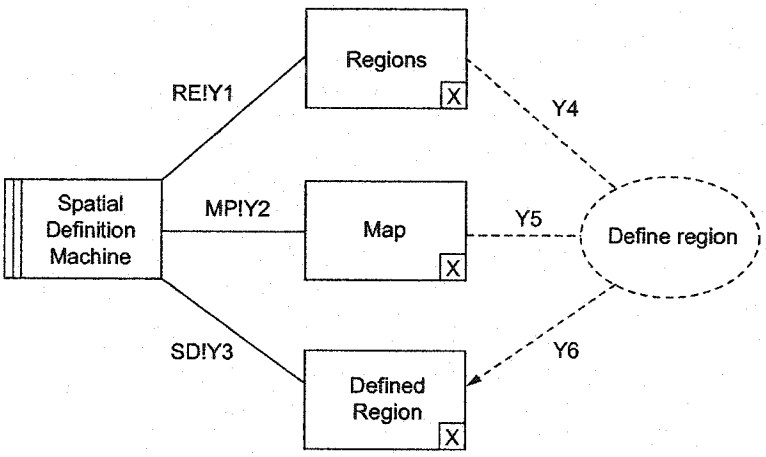


Figure 6.18: The spatial-definition frame diagram

The requirement constrains the new *Defined Region* domain (Y4) and refers to the *Map* (Y5) and input *Regions* (Y6).

Requirements

The requirements for the spatial-definition problem must describe how the regions should be combined. Common examples of requirements for this problem include finding the spatial union, intersection, or difference between the points of the input regions, or applying a function to the values of the input fields.

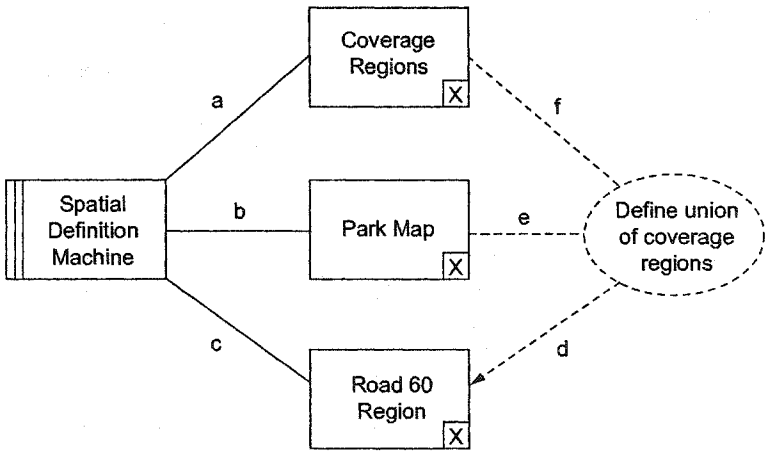
Specification

The machine specification describes how to create the new region given the input regions. For example, if the new region is the intersection of the input regions, the machine specification must calculate the boundary of the new region as the sum of the parts of the boundaries of each input region that are inside (or in the boundary of) every other input region.

Framed sample problem

Figure 6.19 presents the problem of defining the road 60 region as the union of the coverage regions crossed by the road, fitted to the spatial-definition frame.

The *Coverage Regions* share with the machine the points occupied by each region. The *Park*



a: CR! { Region(r), PointsInRegion(r, x, y) } [Y1]

b: PM! { Point(p, x, y) } [Y2]

c: SD! { NewRegion(r), PointsInRegion(r, x, y) } [Y3]

d: RR! { Set of points in the road 60 region } [Y4]

e: PM! { Set of points in the map } [Y5]

f: CR! { Region (set of points) determined by the coverage regions } [Y6]

Figure 6.19: The problem of defining the Road 60 region fitted to the spatial-definition frame

Map provides to the machine all points that belong to the map. The machine determines the union of all input coverage regions, as well as the boundaries of the new region, considering the points that are on the map. The new region is reported to the *Road 60 Region* domain.

The requirement states that every point inside the *Road 60 Region* belongs to at least one of the input *Coverage Regions*. Also, every point in the boundary of the *Road 60 Region* must belong to the boundary of at least one of the input *Coverage Regions*. In addition, every point on the map that is not in the *Road 60 Region* must not belong to any of the input coverage regions.

Chapter 7

Formal description and analysis of geographic problem frames

In this chapter, a formal framework for description and analysis of geographic problem frames is presented. Problem frames can be used to identify the parts of a problem and to describe these parts and the relationships between them. Each frame isolates a single aspect of a complex problem, allowing a detailed understanding of each aspect. The frames provide a diagrammatic, informal description of the problem. If the analyst seeks a formal description, he can further specify the problem or some of its parts using the formal framework presented in this chapter. Formalizing and analyzing a description gives the analyst confidence that the problem has been understood and described correctly.

A formalization of the geographic problems frames is presented in this chapter, focusing on the problems that make use of a map (see Chapter 6). The use of formal methods to describe and analyze problems is time-consuming and expensive. The goal of the work presented in this chapter is to provide an initial formal framework to allow formal descriptions to be created with less effort. The framework concentrates on describing domains, requirements, and machine specifications.

One advantage of formalizing the problem frames is to allow analysis of the problems. This analysis is done by checking properties and generating instances of the described frames. This chapter also presents the kinds of properties this framework will allow an analyst to verify for each problem frame.

The analysis of a geographic problem is done incrementally. An analyst starts by describing and analyzing the map domain. Each subproblem is then described and analyzed in turn, which may lead to revisions and further analysis of the map domain. This approach is illustrated

throughout this chapter.

Before describing the formalization of problem frames, the first part of this chapter explains the choice of notation and tool for this work. The second part concentrates on the different formal representations of a map.

7.1 The Alloy language and constraint analyzer

The Alloy language [Jac] was chosen for this work based on its suitability for describing well-structured data domains. Since in geographic applications the map description can be quite complex, the chosen language must be appropriate to capture this complexity. Many of the existing formal methods focus on behavioural issues such as concurrency and safety. However, these are not the main issues in geographic applications.

Alloy is based on predicate logic and sets, and therefore is suitable for describing facts or truths about the domains involved in a problem. The declarative aspects of the predicate logic statements allow for the description of the effects that a machine must produce in the world — in other words, the *requirements*. On the other hand, Alloy also allows the traversal of sets and relations, showing the procedure by which an input is transformed into the desired output — this is suitable for describing machine *specifications*.

The Alloy constraint analyzer allows the verification of properties specific to each problem. However, at a more general level, it is also possible to verify that the frame concerns hold for each frame. In other words, the analyzer can demonstrate that if the machine specification and the domain properties hold, then the requirements must also hold.

In this chapter, the formal descriptions are presented using the Alloy notation instead of a general mathematical form of predicate logic and sets. The Alloy notation is equivalent to a general notation and can be easily understood. For example, the Alloy statements

```
sig Map {
    P: set Point,
    O: set Object,
    Location: O → P
}
fact { all m: Map | all o1, o2: m.O | o1 != o2 ⇒ m.Location[o1] != m.Location[o2] }
```

are equivalent to:

$Map = \langle P, O, Location \rangle$, where P is a set of Points, O is a set of Objects, and $Location$ is a relation from elements of O to elements of P, such that

$\forall m \in Map \mid \forall o_1, o_2 \in O_m \mid o_1 \neq o_2 \Rightarrow Location_m(o_1) \neq Location_m(o_2)$

Appendix C presents a summary of the Alloy language notation. The Alloy Analyzer beta version, July 2002 build, was used in this dissertation.

7.2 Formal description of map domains

In Chapter 4, different map representations were discussed. Basically, there are the field model, the connected-object model, and the isolated-object model. In formal terms, besides this distinction between fields and objects, there is also an additional concern, which is how geographic coordinates are represented. For example, if a problem requires distances, the map representation must provide either a set of coordinates from which distances can be computed, or the set of all distances between any two objects. If a problem requires topological relations such as *insideness*, the map representation must provide information about the intersections of boundaries and interiors of all objects. Problems requiring information about the connectivity of nodes require a map that represents a network. This section contains various ways in which maps can be formally described. Each of these descriptions captures one projection of the map, which in turn is a model of the real world.

7.2.1 General map elements

A *map* is composed of a set of *points*, which represent locations on a two-dimensional map. The Point type is a given set with no attributes. A sig (signature) in Alloy is the way to define a set.

```
sig Point { }
sig Map { P: set Point }
```

Maps contain *resources*. A resource is a single identifiable entity on a map. Each resource has a set of *features*, which are characteristics that distinguish one resource from another.

```
sig Feature { }
sig Resource { F: set Feature }
```

Some resources are *objects*. Each object has a *geometry*. The geometry represents the dimensions of the resource. There are three different types of geometry. D0 represents zero-dimensional objects (points). D1 represents one-dimensional objects, which are line segments composed of two end points and an interior. D2 represents two-dimensional objects, which have a boundary and an interior; the boundary is composed of a sequence of one-dimensional geometries. However, at this stage the objects do not require details about their geometries; the only requirement is to classify each object according to its dimensionality. Concrete uses of the map may specify further details or constraints on geometries.

```
sig Object extends Resource { G: Geometry }
sig Geometry { }
part sig D0, D1, D2 extends Geometry { }
```

A *region* of the map is also a set of points.

```
sig Region { P: set Point }
```

Object maps are the subset of all maps which contain objects. Each object map contains a set of objects and a function mapping each object to the point (location) where the object is anchored. The exclamation mark after the arrow means that each object O must be related to one and only one point P.

```
sig ObjectMap extends Map {
    O: set Object,
    Location: O →! P
}
```

The other kind of resource is the *field.* A field is composed of a set of points and a function mapping each point to a feature value.

```
sig Field extends Resource {
    P: set Point,
    fn: P →! F
}
```

Field maps are a subset of maps that contain a set of fields. The set of points on these maps are defined as the union of all points of all fields on the map. This is expressed as a fact that immediately follows the Alloy signature definition.

```
sig FieldMap extends Map {
    Fields: set Field
} { P = Fields.Field$P }
```

A few additional facts (axioms) complete the general map definitions: object maps and field maps partition the Map set; Objects and fields partition the Resource set.

```
fact { Map = ObjectMap + FieldMap }
fact { no ObjectMap & FieldMap }
fact { Resource = Object + Field }
fact { no Object & Field }
```

7.2.2 Cartesian projection

The Cartesian projection represents a Cartesian plane where each point has coordinates (x, y). Each of these coordinates has a value. The type Value is described in the NumericValues module, in Appendix D.2.

In the Cartesian projection, every point contains a coordinate pair. Different points must have different coordinates.

```
sig CartesianPoint extends Point {
    x: Value,
    y: Value
}
fact { all p1, p2: CartesianPoint | p1.x = p2.x && p1.y = p2.y ⇔ p1 = p2 }
```

Cartesian maps are a subset of maps where every point has a coordinate.

```
sig CartesianMap extends Map { }
{ P in CartesianPoint }
```

Cartesian map projections are suitable for solving some of the resource location and resource inventory problems.

7.2.3 Metric projections

Metric projections are those that have a *distance* metric. Given a set of points S, a function which takes ordered pairs (x, y) and returns a real number $d(x, y)$ is a distance metric on S if it satisfies the following rules [Wor95]:

1. $d(x, y) > 0$ if $x \neq y$ (non-negativity) and $d(x, x) = 0$; (reflexivity)
   the distance must be a positive number unless the points are the same, in which case the distance will be zero.

2. $d(x, y) = d(y, x)$; (symmetry)
   the distance between two points is independent of which way around it is measured.

3. $d(x, y) \leq d(x, z) + d(z, y)$; (triangle inequality)
   it must always be at least as far to travel between two points via a third point rather than to travel directly.

A commonly used metric is the Euclidean distance over the Euclidean plane. However, the Euclidean metric and plane are defined over the infinite set of real numbers. This set cannot be represented in Alloy or any other model-based formal system, because these systems are based on the fact that the model is finite.[1] The bigger the model, the longer it takes to find valid instances of

[1]Theorem provers are another kind of formal verification system that are able to deal with infinite sets. However, analysts who are not experienced in mathematical proof techniques find it difficult and time-consuming to use theorem provers. Therefore, model-based systems were chosen for this work, since they are more acceptable to the average software developer.

the model. In this work, two different metric projections are presented as a way to solve problems that require metrics.

Metric maps are a subset of the set of all maps. There are two kinds of metric maps: *metric distance* maps, where the distances between points are given, not calculated, and *taxicab* maps, where distances are computed using the taxicab metric [Kra75].

```
sig MetricMap extends Map { }
fact { MetricMap = MetricDistanceMap + MetricTaxicabMap }
```

Given distances

In the first metric map projection, each point has a distance relation that gives the value of the distance from the point to each other point. These distances are given when instances of metric points are generated.

```
sig MetricPoint extends Point {
    d: MetricPoint →! LargeValue
}
```

Although the distances are given, their values are constrained by the three metric rules presented in the beginning of this section. The module Ord is one of the modules provided with the Alloy analyzer tool, and allows for the creation of ordered sets. The function OrdGE is true if the first argument comes after the second argument in the ordered set, or if the arguments are equal (see Appendix D.1).

```
fact { all p1, p2: MetricPoint | {
    // rule 1
    p1 = p2 ⇒ p1.d[p2] = Zero
    p1 != p2 ⇒ p1.d[p2] in Positive
    // rule 2
    p1.d[p2] = p2.d[p1]
  }
}
fact { all p1, p2, p3: MetricPoint |
    // rule 3
    OrdGE(p2.d[p3], subtract(p1.d[p2], p1.d[p3])) &&
    OrdGE(p1.d[p3], subtract(p1.d[p2], p2.d[p3])) &&
    OrdGE(p1.d[p2], subtract(p1.d[p3], p2.d[p3]))
}
```

MetricPoint and CartesianPoint are disjoint subsets of Point.

```
fact { no MetricPoint & CartesianPoint }
```

A MetricDistanceMap is a subset of *MetricMap* where all points are MetricPoints. MetricDistanceMaps and CartesianMaps are disjoint.

```
sig MetricDistanceMap extends MetricMap { }
fact { all m: MetricDistanceMap | m.P in MetricPoint }
fact { no MetricDistanceMap & CartesianMap }
```

Note that the given distances were constrained in groups of three points. However, the rules do not take into account the transitivity required to produce a plane containing a large set of points. Therefore, it is possible that an instance of this map will contain triangles that are correct according to the rules but that cannot be drawn on a two-dimensional plane. Figure 7.1 illustrates this. This projection can be used for problems involving distance but not for precise locations of points.

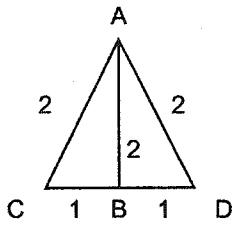


Figure 7.1: ABC, ABD and ACD are valid triangles. CBD are colinear points because the distance from C to D is equal to the distance from B to C plus the distance from B to D. However, points B, C, and D are also on a circle with center in A. Both cannot be true on a two-dimensional plane.

Taxicab distances

Taxicab geometry is an abstraction of Euclidean geometry that is defined for the Integer space, instead of the Real space. In taxicab geometry, points only exist at integer coordinates. The distance between two points is given as the sum of the horizontal and vertical distances between their coordinates, or $d(A, B) = \Delta x_{AB} + \Delta y_{AB}$. This type of geometry is also known as *Manhattan* or *city block* geometry [Kra75]. The names *taxicab* and *Manhattan* stem from the fact that the distance function gives the distance a taxicab needs to travel in Manhattan to get from point A to point B. Figure 7.2 illustrates the taxicab distance definition.

Maps that use taxicab geometry are a subset of CartesianMaps, because the points on these maps have Cartesian coordinates that will be used to compute the taxicab distance.

```
sig MetricTaxicabMap extends CartesianMap { }
fact { MetricTaxicabMap in MetricMap }
```

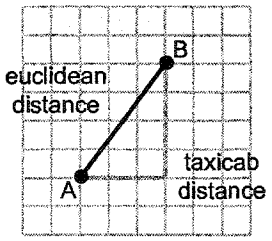


Figure 7.2: Taxicab distance from A to B is equal to 7 while Euclidean distance is equal to 5

Distances are calculated based on the values of Δx and Δy. The following functions are used to compute the taxicab distance. The functions add and subtract are defined in the numericvalues module (see Appendix D.2).

```
fun deltax(p1, p2: CartesianPoint) : Value {
    result = subtract(p1.x, p2.x)
}
fun deltay(p1, p2: CartesianPoint) : Value {
    result = subtract(p1.y, p2.y)
}
```

```
fun taxicabDistance(p1, p2: CartesianPoint) : LargeValue {
    result = { i: LargeValue |
        let dx = deltax(p1, p2) |
        let dy = deltay(p1, p2) | i = add(dx, dy)
    }
}
```

In the Euclidean plane, there is only one line segment that connects two points. However, in taxicab geometry there may be many paths to connect two points, such that all paths have the same taxicab distance. Figure 7.3 illustrates this fact.

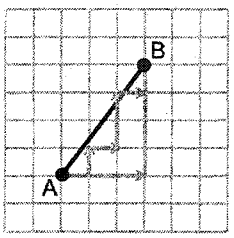


Figure 7.3: Taxicab distance from A to B is equal to 7 regardless of the path taken

For this reason, a point is considered to be "in" the segment that connects A to B if the point is inside the bounding box determined by the two points. This bounding box includes all possible paths from A to B with the same taxicab distance $d(A, B)$. Figure 7.4 shows point C that is "in" segment AB, and point D that is not.

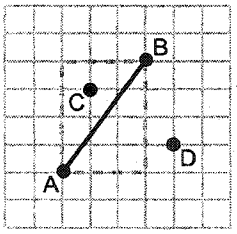


Figure 7.4: Points inside the bounding box determined by AB are "in" segment AB

The function in_segment determines if point p is inside the bounding box determined by points start and end.

```
fun in_segment(p, start, end: CartesianPoint) {
    let minx = min(start.x, end.x) |
    let maxx = max(start.x, end.x) |
    let miny = min(start.y, end.y) |
    let maxy = max(start.y, end.y) | {
        OrdLT(minx, p.x)
        OrdLT(p.x, maxx)
        OrdLT(miny, p.y)
        OrdLT(p.y, maxy)
    }
}
```

According to this definition of in_segment for the taxicab geometry, it will always be the case that if a point is "in" the segment that connects points A and B in Euclidean geometry, it will also be "in" the segment that connects the same points in taxicab geometry. However, the opposite is not necessarily true.

The definition of in_segment can be used to determine if two segments cross. In taxicab geometry, two line segments cross if there is an intersection between the boxes determined by both segments, excluding intersections at the endpoints of the two segments. Figure 7.5 shows two crossing segments, AB and CD.

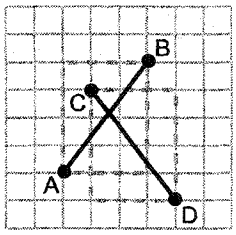


Figure 7.5: *AB* crosses *CD* since their bounding boxes intersect

The following function gives the formal definition of when two taxicab segments cross.

```
fun cross (s1start, s1end, s2start, s2end: CartesianPoint) {
    // The projections of the two segments in the x axis have to overlap
    let s1minx = min(s1start.x, s1end.x) |
    let s1maxx = max(s1start.x, s1end.x) |
    let s2minx = min(s2start.x, s2end.x) |
    let s2maxx = max(s2start.x, s2end.x) |
        OrdGT(s1maxx, s2minx) && OrdGT(s2maxx, s1minx)
    // Same for the y axis
    let s1miny = min(s1start.y, s1end.y) |
    let s1maxy = max(s1start.y, s1end.y) |
    let s2miny = min(s2start.y, s2end.y) |
    let s2maxy = max(s2start.y, s2end.y) |
    OrdGT(s1maxy, s2miny) && OrdGT(s2maxy, s1miny)
    // If the segments S1 and S2 have a common endpoint, the segments intersect
    // if the other endpoint of S1 is inside the segment S2, or vice versa
    (s1start = s2start || s1end = s2end || s1start = s2end || s1end = s2start) ⇒ (
        in_segment(s1start, s2start, s2end) || in_segment(s1end, s2start, s2end)  ||
        in_segment(s2start, s1start, s1end) || in_segment(s2end, s1start, s1end)  ||
        (s1start = s2start && s1end = s2end) || (s1start = s2end && s1end = s2start) )
}
```

Taxicab geometry does not have all the properties of Euclidean geometry. Some properties that hold in the taxicab space will also hold in the Euclidean space. For example, *non-crossing* is such a property. If two segments do not cross in taxicab geometry, they do not cross in Euclidean geometry. Other properties do not hold across both geometries. For example, *equidistance* does not hold. This is illustrated in Figure 7.6.

When using taxicab geometry as an approximation of Euclidean geometry, it is important to consider whether the properties that are being analyzed also hold in Euclidean geometry. If this is not the case, counterexamples must be analyzed carefully to consider whether or not they are also counterexamples in Euclidean geometry.

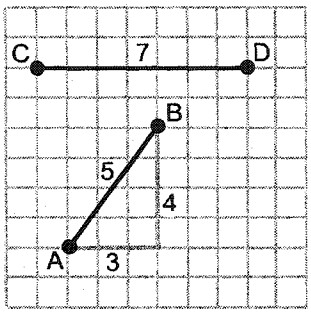


Figure 7.6: Segment *AB* has the same length as segment *CD* in taxicab geometry, but not in Euclidean geometry

7.2.4 Topological projections

Topological projections are used when a problem requires the use of maps that follow the connected-object model. These maps do not take into account the precise location of the objects in the map. Instead, objects are described in relation to each other. Examples of these relationships are intersection between object parts, like their interiors and boundaries; whole-part relationships; and connectivity between nodes in a network. Two formal, topological map projections are presented here: the nine-intersection projection and the network projection.

The TopologicMap set is a subset of ObjectMaps, because field maps do not have topological relations. In these maps, the topological relations between objects are described in terms of their geometries and, therefore, every unique object has a unique geometry. The TopologicMap set is partitioned into two sets, representing the nine-intersection projection and the network projection.

```
sig TopologicMap extends ObjectMap { }
{ all o1, o2: O | o1.G = o2.G ⇒ o1 = o2 }
fact { TopologicMap = NineIntersectionMap + NetworkMap }
fact { no NineIntersectionMap & NetworkMap }
```

Nine-intersection projection

The nine-intersection model was proposed by Egenhofer and Herring [EH91]. The model describes binary topological relations in terms of the intersections of the boundaries, interiors, and exteriors of two objects. The model distinguishes between three kinds of objects: points, lines, and regions.

A *point* object is a zero-dimensional object, having no boundary. A *line* object is one-dimensional. It has a boundary formed by two points. Lines cannot cross themselves or have cycles. A

region object is two-dimensional and has a one-dimensional boundary that separates its interior from its exterior. Regions cannot have "holes", and therefore such regions (such as lakes with islands) cannot be described in this projection. Every object has an interior. The *closure* of an object is defined as the union of its interior and boundary. Every object also has an exterior that is equal to the universe minus the closure. Figure 7.7 shows the parts of a point, line and region.

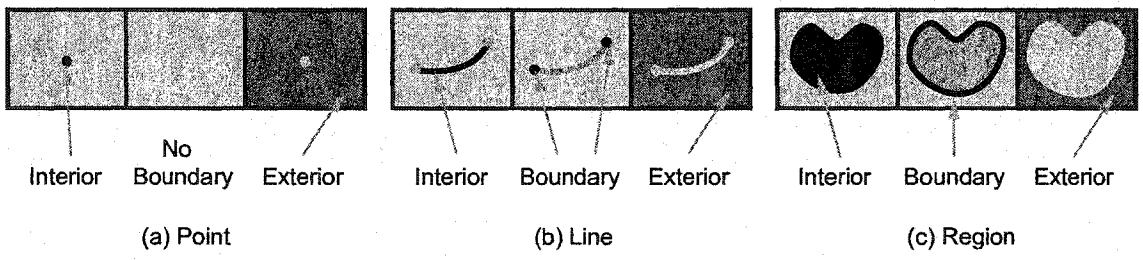


Figure 7.7: Parts of objects

For every pair of objects, the model stores the nine possible intersections among the three object parts (interior, boundary, exterior). Figure 7.8 shows a pair of objects (one line, one region) and the nine-intersection matrix for the pair. A filled square in the matrix indicates that the parts intersect.

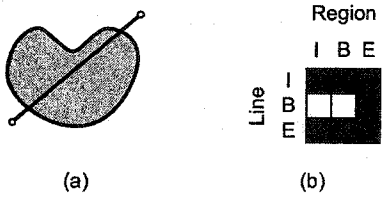


Figure 7.8: (a) a topological relation between a line and a region, (b) the nine-intersection matrix that represents the topological relation shown in (a)

The formal definition for the nine-intersection projection extends the geometry of each object to include the intersection between parts of the object and every other object. For an object o with geometry $o.G$, the set $o.G.IB$ contains the geometries of other objects whose boundaries intersect with the interior of o.

```
sig TopologicGeometry extends Geometry {
    II, IB, IE: set Geometry,
    BI, BB, BE: set Geometry,
    EI, EB, EE: set Geometry
}
```

The nine-intersection matrix is symmetric. It can be read (line, column) or (column, line). The following fact guarantees this symmetry.

```
fact { all g1, g2: Geometry | {
    g2 in g1.IB ⇔ g1 in g2.BI
    g2 in g1.IE ⇔ g1 in g2.EI
    g2 in g1.BE ⇔ g1 in g2.EB
    g2 in g1.II ⇒ g1 in g2.II
    g2 in g1.BB ⇒ g1 in g2.BB
    g2 in g1.EE ⇒ g1 in g2.EE
  }
}
```

There are 512 (2^9) possible relations between each pair of objects. However, most of these combinations are impossible. For example, the exterior of each object must intersect with the exteriors of all other objects. This can be stated as a fact in Alloy:

```
fact { all g: TopologicGeometry | g.EE = TopologicGeometry }
```

Another example of a constraint is that if the interiors of two regions A and B are disjoint, then A's interior intersects with B's exterior.

```
fact { all a, b: D2 & TopologicGeometry |
    (b !in a.II && a !in b.II) ⇒ b in a.IE
}
```

Egenhofer and Herring [EH91] enumerate 23 rules that govern the possible intersections in a nine-intersection matrix. When all the rules are applied, only a small subset of the 512 total possibilities for each object pair become possible. For example, between a line and a region there are only 19 valid topological relations that can be distinguished. All of the rules have been written in Alloy, as part of the nine-intersection projection. The complete set can be found in Appendix D.3.

Several topological relations can be defined using the nine-intersection matrices. For example, the following functions verify whether an object is completely inside the interior, boundary, or closure of another object.

```
// Verifies if g1 is inside the interior of g2
fun inside_interior (g1, g2: TopologicGeometry) {
    ( g2 in g1.II && (g1 in D0 || g2 in g1.BI) && g2 !in g1.(IB + IE + BB + BE) )
}
```

```
// Verifies if g1 is inside the boundary of g2
fun inside_boundary (g1, g2: TopologicGeometry) {
    ( g2 in g1.IB && (g1 in D0 || g2 in g1.BB) && g2 !in g1.(II + IE + BI + BE) )
}
```

```
// Verifies if g1 is inside the closure (boundary + interior) of g2
fun inside_closure (g1, g2: TopologicGeometry) {
    ( g2 in g1.(IB + II) && (g1 in D0 || g2 in g1.(BB + BI)) && g2 !in g1.(BE + IE) )
}
```

The outside function determines whether or not two objects are disjoint.

```
// Verifies if g1 is outside of g2
fun outside (g1, g2: TopologicGeometry) {
    ( g2 !in g1.(II + IB) && (g1 in D0 || (g2 !in g1.(BI + BB) && g2 in g1.BE)) && g2 in g1.IE )
}
```

The nine-intersection model only provides binary relations between two objects. It does not deal with groups of three or more objects. The following constraints were added to the model to account for transitivity in the cases where objects are contained in one another, or are disjoint.

```
//transitivity facts
fact { all g1, g2, g3: TopologicGeometry | {
    inside_closure (g1, g2) && inside_interior (g2, g3) => inside_interior (g1, g3)
    inside_closure (g1, g2) && inside_boundary (g2, g3) => inside_boundary (g1, g3)
    inside_closure (g1, g2) && inside_closure  (g2, g3) => inside_closure  (g1, g3)
    inside_closure (g1, g2) && outside(g2, g3) => outside(g1, g3))
    inside_interior(g1, g2) && inside_interior(g1, g3) => g2 in g3.II
    inside_boundary(g1, g2) && inside_boundary(g1, g3) => g2 in g3.BB
    inside_interior(g1, g2) && inside_boundary(g1, g3) => g2 in g3.BI
  }
}
```

Many higher-level topological relations can be defined using the matrix or lower-level functions.

```
// True when object p is inside region r
fun ObjectInRegion(p: TopologicGeometry, r: D2) {
    inside_closure(p, r)
}

// True when a line crosses a region - that is, when the  interior of the given
// line intersects with the interior of the given region, and the boundaries of
// the line are not inside the region.
fun LineCrossesRegion(line: D1, region: D2) {
    line in region.II && line !in region.IB
}
```

It is also possible to create functions that create geometries that follow specific topological relationships with other geometries. For example, the BoundingRegion function finds a region that contains all the regions given as arguments. Every geometry that is outside of all of the input regions is also outside the resulting region. In this function, every geometry that intersects with the exterior of any input region also intersects with the exterior of the resulting region. There are

other ways of defining a bounding region.

```
det fun BoundingRegion(parts: set D2) : D2 {
    result.II = parts.II + parts + result
    result.IB = parts.IB + parts - result
    result.IE = parts.IE + parts - result
    result.BI = parts.BI - parts - result
    result.BB = parts.BB + result
    result.BE = parts.BE + parts - result
    result.EI = parts.EI - parts - result
    result.EB = parts.EB - parts - result
}
```

Figure 7.9 presents the IsBoundingRegion operation, which checks whether a given set of regions ("parts") is contained in a given region ("whole"). The parts must be disjoint. This is a way to use the nine-intersection model to define whole-part topological relations. Not all whole-part relationships can be expressed using the nine-intersection model. For example, it is not possible to determine if a set of parts form a strict partition of the whole. The function below is enough for a *containment* relation, but not for a *partition* since there may be "holes" between two parts.

A nine-intersection map projection is a subset of topological maps where all objects have nine-intersection matrices. This projection also extends the concept of *region* to include nine-intersection information.

```
sig NineIntersectionMap extends TopologicMap { }
{ O.G in TopologicGeometry
    // Objects only intersect other objects that are in the same map
    all g: O.G | g.(II + IB + BB) in O.G
    // If objects are in the same locations, their boundaries intersect.
    // Since points have no boundaries, their interiors intersect.
    all o1, o2: O | o1 != o2 && Location[o1] = Location[o2] ⇒
        (o1.G in D0  && o2.G in D0  ⇒ o1.G in o2.G.II) &&
        (o1.G in D0  && o2.G !in D0 ⇒ o1.G in o2.G.BI) &&
        (o1.G !in D0 && o2.G in D0  ⇒ o1.G in o2.G.IB) &&
        (o1.G !in D0 && o2.G !in D0 ⇒ o1.G in o2.G.BB)
}
sig TopologicRegion extends Region { G: TopologicGeometry & D2 }
```

The transitivity rules described previously covered cases where objects are completely inside or completely outside one another. There are many interactions between three or more objects that cannot be expressed in the nine-intersection model. For example, there is no way to know if there is an intersection between three regions A, B, and C ($A \cap B \cap C$). Furthermore, since the only relationship between objects is intersection, there is no way to express relationships that rely on other operations such as union. For example, it is not possible to say that the interior of an object is equal to the union of two other objects ($A = B \cup C$).

```
fun IsBoundingRegion(parts: set D2, whole: D2) {
    some parts ⇒ ( {
        // All parts of the whole are disjoint - their interiors do not intersect
        all p1, p2: parts | p1 != p2 ⇒ p1 !in p2.II && p1 !in p2.BI
        // All parts are contained inside the interior + boundary of the whole
        all p: parts | inside_closure(p, whole)
        // Rules regarding when an object intersects with the whole - it must
        // also intersect with some of its parts
        all g: D0 | whole in g.II ⇒ (one  p: parts | p in g.II) ||
                (some p1, p2: parts | p1 != p2 && p1+p2 in g.IB && p1 in p2.BB)
        all g: D0 | (some p: parts | p in g.II) ⇒ whole in g.II
        all g: D0 | whole in g.IB ⇒ (some p: parts | p in g.IB)
        all g: D0 | (some p: parts | p in g.IB) ⇒ whole in g.IB ||
                ((some p1, p2: parts | p1 != p2 && p1+p2 in g.IB && p1 in p2.BB) && (whole in g.II))
        all g: D1 | whole in g.II ⇒ (some p: parts | p in g.II) ||
                (some p1, p2: parts | p1 != p2 && p1+p2 in g.IB && p1 in p2.BB)
        all g: D1 | (some p: parts | p in g.II) ⇒ whole in g.II
        all g: D1 | whole in g.BI ⇒ (some p: parts | p in g.BI) ||
                (some p1, p2: parts | p1 != p2 && p1+p2 in g.BB && p1 in p2.BB)
        all g: D1 | (some p: parts | p in g.BI) ⇒ whole in g.BI
        all g: D1 | whole in g.IB ⇒ (some p: parts | p in g.IB)
        all g: D1 | (some p: parts | p in g.IB) ⇒ (parts = whole) ||
                (some p1, p2: parts + whole | p1 != p2 && p1+p2 in g.IB && p1 in p2.BB)
        all g: D1 | whole in g.BB ⇒ (some p: parts | p in g.BB)
        all g: D1 | (some p: parts | p in g.BB) ⇒ (parts = whole) ||
                (some p1, p2: parts + whole | p1 != p2 && p1+p2 in g.BB && p1 in p2.BB)
        all g: D2 - parts - whole | whole in g.II ⇔ (some p: parts | p in g.II)
        all g: D2 - parts - whole | whole in g.BB ⇒ (some p: parts | p in g.BB)
        all g: D2 - parts - whole | (some p: parts | p in g.BB) ⇒ (parts = whole) ||
                (some p1, p2: parts + whole | p1 != p2 && p1+p2 in g.BB && p1 in p2.BB)
        all g: TopologicGeometry | outside(g, whole) ⇔ (all p: parts | outside(g, p))
    } )
}
```

Figure 7.9: *IsBoundingRegion* operation

Network projection

The network projection is a topological projection where objects are either nodes or edges in a graph or network. This projection is useful when the problem requires determining routes between resources or reachability among resources.

Node is a subset of Object that contains elements with zero- or two-dimensional geometries. Edge is also a subset of Object, containing elements with one-dimensional geometries. Each edge connects two nodes.

```
sig Node extends Object { }
{ G in (D0 + D2) }
```

```
sig Edge extends Object {
    EndNodes : set Node,
    start: EndNodes,
    end: EndNodes
} { G in D1
  // The "EndNodes" set contains the start and end nodes - this is redundant but
  // different properties are easier to describe using one of the two representations
  EndNodes = start + end
  // Each edge has two distinct endpoints - loops are not allowed
  start != end
}
```

A Network is a structure that combines nodes and edges. It also defines an adjacency relation between nodes in order to allow traversals.

```
sig Network {
    nodes: set Node,
    edges: set Edge,
    adj: nodes → nodes
} { edges.EndNodes in nodes
  all n: nodes | adj[n] = { n2: nodes | some e: edges | e.EndNodes = n + n2 }
}
```

The *Network* set can be used to represent directed or undirected graphs. In the case of undirected networks, there can only be one edge between any two nodes.

```
sig UndirectedNetwork extends Network { }
{ all e1, e2: edges | e1 != e2 ⇒ e1.EndNodes != e2.EndNodes }
```

NetworkMaps form a subset of TopologicMaps. Each NetworkMap contains a Network.

```
sig NetworkMap extends TopologicMap { Net: Network }
{ (Net.nodes + Net.edges) in O }
```

It is possible to add constraints to networks. For instance, the function connected is true if all nodes of a network can be reached from every node. The function tree is true if the network is a tree.

```
fun connected(net: Network) {
    all n1, n2: net.nodes | n1 in n2.*(net.adj)
}
```

```
fun tree(net: Network) {
    connected(net)
    all node: net.nodes |
        all next: net.adj[node] |
            let subgraph = net.adj - (next → node) |
                node !in next.*subgraph
}
```

7.2.5 Analyzing map domain descriptions

One way to analyze the map projections to ensure that their structures correspond to what is expected is to create instances of the maps. Alloy can attempt to find instances of a model that satisfy stated constraints. Attempting to create instances is also a way of detecting the presence of inconsistencies in the map description. If Alloy is unable to generate an instance, it is likely that a contradiction is present in the map.

In order to create an instance, it is necessary to create an Alloy function that contains the desired constraints over the instance. For example, the following creates an instance that contains a network map projection that is a tree and has more than two nodes in the network:

```
fun test_NetworkMap( ) {
    some m: NetworkMap | tree(m.Net) &&
        (some m.Net.nodes) && !(one m.Net.nodes) && !(two m.Net.nodes)
}
```

In order to create the instance, the Alloy command "run" is used. The parameter to the "run" command gives the *scope* of the various sets in the model. The scope is the maximum number of elements that Alloy attempts to create for each set.

```
run test_NetworkMap for 4
```

Map descriptions can also be verified by checking assertions. See Section 7.3.3 for an explanation regarding assertion checking.

7.3 Formal descriptions of problem frames

This section presents an approach to formalizing problem frames to allow the analysis of each individual frame and, in a later step, the analysis of a multi-frame application.

The approach consists of three stages. First, the framework of a generic problem frame is described. This framework determines the main parts of a frame and any constraints that must apply to all instances of that particular frame. For example, in the proximity frame, all resources shared with the machine must be on the map. The second stage consists of specializing the generic frame to solve a particular problem. At this point, the frame addresses a specific class of problems but is not applied to any particular application. The specialization consists of adding additional constraints and behaviour to the generic frame. For example, one particular proximity problem is finding the *nearest* resource to the input; another one consists of finding all resources that are *within a particular distance* of the input resource. The third stage consists of creating an instance of a specific problem constrained to a particular application, assigning real domains to the framework. For example, an application may need to find the nearest pole to a consumer in a power supply map.

A problem frame is composed of three main parts: the requirements description; the machine specification; and the domains that share phenomena with the machine and where the effects of the requirements can be observed. Each problem frame has a different set of domains, a different requirement, and a different machine specification.

7.3.1 Framework of a generic problem frame

The skeleton of a generic problem frame is formalized in Alloy as a signature. The frame contains three elements: machine, requirement, and domains. Both the machine and requirement refer to phenomena from the same set of domains. The signature below is an example from the proximity frame.

```
sig ProximityFrame {
    machine: ProximityMachine,
    req: ProximityRequirement,
    domains: ProximityDomains
}
{ machine.ProximityMachine$domains = domains
  req.ProximityRequirement$domains = domains
}
```

The domains formalized here for the proximity frame are described in Section 6.5. There are five domains in this frame. Since this problem deals with distances, the map must be metric.

This particular instance of the proximity frame uses the taxicab projection. The other domains are sets of features and resources. The resources in the domains are constrained to be a subset of the resources in the map.

```
sig ProximityDomains {
    M: MetricTaxicabMap,
    sourceFeature, desiredFeature: set Feature,
    inputResource, proximalResource: set Resource
}
{ inputResource + proximalResource in M.O }
```

The proximity machine has access to phenomena shared with the domains. The machine may have its own private phenomena. In this case, the machine maintains two sets, validSource and validTarget, which contain the resources from the shared domains that have the source and desired features, respectively. The facts in the machine constrain its private phenomena.

```
sig ProximityMachine {
    validSource: set Resource,
    validTarget: set Resource,
    domains: ProximityDomains
}
{ validSource = { r: Object | r in domains.inputResource &&
    (some r.F & domains.sourceFeature) }
  validTarget = { r: Object | r in domains.M.O &&
    (some r.F & domains.desiredFeature) && r !in domains.inputResource }
}
```

The facts shown in the machine signature represent the weakest constraints over the features that the source and target resources must have. In this case, this constraint is that the valid resources must have at least one of the desired features. Stronger constraints may be added to specific machines.

The requirement constrains one or more of the domains in the frame. It may refer to any of the frame's domains.

```
sig ProximityRequirement {
    domains: ProximityDomains
}
{ // proximal resources are in the map
  domains.proximalResource in domains.M.O
  // resources have the desired features
  all r: domains.proximalResource | some r.F & domains.desiredFeature
}
```

7.3.2 Specific problem frames

A specific problem frame is a subset of a generic frame. It constrains the generic frame by adding the machine's functionality. The specific requirement does not constrain the frame. It is developed separately and appears in the frame concern (see Section 7.3.3).

The specific frame below constrains the proximity frame to solve the *nearest resource* problem.

```
sig NearestProximityFrame extends ProximityFrame { }
{ find_nearest_resources(machine) }
```

The find_nearest_resources function constrains a proximity machine so that it determines the resources that have a desired feature and are nearest to the input resources.

```
fun find_nearest_resources (m: ProximityMachine) {
    // Find the points where each of the validSource resources are located
    let sourcePoints = m.domains.M.Location[m.validSource] |
    // Find the points where each of the validTarget resources are located
    let targetPoints = m.domains.M.Location[m.validTarget] |
    // Find the smallest distance between source points and target points
    // We are computing distances from the locations where the objects are
    // anchored, not taking into account the objects' geometries.
    let dist = OrdSmallest(distances_btw_points(sourcePoints, targetPoints)) |
    // Find the target points that are at the smallest distances from the source points
    m.domains.proximalResource = points_at_distance(dist, sourcePoints,
                                                    targetPoints).~(m.domains.M.Location) & m.validTarget
}
```

The distances_btw_points function determines the taxicab distances between each point in set p1 to each point in set p2.

```
fun distances_btw_points(p1, p2: set CartesianPoint) : set LargeValue {
    result = { v: LargeValue | some pt1: p1, pt2: p2 | v = taxicabDistance(pt1, pt2) }
}
```

The points_at_distance function finds the points in set tp that are at distance dist from some point in set sp.

```
fun points_at_distance(dist: option LargeValue, sp, tp: set Point) : set Point {
    result = { p1: tp | some p2: sp | taxicabDistance(p1, p2) = dist }
}
```

In general, the specific machine specifications use Alloy's relational algebra to traverse relations to determine the desired results. This resembles a procedure that shows *how* to obtain the results given the inputs. In contrast, a specific requirement uses predicate logic quantifiers to describe the elements that should be in the result, and those that should not be. This conforms to the informal idea that the requirements should describe *what* are the effects of the machine over the

domains.

The following is the specific requirement for the *nearest* problem. It describes the proximal resources as the set of resources that are located at the smallest distance from one of the source resources.

```
fun nearest_requirement(pr: ProximityRequirement) {
    // If there is a viable target resource in the map, and one of the input
    // resources is valid, there must be a solution.
    (some r: pr.domains.inputResource | some r.F & pr.domains.sourceFeature) &&
        (some r: pr.domains.M.O - pr.domains.inputResource | some r.F & pr.domains.desiredFeature)
            => (some pr.domains.proximalResource)
    // Determine which of the source resources are valid
    let sources = { r: pr.domains.inputResource | (some r.F & pr.domains.sourceFeature) && r in pr.domains.M.O } |
        // For every resource in the solution:
        all r: pr.domains.proximalResource | {
            // Input cannot be in the output
            r !in pr.domains.inputResource
            // All resources in the solution set must have the same distance from the valid sources
            all r2: pr.domains.proximalResource | {
                smallest_distance(pr.domains.M, r, sources) = smallest_distance(pr.domains.M, r2, sources)
            }
        // No resource outside the solution set must be closer to the valid sources
        no candidate: pr.domains.M.O - pr.domains.proximalResource - pr.domains.inputResource | {
            some candidate.F & pr.domains.desiredFeature
            OrdLE(smallest_distance(pr.domains.M, candidate, sources),
                    smallest_distance(pr.domains.M, r, sources) )
    }
  }
}
```

The smallest_distance function returns the smallest distance value between resource r and one of the resources in set rset.

```
fun smallest_distance(m: MetricTaxicabMap, r: Resource, rset: set Resource) : option LargeValue {
    result = OrdSmallest( {d: LargeValue | some r2: rset | d = taxicabDistance(m.Location[r], m.Location[r2])} )
}
```

Appendix D.4 contains a formal description of the proximity frame using the taxicab map projection, specific for finding resources within a given distance from the input resources.

7.3.3 Frame concern

Analysis of the frame concern is used to verify that the machine specification satisfies the requirements. For every specific frame, given the domain properties and the machine specification, the requirements must be true ($S \wedge D \Rightarrow R$). In Alloy, this is verified using an assert. In the example below, f is an instance of the "nearest" proximity frame. The frame contains the domain properties.

The assert specifies that for every frame, if the machine specification holds, the requirements also hold.

```
assert frameConcern_nearest {
    all f: NearestProximityFrame |
        find_nearest_resources(f.machine) ⇒ nearest_requirement(f.req)
}
```

The "check" command in Alloy is used to verify the assertion. If the assertion is false, Alloy provides a counterexample — in this case, a value of f where the property does not hold. The parameters to the "check" command give the *scope* of the various sets in the model. The scope is the maximum number of elements that Alloy attempts to create for each set. If a counterexample is found, it must be examined in conjunction with the descriptions of the map and subproblem to determine the reasons why the desired property does not hold. The following is a sample "check" command for the above assert. It verified to be true.

```
check frameConcern_nearest for 1 but 3 Point, 3 Resource, 5 LargeValue, 2 Feature
```

Before verifying assertions, it is important to attempt to generate instances of the model to ensure that there are no inconsistencies. In the presence of inconsistencies, counterexamples to assertions will never be found.

7.3.4 Summary

The first stage of the problem frame formalization approach consists of providing a generic frame that gives the main parts of the problem frame and any constraints that all instances must follow.

The second stage consists of further constraining the machine specification and requirements in order to address a specific problem in the class delineated by the generic problem frame.

The third stage, which was not presented in this section, consists of using a specific frame in a particular application. This will be illustrated in Chapter 8 with two case studies.

Although the formalization approach can be applied to any problem frame, the focus of this work was the formalization of the problems that make use of a map, discussed in Chapter 6. Appendix D contains the formal descriptions of the following problem frames:

- Proximity Frame
  - Find nearest resources
  - Find resources within given distance
- Site Selection Frame
  - Find empty regions within given distance
- Routing Frame

  - Find route from given node to node with desired feature
- Resource Topology
  - Find enclosing region
  - Find resources inside region
  - Find regions crossed by line
- Spatial Definition Frame
  - Determine region that encloses the given regions

The specific frames that were formalized are used in the case studies of Chapter 8.

7.4 Analyzing properties of problem frames

After a specific application has been formalized, the description can be challenged by attempting to verify various properties involving the frames and their maps. There are a number of different properties that can be verified. This section enumerates several of these properties for each problem frame. The case studies in Chapter 8 show the verification of some of these properties, with respect to specific applications.

The following list describes, for each frame, the kinds of map projections that are necessary or appropriate for solving the corresponding problem. A list of suggested properties and a formal example are also given for each frame. These properties must be tailored to specific applications in order to be useful.

1. **Measurement problem frame** (metric maps)

   - *Range*

     All measurements in a frame fall within a certain value range.

     $\forall v : measurementValue \mid v \geq MIN \wedge v \leq MAX$

   - *Presence*

     At least one of the resulting measurements has a certain value.

     $\exists v : measurementValue \mid v = CONST$

   - *Comparison*

     The measured values of a frame are bigger/smaller than the values of another measurement frame.

     $\forall v_1 : measurementValue_{frame1}, v_2 : measurementValue_{frame2} \mid v_1 > v_2$

2. **Classification problem frame** (any map — depends on the classification criteria)

   - *Distribution*

     None of the classes are empty (contain no resources).

     $\forall c : Class \mid \exists r : Resource \mid r \in c$

   - *Isolation*

     All resources fall into the same class.

     $\exists c : Class \mid \forall r : Resource \mid r \in c$

   - *Disjointness*

     No resource falls into more than one class.

     $\forall r : Resource, c_1, c_2 : Class \mid r \in c_1 \wedge c_1 \neq c_2 \Rightarrow r \notin c_2$

3. **Resource location problem frame** (Cartesian maps)

   - *Limits*

     Located resources lie within certain limited coordinates.

     $\forall c : OutputCoordinate \mid c_x > MINX \wedge c_x < MAXX \wedge$
     $c_y > MINY \wedge c_y < MAXY$

   - *Overlay*

     Two resources may not be in the same location.

     $(Resource_{frame1} \neq Resource_{frame2}) = \emptyset \Rightarrow$
     $(OutputCoordinate_{frame1} \neq OutputCoordinate_{frame2}) = \emptyset$

4. **Resource inventory problem frame** (Cartesian maps)

   - *Multiple membership*

     If two disjoint regions are used as the input in two instances of the problem, the same resource may appear in both outputs.

     $\exists res_1 : OutputResource_{frame1}, res_2 : OutputResource_{frame2} \mid$
     $(InputRegion_{frame1} \cap InputRegion_{frame2}) = \emptyset \wedge res_1 = res_2$

   - *Subset*

     If the output resources of a frame are a subset of the output resources of another instance of the same frame, the input region of the first frame is a subregion of the second.

$OutputResource_{frame1} \subset OutputResource_{frame2} \Rightarrow$
$InputRegion_{frame1} \subset InputRegion_{frame2}$

5. **Proximity problem frame** (metric maps)

   - *Commutativity*

     If the input to the frame is res_A and the output is res_B, when the input to another instance of the same frame is res_B, the output is res_A.

     $InputResource_{frame1} = res_A \wedge ProximalResource_{frame1} = res_B \wedge$
     $InputResource_{frame2} = res_B \Rightarrow ProximalResource_{frame2} = res_A$

   - *Singleton*

     There is always exactly one output resource.

     $\exists res_1 : ProximalResource \mid$
     $\neg(\exists res_2 : ProximalResource \mid res_1 \neq res_2)$

6. **Resource topology problem frame** (nine-intersection maps)

   - *Feature dependency*

     A resource with feature f_2 is always an output if the input has feature f_1.

     $\forall r_1 : InputResource \mid f_1 \in Feature_{r1} \Rightarrow$
     $\exists r_2 : OutputResource \mid f_2 \in Feature_{r2}$

   - *Transitivity*

     If B is in the solution set of input A, and C is in the solution set of input B, then C is in the solution set of input A.

     $InputResource_{frame1} = res_A \wedge res_B \in OutputResource_{frame1} \wedge$
     $InputResource_{frame2} = res_B \wedge res_C \in OutputResource_{frame2} \wedge$
     $InputResource_{frame3} = res_A \Rightarrow res_C \in OutputResource_{frame3}$

7. **Routing problem frame** (network maps)

   - *Hamiltonian path*

     There is a route that covers all nodes.

     $\exists r : Route \mid \forall n : Nodes \mid n \in Nodes_r$

- *Eulerian path*

  There is a route that covers all edges.

  $\exists r : Route \mid \forall e : Edges \mid e \in Edges_r$

- *Node features*

  There are no nodes in the route that have feature *f*.

  $\forall r : Route \mid \neg(\exists n : Node_r \mid f \in Feature_n)$

8. **Site selection problem frame** (metric maps)

   - *Availability*

     There is always a site available no matter what the input to the frame.

     $\forall r : Resource \mid InputResource_{frame} = r \Rightarrow$
     $OutputRegion_{frame} \neq \emptyset$

   - *Independent buffer*

     Different resources result in different, non-overlapping sites.

     $(InputResource_{frame1} \cap InputResource_{frame2}) = \emptyset \Rightarrow$
     $(OutputRegion_{frame1} \cap OutputRegion_{frame2}) = \emptyset$

9. **Spatial definition problem frame** (any map — depends on the definition criteria)

   - *Containment*

     The defined region contains all of the input regions.

     $\forall r : InputRegion \mid r \subset OutputRegion$

   - *Intersection*

     The defined region contains some part of each input region.

     $\forall r : InputRegion \mid \exists p : Point \mid p \in r \wedge p \in OutputRegion$

7.5 Summary

The formal part of the problem-oriented approach consists of the description and analysis of a map domain and of the geographic subproblems involved in the application. The description and analysis is done incrementally. Each description can be analyzed individually by generating instances and checking assertions. The results of the analysis of a subproblem description may influence the map domain description.

Chapter 8

Case studies

Two case studies are presented to demonstrate how informal and formal problem description and analysis of geographic applications can be performed using the problem-oriented approach introduced in this dissertation. Both case studies use multiple frames to solve real-world problems.

8.1 Power supply network

The first case study is an example of a geographic application from the utilities industry. This section begins with a natural-language description of a geographic problem. A client may provide this description as the basis for the development of a geographic application. The example is adapted from Kosters, Pagel and Six [KPS97]. It was chosen in order to illustrate multiple map-domain projections (metric and network), multiple subproblems involved in the description of a real-world problem, the role of abstraction in formal descriptions, and the suitability of these descriptions for analyzing properties.

8.1.1 Natural language problem description

An electric power supply network is composed of lines for high and low voltage. The high-voltage lines and the low-voltage lines are connected via well-defined transition points called transformers. The high-voltage lines link power plants to transformers. The low-voltage lines connect transformers to consumers. Towers are used to support the high-voltage lines, which are called transmission lines, and poles are used to support the low-voltage lines, which are called cables. Each cable cannot exceed 30 meters in length. Figure 8.1 illustrates a power supply network.

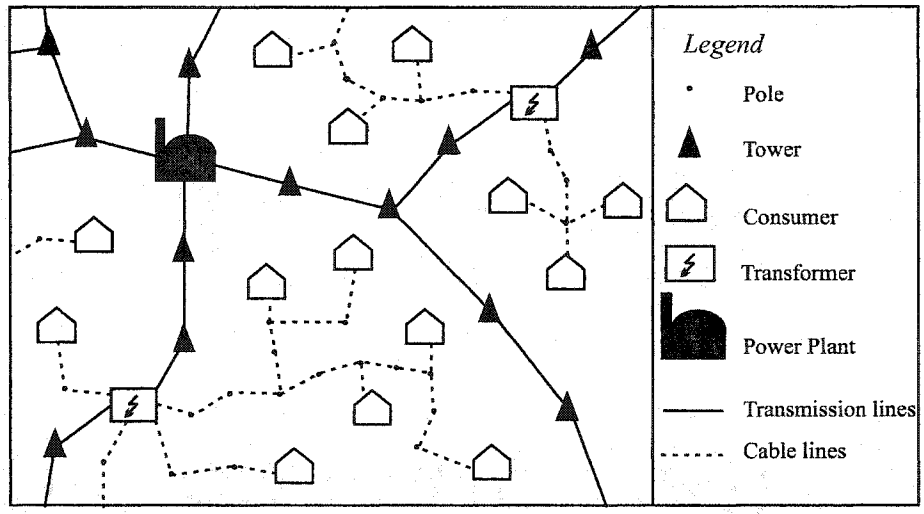


Figure 8.1: An illustration of the power supply network

Each transformer receives power from a power plant. Each consumer has a demand of power supply and each transformer has a maximum capacity that it can receive and distribute. A map of the existing power supply network must be created and used to connect new consumers and determine routes between nodes in the network.

When adding a new consumer to the map, the consumer needs to be connected to the network. A cable is used to connect the consumer to the nearest pole (P_A). If P_A is not within 30 meters, a new pole P_B must be added. The site for P_B lies between the consumer and P_A, at any location, as long as the distances between P_B and P_A and between P_B and the consumer are within 30 meters[1]. Finally, P_B and the consumer are connected by cables to the poles that are nearest to them.

The map is also used to determine routes from consumers to transformers. This information is important for maintenance operations, in case of service interruptions to specific consumers. For planning purposes, the map is used to find the nearest transformer to a consumer. This information is used to plan future expansions of the power supply network.

[1]In this problem, it is assumed that a consumer will never be placed more than 60 meters away from the nearest pole. This is done due to limitations with the Alloy notation that was used for the formal description. A discussion of this and other limitations of the Alloy notation is presented in section 9.4.

8.1.2 Description using problem frames

Map domain description

In order to describe the problems of the power supply network, a connected-object map containing the Cartesian coordinates of the resources is necessary. Some of the problems (e.g., site selection) require a metric projection of the map. Others (e.g., routing) require a network projection. GeoOOA is the chosen geographic notation to describe the map domain. This notation is adequate for representing network relationships, as well as the geometric type of each map resource. In addition, the original version of the power-supply example comes from work on GeoOOA.

Figure 8.2 shows the GeoOOA description of the power-supply map. The description includes symbols for nodes, edges (links), and networks. The nodes for the high-voltage network are power plants, towers, and transformers. They are connected through transmission lines. The nodes for the low-voltage network are consumers, poles, and transformers. They are connected through cable lines. Transformers connect both networks.

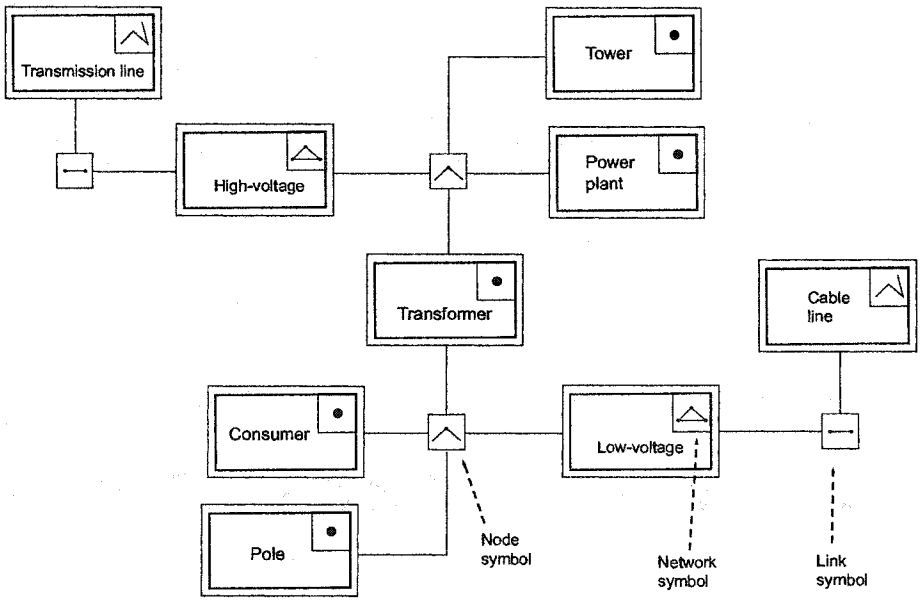
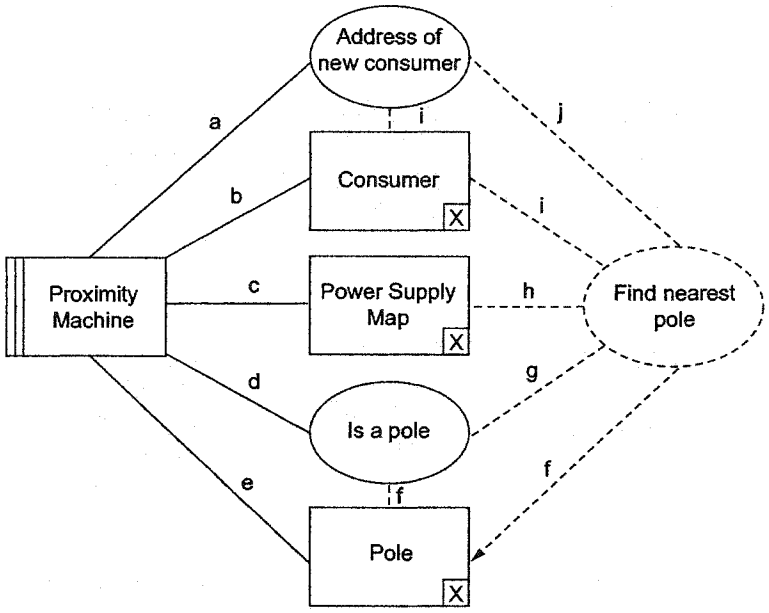


Figure 8.2: The power supply network described using GeoOOA (from [KPS97])

Proximity problems

There are four instances of the proximity problem frame in the power supply example. The first is used to find the nearest pole to a new consumer. The second is used to find the poles within 30 meters of a consumer. The third is used to find the pole that is nearest to another pole. Finally, the fourth is used to find the transformer that is nearest to a consumer. Figure 8.3 shows one of these — the problem of finding the nearest pole to a new consumer.

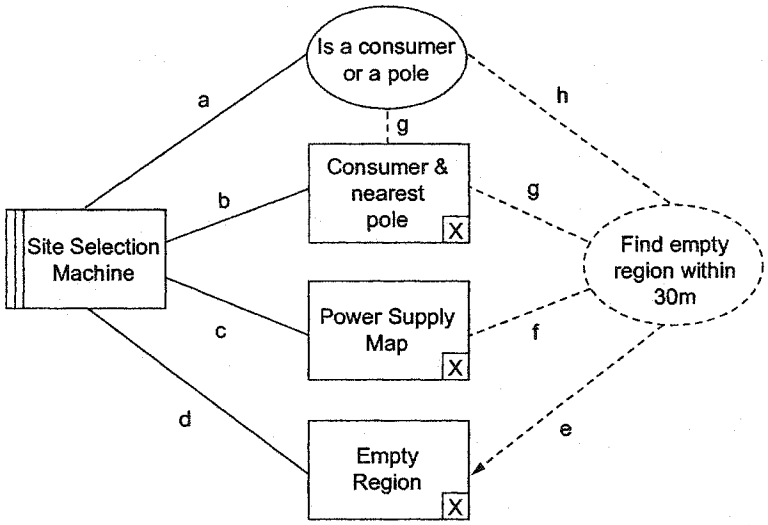


a: SF! { type 'consumer', criteria 'address=address of new consumer' } [Y1]
b: CO! { AlphaNumeric codes } [Y2]
c: PSM! { Point(p, x, y), Location(o, p), Type(o, t), Address(o, a) } [Y3]
d: DF! { type 'pole' } [Y4]
e: PM! { AlphaNumeric code } [Y5]
f: PO! { identification of nearest pole } [Y6]
g: DF! { objects that are poles } [Y7]
h: PSM! { Consumer(co, x, y), Pole(po, x, y), Address(co, a) } [Y8]
i: CO! { identification of consumers } [Y9]
j: SF! { address of consumers that are not connected } [Y10]

Figure 8.3: Finding the nearest pole from new consumer

Site selection problem

The problem of determining locations where a new pole can be placed is an instance of the site-selection frame. Figure 8.4 shows this problem fitted to the frame.



a: SF! { type 'consumer' or type 'pole' } [Y1]
b: CP! { AlphaNumeric codes } [Y2]
c: PSM! { Point(p, x, y), Location(o, p), Type(o, t) } [Y3]
d: SSM! { Point(p, x, y) } [Y4]
e: ER! { coordinates of points in empty region } [Y5]
f: PSM! { All points in the map and locations of all objects } [Y6]
g: CP! { identification of consumers and poles } [Y7]
h: SF! { objects that are consumers or poles } [Y8]

Figure 8.4: Finding empty region within distance from a new consumer and its nearest pole

Routing problem

Figure 8.5 shows the problem of finding the route from a consumer to the transformer to which it is connected, fitted to the routing problem frame. Note that the transformer is not known beforehand, and determining it is part of the problem.

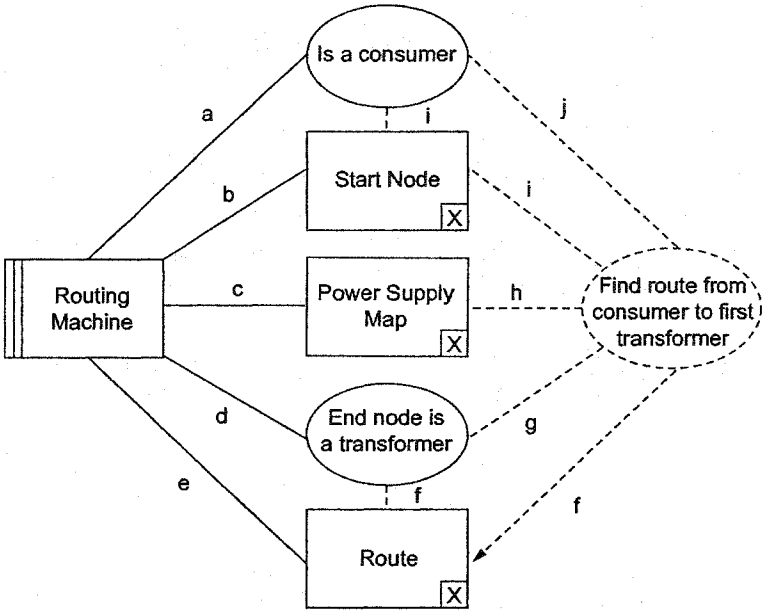


Figure 8.5: Finding the route from a consumer to the first transformer

Summary

In order to connect a new consumer to the network, the following problems should be put together using a metric projection of the map: find the nearest pole to the new consumer; find the poles within 30 meters of the new consumer; verify that the nearest pole is within 30 meters; if it is, connect the consumer to the nearest pole; if not, find locations where a new pole can be placed, find the nearest pole to the new pole, connect the new pole to its nearest pole, and connect the consumer to the new pole. The editing steps of adding a new pole and connecting consumers and poles are instances of the editing frame, which are not shown here.

In order to answer maintenance requests from consumers, it is necessary to find the route from the consumer to the transformer that connects it to the high-voltage network. This uses a network map projection.

Finally, planning future expansions of the network requires determining the nearest transformer to each consumer. This requires a metric map projection.

8.1.3 Formal description using Alloy

The informal description of the problem frames details shared phenomena, domain properties, requirements, and specifications of each of the problems in the power supply network. However, this description is not suitable for the purpose of automated analysis. For example, it is hard to tell from the informal descriptions whether or not adding a new consumer to the network may result in crossed cables. This can be done using formal descriptions of the problem frames.

Formal map domain description

The problem frames that use the power-supply map require two different map projections. Some frames use a metric projection and some use a network projection (see Section 7.2.4). The taxicab projection (see Section 7.2.3) is used as a metric projection because distances are necessary for finding the proximal resources, and coordinates are necessary for locating empty regions to place poles.

```
sig PowerSupplyNetworkMap extends NetworkMap { }
sig PowerSupplyMetricMap extends MetricTaxicabMap { }
sig PowerSupplyMap extends Map { }
fact { PowerSupplyMap = PowerSupplyNetworkMap & PowerSupplyMetricMap }
```

In the power-supply map, many resources have point-type geometries (D0). These resources cannot occupy the same location.

```
fact { all m: Map | all o1, o2: m.O |
    (o1 + o2).G in D0 && o1 != o2 ⇒ m.Location[o1] != m.Location[o2]
}
```

Poles, consumers, cables, transformers, towers, transmission lines, and power plants partition the resource set.

```
part sig Pole, Consumer, Cable, Transformer, PowerPlant, Tower, TransmissionLine extends Resource { }
```

Each type of resource has some specific features. The feature that is used in all of the problems is the type of the resource. This information could have been found using the membership operation, determining to which set the resource belongs. However, to make it easier to access the type, redundant features were created capturing the type of the resource. The features partition the feature set.

```
part sig isPole, isConsumer, isCable, isTransformer, isTransmissionLine, isTower, isPowerPlant extends Feature { }
```

To make both redundant descriptions consistent with each other and to provide the geometry constraints of each type of resource, some facts are stated.

```
// all resources have exactly one feature
fact { all r: Resource | r.F = isPole || r.F = isConsumer || r.F = isCable || r.F = isTransformer
                || r.F = isTower || r.F = isTransmissionLine || r.F = isPowerPlant }
// all elements of Pole have feature "isPole" and have point-type geometry (D0)
fact { all p: Pole | p.F = isPole && p.G in D0 }
// all elements of Consumer have feature "isConsumer" and have point-type geometry (D0)
fact { all c: Consumer | c.F = isConsumer && c.G in D0 }
// all elements of Transformer have feature "isTransformer" and have point-type geometry (D0)
fact { all t: Transformer | t.F = isTransformer && t.G in D0 }
// all elements of Tower have feature "isTower" and have point-type geometry (D0)
fact { all t: Tower | t.F = isTower && t.G in D0 }
// all elements of PowerPlant have feature "isPowerPlant" and have point-type geometry (D0)
fact { all p: PowerPlant | p.F = isPowerPlant && p.G in D0 }
// all elements of Cable have feature "isCable" and have line-type geometry (D1)
fact { all c: Cable | c.F = isCable && c.G in D1 }
// all elements of TransmissionLine have feature "isTransmissionLine" and have line-type geometry (D1)
fact { all t: TransmissionLine | t.F = isTransmissionLine && t.G in D1 }
```

In the network map projection, nodes consist of poles, consumers, transformers, towers, and power plants. The edges consist of cables and transmission lines.

```
fact { all m: PowerSupplyNetworkMap | m.Net.nodes = (Pole + Consumer + Transformer + PowerPlant + Tower) & m.O }
fact { all m: PowerSupplyNetworkMap | m.Net.edges = (Cable + TransmissionLine) & m.O }
```

The cables connect poles, consumers, and transformers, while the transmission lines connect transformers, towers, and power plants.

```
fact { Cable.EndNodes in Pole + Consumer + Transformer }
fact { TransmissionLine.EndNodes in Transformer + PowerPlant + Tower }
```

The power supply network is acyclic and fully connected, and therefore it should be a tree. The tree function is defined in the network projection (see Section 7.2.4).

```
fact { all m: PowerSupplyNetworkMap | tree(m.Net) }
```

Each consumer must be connected to a single pole, because consumers should be the leaves of the tree.

```
fact { all m: PowerSupplyNetworkMap | all c: m.Net.nodes & Consumer |
        one m.Net.adj[c] && m.Net.adj[c] in Pole
}
```

In order to be able to analyze properties involving more than one map projection, it is necessary to relate the two projections. In this problem, it is sufficient to say that the location of all edges (network projection) must be equal to the location (metric, taxicab projection) of one of its end points.

```
fact { all m: PowerSupplyMap | all e: m.Net.edges |
        some p: e.EndNodes | m.Location[e] = m.Location[p]
}
```

The function below creates an instance of the power supply map with no crossing cables. This is used when analyzing properties. The function cross belongs to the metric, taxicab module (see Section 7.2.3).

```
fun no_crossing_cables(m: PowerSupplyMetricMap) {
    no c1, c2: m.O & Cable | {
        c1 != c2 &&
        cross(m.Location[c1.start],
              m.Location[c1.end],
              m.Location[c2.start],
              m.Location[c2.end])
    }
}
```

Formal Problem Frames description

The first frame to be formalized describes the problem of finding the nearest pole to a consumer. This frame is an instance of the *nearest* proximity problem (see Section 7.3.2). In the signature, the domains are set to be specific to the power-supply problem. The other two instances of the nearest proximity frame are very similar. Only the source and desired features are different.

```
sig FindNearestPoleFromConsumer extends NearestProximityFrame { }
{ domains.M in PowerSupplyMetricMap
  one domains.inputResource
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isPole
}
```

```
sig FindNearestPoleFromPole extends NearestProximityFrame { }
{ domains.M in PowerSupplyMetricMap
  one domains.inputResource
  domains.sourceFeature = isPole
  domains.desiredFeature = isPole
}
```

```
sig FindNearestTransformerFromConsumer extends NearestProximityFrame { }
{ domains.M in PowerSupplyMetricMap
  one domains.inputResource
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isTransformer
}
```

Another problem of interest is to determine which poles are within 30 meters from a new consumer. This problem is also an instance of the proximity problem frame, but this time specific for finding resources within a given distance.

```
sig FindPolesWithinDistance extends WithinDistanceProximityFrame { }
{ domains.M in PowerSupplyMetricMap
  one domains.inputResource
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isPole
}
```

The problem of determining suitable regions to add poles is an instance of the site selection frame. This instance is specific to the task of finding empty regions within a given distance from the new consumer and the new pole. See Section D.5 for a formal description of the site selection frame.

```
sig SelectSiteForPole extends EmptyRegionsWithinDistanceSiteSelectionFrame { }
{ domains.M in PowerSupplyMetricMap
  domains.sourceFeature = isConsumer + isPole
}
```

The last subproblem in the power-supply example is an instance of the routing problem. See Section D.7 for a formal description of the routing frame. The problem is to determine a route from a consumer to the transformer that connects this consumer to the high voltage network. The other two routing problems presented here are not part of the power-supply subproblems, but will be used to analyze properties of the network map description.

```
sig RouteToTransformer extends FindRouteFromGivenNodeToNodeWithFeatureRoutingFrame { }
{ domains.M in PowerSupplyNetworkMap
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isTransformer
}
```

```
sig RouteToPowerPlant extends FindRouteFromGivenNodeToNodeWithFeatureRoutingFrame { }
{ domains.M in PowerSupplyNetworkMap
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isPowerPlant
}
```

```
sig RouteToConsumer extends FindRouteFromGivenNodeToNodeWithFeatureRoutingFrame { }
{ domains.M in PowerSupplyNetworkMap
  domains.sourceFeature = isConsumer
  domains.desiredFeature = isConsumer
}
```

8.1.4 Analysis of the formal description

Analysis is done incrementally, by generating instances and checking properties. This section illustrates the different types of analysis for the power-supply example. Creating instances and verifying properties took from a few seconds to an hour in this case study.

Instances of the map domain and problem frames

The first analysis of the formal descriptions involve generating instances of the various models. The following is a sample instance of the power-supply map that contains two consumers, two poles, and no crossing cables. The instance was generated with scope 7.

```
fun test_no_crossing_cables( ) {
    some m: PowerSupplyMetricMap | no_crossing_cables(m) && two Consumer && two Pole
}
```

The same analysis should be done for the frames, to make sure that instances of the frames can be generated. Every instance of a frame is constrained by the specific machine specification included as a fact in the frame definition (see Section 7.3.2 for an example). This means that an instance can only be created if the machine specification can be satisfied. The following generates an instance of the routing problem frame where the map has at least one consumer and one transformer.

```
fun test_RouteToTransformer( ) {
    some f: RouteToTransformer |
        some f.domains.M.O & Consumer && some f.domains.M.O & Transformer
}
```

Properties of each individual frame

Section 7.4 discusses various properties that can be analyzed for each problem frame. The type of property to check depends on the problem at hand. For one of the proximity frames in the power-supply problem, the following property was analyzed:

```
assert singleton_proximity {
    all f: FindNearestPoleFromConsumer |
        some f.domains.proximalResource ⇒ one f.domains.proximalResource
}
```

This property states that there is only one pole that is the nearest pole to a consumer. The property was found to be false for a scope of 5, since two poles may be equidistant from the consumer, and therefore both qualify as the nearest pole.

For the routing frame[2] in this problem, the following properties were analyzed:

```
assert node_feature_includes_pole {
    all f: RouteToTransformer |
        some f.domains.route.nodes ⇒ some Pole & f.domains.route.nodes
}
```

[2]The formal definition of the routing frame is presented in Appendix D.7.

This property states that any route from a consumer to a transformer includes at least one pole. No counterexamples were found for a scope of 7.

```
assert node_features_includes_transformer {
    all f: RouteToPowerPlant |
        some f.domains.route.nodes ⇒ some Transformer & f.domains.route.nodes
}
```

This property states that any route from a consumer to a power plant includes at least one transformer. No counterexamples were found for a scope of 7.

```
assert edge_features_TwoCablesBetweenConsumers {
    all f: RouteToConsumer |
        some f.domains.route.nodes ⇒
            ! (no Cable & f.domains.route.edges || one Cable & f.domains.route.edges )
}
```

This property states that the route between any two consumers contains at least two cables. No counterexamples were found for a scope of 7.

Properties about the combination of frames

Some of the properties that were analyzed involve not only a single problem frame, but the effects of two or more frames used together. In order to analyze these properties, the frames were combined through their domains. For example, the input domains of one frame may be restricted to be the output domains of another frame. Combining problem frames by identifying common domains is only one way of combining frames. This case study does not involve other forms of combination. Other possibilities for combination in general are considered for future work in Section 9.5.

Figure 8.6 combines the NearestPoleFromConsumer and the FindPolesWithinDistance subproblems. It is concerned with the case where the nearest pole to the consumer is also within 30 meters. In this case, the consumer will be connected to this pole by a single cable. The assertion tests whether this cable will cross any other cables. It assumes that the map did not have crossing cables before the new consumer was connected.

A complete description would also require an instance of the editing frame to add the cable to the map. However, since the property does not depend on the method used to edit the map, the new consumer and cable are manually added to the map with a series of constraints.

```
assert CablesDontCrossWhenConnectingConsumer {
    all m1, m2: PowerSupplyMap, newcable: Cable, newconsumer: Consumer |
        // Properties of m1 - the original map
        // m1 does not have cables that cross
        no_crossing_cables(m1) &&
        // m1 only has poles, consumers, and cables
        m1.O in (Cable + Pole + Consumer) &&
        // The new cable is connected to the new consumer
        newcable.start = newconsumer &&
        // Set up m2 with m1 + the new consumer + the new cable
        m2.O = m1.O + newconsumer + newcable &&
        m2.Location = m1.Location + (newconsumer → m2.Location[newconsumer])
            + (newcable → m2.Location[newconsumer]) &&
        // The new consumer must be in a previously empty location
        no ~(m1.Location)[m2.Location[newconsumer]] &&
        m2.P = m1.P &&
        // Combine the two frames so that a nearest pole is within 30m
        (some p: FindNearestPoleFromConsumer.domains.proximalResource
                    & FindPolesWithinDistance.domains.proximalResource |
            // Connect the new consumer to this pole
            m2.Net.nodes = m1.Net.nodes + newconsumer &&
            m2.Net.edges = m1.Net.edges + newcable &&
            m2.Net.adj[newconsumer] = p &&
            m2.Net.adj[p] = m1.Net.adj[p] + newconsumer &&
            all n: Node - newconsumer - p | m2.Net.adj[n] = m1.Net.adj[n]
        )
        &&
        // Set the domains of the frames
        one FindNearestPoleFromConsumer && one FindPolesWithinDistance &&
        FindNearestPoleFromConsumer.domains.inputResource = newconsumer &&
        FindPolesWithinDistance.domains.inputResource = newconsumer &&
        FindNearestPoleFromConsumer.domains.M = m2 &&
        FindPolesWithinDistance.domains.M = m2
    ⇒
        no_crossing_cables(m2)
}
```

Figure 8.6: Assertion stating that when connecting a consumer to a network that has no crossing cables, the new cable will not cross any others

This property was found to be false for a scope of 7. An illustration of a counterexample is shown in Figure 8.7. Since the problem uses taxicab geometry as an abstraction, and when cables intersect in taxicab geometry, they do not necessarily intersect in Euclidean geometry, it is necessary to inspect the counterexamples found to see whether or not they apply to the Euclidean plane. In this case, a counterexample applies to both geometries, so the property is also false for the Euclidean plane.

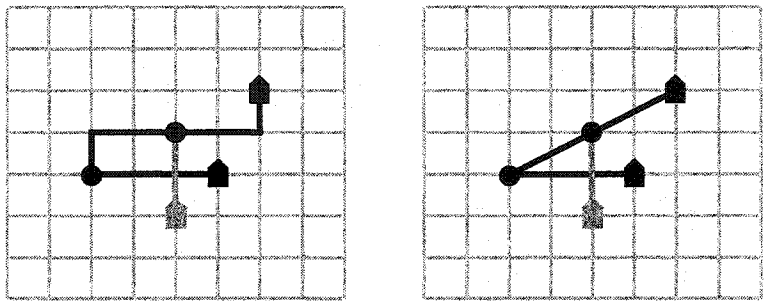


Figure 8.7: Counterexample of the assertion CablesDontCrossWhenConnectingConsumer (a) in the taxicab plane (b) in the Euclidean plane

The original power-supply problem states that when it is necessary to add a new pole to connect a consumer to the network, both the consumer and the new pole must be connected to the nearest poles. If the nearest poles can be determined beforehand, it may not be necessary for field work teams to perform unnecessary measurements. Therefore, two properties were analyzed. Let P_A be the nearest pole to the consumer, originally, and P_B be the newly added pole. P_B is placed in an empty region that is within 30m of the consumer and of P_A.

The first property that was analyzed is shown in Figure 8.8. The property combines the NearestPoleFromConsumer, NearestPoleFromPole and SelectSiteForPole problem frames. It checks whether the nearest pole to P_B is P_A.

```
assert SameNearestPole {
    // Map m1 is the original map, before adding the new pole. Map m2 is the map with the new pole.
    // For this property, the maps use only the metric taxicab projection.
    // No cables or connections are represented since they are not relevant.
    all m1, m2: PowerSupplyMetricMap, newpole: Pole, newconsumer: Consumer |
        // Properties of m1
        m1.O in (Pole + Consumer) &&
        newconsumer in m1.O &&
        newpole !in m1.O &&
        // The new consumer is placed more than 30m away from the nearest pole.
        OrdLT(SelectSiteForPole.dist,
            taxicabDistance(m1.Location[newconsumer],
            m1.Location[FindNearestPoleFromConsumer.domains.proximalResource]) ) &&
        // There is exactly one pole that is nearest to the consumer
        one FindNearestPoleFromConsumer.domains.proximalResource &&
        // Set up m2 with m1 + the new pole
        m2.P = m1.P &&
        m2.O  = m1.O + newpole &&
        (all r: m1.O | m2.Location[r] = m1.Location[r]) &&
        // The location of the new pole is given by the site selection frame
        m2.Location[newpole] in SelectSiteForPole.domains.outputRegion.P &&
        // Frames working with m1
        one FindNearestPoleFromConsumer &&
        one SelectSiteForPole &&
        FindNearestPoleFromConsumer.domains.inputResource = newconsumer &&
        SelectSiteForPole.domains.inputResource = newconsumer +
                                FindNearestPoleFromConsumer.domains.proximalResource &&
        FindNearestPoleFromConsumer.domains.M = m1 &&
        SelectSiteForPole.domains.M = m1 &&
        // Frames working with m2
        one FindNearestPoleFromPole &&
        FindNearestPoleFromPole.domains.inputResource = newpole &&
        FindNearestPoleFromPole.domains.M = m2
    ⇒
        // The set of nearest poles to the new pole contains the original pole that was nearest to the consumer
        FindNearestPoleFromConsumer.domains.proximalResource in
            FindNearestPoleFromPole.domains.proximalResource
}
```

Figure 8.8: Assertion SameNearestPole combining three problem frames

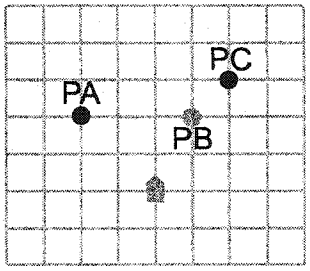


Figure 8.9: Counterexample of the assertion SameNearestPole

This property was found to be false for a scope of 7. An illustration of a counterexample is shown in Figure 8.9. The nearest pole to P_B is P_C and not P_A as stated by the assertion, no matter which of taxicab and Euclidean distance are taken.

The second property checks whether the nearest pole to the consumer after P_B is placed is in fact P_B. It is shown in Figure 8.10. The property combines two instances of the NearestPole-FromConsumer frame and one instance of the SelectSiteForPole frame. No counterexamples were found for a scope of 7.

The last property combines the Routing and NearestTransformerFromConsumer problem frames. Each of these frames uses a different map projection. The description of the property is shown in Figure 8.11. The property states that the first reachable transformer in the network from a consumer is the nearest transformer to the consumer. This property was shown to be false for a scope of 7.

When putting subproblems together to create more complex machines, first it is recommended to generate instances of the composition and check that the frame concerns hold for the composed machine specification and requirements.

8.1.5 Comparison between the problem frames approach and GeoOOA

GeoOOA claims to be a "domain-tailored, requirements-engineering method for the development of GIS-applications." It is the most well-known and discussed method found in the literature. This section compares GeoOOA as a requirements engineering method with the problem-oriented approach presented in this dissertation, in the light of the power supply network example.

```
assert NewPoleIsNearest {
    // Map m1 is the original map, before adding the new pole. Map m2 is the map with the new pole.
    // For this property, the maps use only the metric taxicab projection.
    // No cables or connections are represented since they are not relevant.
    all m1, m2: PowerSupplyMetricMap, newpole: Pole, newconsumer: Consumer,
    nearest1, nearest2: FindNearestPoleFromConsumer |
        // Properties of m1
        m1.O in (Pole + Consumer) &&
        newconsumer in m1.O &&
        newpole !in m1.O &&
        // The new consumer is placed more than 30m away from the nearest pole.
        OrdLT(SelectSiteForPole.dist,
            taxicabDistance(m1.Location[newconsumer],
            m1.Location[FindNearestPoleFromConsumer.domains.proximalResource]) ) &&
        one FindNearestPoleFromConsumer.domains.proximalResource &&
        // Set up m2 with m1 + the new pole
        m2.P = m1.P &&
        m2.O  = m1.O + newpole &&
        (all r: m1.O | m2.Location[r] = m1.Location[r]) &&
        // The location of the new pole is given by the site selection frame
        m2.Location[newpole] in SelectSiteForPole.domains.outputRegion.P &&
        // Frames working with m1
        one SelectSiteForPole &&
        nearest1.domains.inputResource = newconsumer &&
        SelectSiteForPole.domains.inputResource = newconsumer +
                            FindNearestPoleFromConsumer.domains.proximalResource &&
        nearest1.domains.M = m1 &&
        SelectSiteForPole.domains.M = m1 &&
        // Frames working with m2
        nearest2.domains.inputResource = newconsumer &&
        nearest2.domains.M = m2
    ⇒
        // The newly placed pole is the nearest pole to the consumer
        nearest2.domains.proximalResource = newpole
}
```

Figure 8.10: Assertion NewPoleIsNearest combining three problem frames

```
assert ConnectedTransformerIsNearest {
    all m: PowerSupplyMap, c: Consumer, t: Transformer |

        // Properties of m
        m.O in (Cable + Pole + Consumer + Transformer + Tower + TransmissionLine)

        // Frames working with m
        && one FindNearestTransformerFromConsumer
        && one RouteToTransformer
        && FindNearestTransformerFromConsumer.domains.inputResource = c
        && FindNearestTransformerFromConsumer.domains.M = m

        && RouteToTransformer.domains.nodeResource = c
        && RouteToTransformer.domains.M = m

        // "t" is the transformer in the route that was found
        && t = RouteToTransformer.domains.route.nodes & Transformer

    ⇒
        // "t" is also the nearest transformer to the consumer
        t in FindNearestTransformerFromConsumer.domains.proximalResource
}
```

Figure 8.11: Assertion ConnectedTransformerIsNearest combining two problem frames with two different map projections

Map domain description

The GeoOOA notation was chosen to describe the map domain for the power-supply example, as it was already used for this purpose in [KPS97]. However, after creating the formal description of the map, many flaws in the original GeoOOA description shown in 8.2 were discovered:

1. It is not clear whether or not the power supply network can or cannot have cycles;
2. It is not clear whether or not consumers must be leaves in the graph;
3. Although the description shows the two kinds of network (low-voltage and high-voltage) and the types of the nodes and edges of each network, it is not clear what kind of edge (cable or transmission line) should be used to connect two transformers.

No matter what decisions are made regarding these issues, GeoOOA is not expressive enough to describe them. A formal description is an invaluable extension to the GeoOOA diagrams, as it is able to capture the answers to the above issues and many others without ambiguities. Analysis

of the formal map description is also important to discover bits of the description that were overlooked. For example, the formal map description of the power-supply problem in Section 8.1.3 is ambiguous with respect to item 3. This fact was only discovered after analysis of the map.

Problem descriptions

The kinds of problems the GeoOOA map claims to address are represented as services or methods offered by each of the specialized classes. For example, the standard services of a *network* class include test on connectivity or cycles, traversals, detection of sources and sinks, and determination of paths. These are generic services provided by the class. They do not refer to any specific problem domain in which the class may be used. GeoOOA offers no notation or form of analysis of the problems that a described map will be used to address. It may be adequate for describing map-domain properties, but not for requirements analysis, in that it is not concerned with the behaviour of applications that use the map, or the effects of these applications on the real world. In the case of the GeoOOA power-supply-network description, it is not possible to predict the problems that the map will be used to address.

Problem frames, on the other hand, are concerned not only with describing the domain properties, but also what elements of these domains will be shared with the machine — the shared phenomena, how the machine interacts with elements of the domains — the specification, and what effects the machine will produce over the domains — the requirements. It is important to perform this problem analysis in conjunction with the domain-property analysis, because the problem analysis is the driving force behind determining the important elements in the map domain.

The two approaches complement each other. GeoOOA provides a graphic notation for informal map description that should be supported by clarifying text — formal or informal. The problem-frames approach provides problem analysis that should be done along with the map description.

8.2 Algonquin Park fire prevention

The second case study is spread throughout the dissertation, as the Algonquin Park fire prevention problem was used to illustrate the problem-frames approach. The informal description of each subproblem was developed in detail in Chapters 5, and 6.

This section concentrates on the formal descriptions of a few subproblems presented previously. The goal is to illustrate topological properties of maps that use the nine-intersection projection.

8.2.1 Formal map description

The Algonquin Park map is a topological map that uses the nine-intersection model, which is described in Section 7.2.4 and Appendix D.3.

```
sig AlgonquinParkMap extends NineIntersectionMap { }
```

The set of resources in the map is partitioned into the following subsets:

```
part sig Transmitter, FireStation, CoverageRegion, ParkBoundary, Road extends Resource { }
```

The following facts determine the geometry of each kind of resource:

```
fact { (Transmitter + FireStation).G in D0 }
fact { Road.G in D1 }
fact { (CoverageRegion + ParkBoundary).G in D2 }
```

These are the features that distinguish each resource. The feature information is redundant since it is already found in the type of the resource. However, it makes the problem descriptions more readable. Other features could also have been defined. The features partition the feature set.

```
part sig isTransmitter, isFireStation, isCoverageRegion, isParkBoundary, isRoad
  extends Feature
```

In this example, each resource has only one feature, which specifies its type.

```
fact { all t: Transmitter | t.F = isTransmitter }
fact { all f: FireStation | f.F = isFireStation }
fact { all c: CoverageRegion | c.F = isCoverageRegion }
fact { all p: ParkBoundary | p.F = isParkBoundary }
fact { all r: Road | r.F = isRoad }
```

The following facts enforce the existence of a feature whenever the corresponding resource exists. This is done so that instances with empty feature sets are not created.

```
fact { some Transmitter ⇒ one isTransmitter }
fact { some FireStation ⇒ one isFireStation }
fact { some CoverageRegion ⇒ one isCoverageRegion }
fact { some ParkBoundary ⇒ one isParkBoundary }
fact { some Road ⇒ one isRoad }
```

Two zero-dimensional objects (D0) cannot be in the same location.

```
fact { all m: Map | all o1, o2: m.O |
        (o1 + o2).G in D0 && o1 != o2 ⇒ m.Location[o1] != m.Location[o2]
}
```

The following facts enforce the cardinality of some resources on the map and the map itself.

```
// There is only one instance of the map
fact { one AlgonquinParkMap }
// The park has a single boundary
fact { one ParkBoundary }
// There are some coverage regions in the park
fact { some CoverageRegion }
```

The following facts enforce topological relations between resources on the map.

```
// Every transmitter is inside the park boundary
fact { all o: Transmitter | ObjectInRegion(o.G, ParkBoundary.G) }
// Every fire station is in the closure of a coverage region
fact { all o: FireStation | some r: CoverageRegion | ObjectInRegion(o.G, r.G) }
// Every coverage region has one and only one fire station inside it
fact { all r: CoverageRegion | one f: FireStation | ObjectInRegion(f.G, r.G) }
// Coverage Regions form a partition of the park boundary
fact { IsBoundingRegion(CoverageRegion.G, ParkBoundary.G) }
```

The last fact does not make the coverage regions a true partition of the park boundary, which is impossible to state in the nine-intersection model. However, it does represent many of the same properties of a partition, such as the fact that the parts are disjoint and there is no resource that intersects with the interior of the park and does not intersect with some coverage region.

8.2.2 Formal description of problem frames

In this subsection, three instances of the resource-topology problem frame and one instance of the spatial-definition problem frame are formalized for the Algonquin Park problem. The first instance of the resource topology finds the coverage region where a transmitter in alert state is located. It is an instance of the FindEnclosingRegion specific problem frame. The second instance finds the fire station that is responsible for answering requests in that coverage region. It is an instance of the FindResourcesInRegion specific problem frame. These problems are informally described in Section 6.6.

```
sig FindCoverageRegion extends FindEnclosingRegion { }
{ domains.M = AlgonquinParkMap
  domains.sourceFeature = isTransmitter
  domains.desiredFeature = isCoverageRegion
}

sig FindFireStation extends FindResourcesInRegion { }
{ domains.M = AlgonquinParkMap
  domains.sourceFeature = isCoverageRegion
  domains.desiredFeature = isFireStation
}
```

The third instance of the resource topology frame determines which coverage regions are crossed by Road 60. It is an instance of the FindCrossedRegions specific problem frame.

```
sig FindCrossedCoverageRegions extends FindCrossedRegions { }
{ domains.M = AlgonquinParkMap
  domains.sourceFeature = isRoad
  domains.desiredFeature = isCoverageRegion
}
```

The last problem is an instance of spatial definition problems. It defines a new region — "Road 60 region" — as the spatial union of all coverage regions crossed by Road 60. It is an instance of the TopologicRegionUnionFrame specific problem frame. This problem is informally described in Section 6.9.

```
sig FindRoad60Region extends TopologicRegionUnionFrame {
    domains.M = AlgonquinParkMap
}
```

8.2.3 Formal analysis

Instances of the map domain and problem frames

The following generates an instance of the Algonquin Park map with at least one transmitter and exactly two coverage regions. The instance was generated for a scope of 4.

```
fun AlgonquinParkMapInstance() {
    some AlgonquinParkMap.O & Transmitter
    two  AlgonquinParkMap.O & CoverageRegion
}
```

The following function generates instances of two resource topology frames. The output of the first frame is the input of the second frame. These two problem frames are combined in order to determine which fire station is responsible for answering a transmitter in alert. An instance of this function was generated for a scope of 5.

```
fun FindFireStationOfTransmitter(f1: FindCoverageRegion, f2: FindFireStation, t: Transmitter) {
    f1.domains.inputResource = t
    f2.domains.inputResource = f1.domains.outputResource
}
```

Finally, the following function combines a resource-topology frame with a spatial-definition frame. The output of the resource-topology frame is the input to the spatial-definition frame. The frames are combined to determine the region that is crossed by Road 60. An instance of this function was generated for a scope of 5.

```
fun CreateRoad60Region(f1: FindCrossedCoverageRegions, f2: FindRoad60Region, road60: Road) {
    f1.domains.inputResource = road60
    all res: f1.domains.outputResource |
        one reg: f2.domains.inputRegion | reg.G = res.G
    no reg: f2.domains.inputRegion | reg.G !in f1.domains.outputResource.G
}
```

Properties of each individual frame

One of the properties that can be checked for spatial-definition problems mentioned in Section 7.4 was the *containment* property. The following property checks whether all the inputs to the "find Road 60 region" problem are contained inside the new region (the output of the problem). This property is expected to be true. No counterexamples were found for a scope of 7.

```
assert Road60Region_containment {
    all f: FindRoad60Region | all r: f.domains.inputRegion.G |
        inside_closure(r, f.domains.outputRegion.G)
}
```

Properties about the combination of frames

The first analysis involves the combination of two instances of the topology frame. The function FindFireStationOfTransmitter combining the two instances was presented previously in this section. The property that needs to be analyzed checks whether there is one and only one fire station that is in the same coverage region as each transmitter. The following description states the property:

```
assert OneFireStation {
    all f1: FindCoverageRegion, f2: FindFireStation, t: Transmitter |
        FindFireStationOfTransmitter(f1, f2, t) ⇒ one f2.domains.outputResource & FireStation
}
```

It seems correct to assume that there is only one fire station in the same coverage region as the transmitter, given that the coverage regions form a partition of the park boundary. However, this property turns out to be false (for a scope of 6) if the transmitter is on the boundary between two coverage regions. In this case, two fire stations are responsible for providing emergency response to that transmitter. Additional constraints should be added to restrict the location of transmitters to the interior of coverage regions if it is desirable that only one fire station covers each transmitter.

The second analysis was made to verify that the newly created region for Road 60 was really crossed by the road. If the Road 60 region is defined as the spatial union of the coverage regions crossed by Road 60, is it the case that the newly created region is also crossed by Road 60?

The function CreateRoad60Region combining the resource topology instance and the spatial definition instance was presented previously in this section.

```
assert Road60CrossesDefinedRegion {
    all f1: FindCrossedCoverageRegions, f2: FindRoad60Region, road60: Road |
        CreateRoad60Region(f1, f2, road60) ⇒ LineCrossesRegion(road60.G, f2.domains.outputRegion.G)
}
```

No counterexamples to this property were found for a scope of 6.

8.2.4 Summary

One of the important lessons learned in this study was that the concept of spatial partition is different from set partitions. That is, in a spatial partition the parts intersect at their boundaries. This is a property that may be overlooked in geographic applications. The formal analysis brought attention to this fact and showed that it might influence the outcome of some problems. The formal analysis operations performed in this case study using the Alloy tool took from a few seconds to half an hour.

Chapter 9

Conclusions

9.1 Summary

This dissertation introduces a problem-oriented approach to description and analysis of geographic requirements. A representative set of geographic subproblems provides the basis for the problem-oriented approach. The approach consists of informal description and analysis based on problem frames that were extended to capture geographic subproblems. This led to the definition of a formal framework that allows the formal representation and analysis of geographic problems and requirements.

The informal description and analysis is based on a set of classes of geographic subproblems that are presented in Chapter 3. These subproblems are typical of geographic applications. Each subproblem is described as a problem frame. A problem frame shows the problem domains, shared phenomena, requirements, and machine specification of each general problem. An example of each subproblem was developed to illustrate how real-world problems are fitted to problem frames.

Two major categories of problems were addressed in this work. The first category deals with creating and manipulating maps. The problems in the first category are: data capture, data conversion, map visualization, map editing, and map selection. These problems are described in Chapter 5. The second category deals with using maps to address real-world problems. The problems in this category are: measurement, classification, resource location, resource inventory, proximity, resource topology, routing, site selection, and spatial definition. These problems are described in Chapter 6. All of the problems in both categories refer to a map domain. Common map domain descriptions and phenomena are presented in Chapter 4.

The formal framework encompasses three parts: descriptions of geographic problem frames,

descriptions of specialized requirements and specifications, and projections of map domains. The Alloy language and tool were used for the description and analysis of properties. Four kinds of analysis can be performed. Initially, a specific map domain can be analyzed to detect inconsistencies. The second analysis involves showing that the requirements of a specific problem are satisfied given a specific machine specification and domain properties. Frame-specific properties and properties about a combination of frames can also be analyzed. The formal framework is described in Chapter 7.

Finally, the validity of the approach is demonstrated through two case studies, described in Chapter 8. The first case study comes from the utilities industry and illustrates problems of connecting a new consumer to an existing power supply network. The problem consists of several subproblems, and the map is described using two projections. Both the informal and formal description and analysis were performed. Performing the formal analysis revealed gaps in the understanding of the power supply network. Both kinds of analysis were useful in determining which map phenomena were required in the map domain descriptions in order to describe the problems.

The second case study is spread throughout the dissertation and explores subproblems in fire detection and prevention at Algonquin Park. The informal analysis of this problem was used to demonstrate and validate each problem frame. The formal analysis revealed that an expected property of one of the subproblems was false due to the difference between spatial partitions and set partitions.

9.2 Scope of the work

The geographic problem frames encompass a representative set of geographic problems. However, some geographic applications may require concepts not included in this work. For example, a *navigational* application is a particular type of geographic application that deals with the concept of *direction*. This concept was not explored in this dissertation, so navigational problems are not well suited to the geographic problem frames. The concept of *time* may also be a variable when considering geographic requirements. It is mostly required to determine history, patterns or trends in the changing environment. Temporal problems associated with geography were not explored in this work.

The formal projections of the map domain used in this dissertation are abstractions that approximate the real world. As such, they are imperfect when it comes to representing continuous space and non-binary topological relationships. These imperfections happen because the selected approaches to automated formal analysis have space and time constraints and deal with discrete

models. These constraints may lead to the state explosion problem [CK96]. The use of abstractions is a way to deal with this problem.

The Alloy tool that was used for analysis was appropriate for handling the complexity of the map domain but is limited in the types of analysis that are possible. For example, it is unable to perform reachability analysis [JSS01].

9.3 Evaluation of the problem-oriented approach

Problem analysis is the underlying technique for the approach presented in this dissertation. The motivation for using this technique is the fact that existing requirements approaches for geographic applications pay minimal or no attention to the geographic problems that an application needs to solve. Instead, these approaches concentrate on the application's implementation and design issues.

After developing the problem-oriented approach for describing and analyzing geographic requirements and having applied the approach to two case studies, the following conclusions can be drawn:

- The classes of geographic problems have contributed to the description of subproblems. They provide a specific vocabulary for describing the subproblems and guide the problem decomposition.
- Fitting a problem into a problem frame helps the analyst decide whether or not the right frame was selected.
- Framing a problem helps the analyst not to forget or overlook any of its major parts
- The use of geographic problem frames led to questions about the subproblems in the case studies. For example, the power supply case study started with a detailed map domain description and a general problem description. When decomposing the problem of connecting a consumer to the network, it was necessary to know whether or not a consumer had to be a leaf in the network. This question could not be answered by examining the original map description.

9.3.1 Similarities among problem frames

All problems in Chapter 6 are specialized versions of the static information system frame [Jac99]. Therefore, these frames resemble each other. While similar at first glance, the frames differ in the

enquiries domain which contains the initial references to the information, the output domain which is the information being reported, the characteristics of the model which is the map that is required to solve each problem, the phenomena shared with the machine, and the information relation provided in the requirement. The problems are quite distinct from each other. The differences are not apparent in the diagrammatic representation, but in the verbal descriptions and in the formalization.

For example, the frame diagrams do not capture the differences between each map, which is information that has a major impact on the frames. Perhaps a refinement of the diagrammatic notation would enhance its value and make the frames more distinct from each other. Since all maps are lexical domains, the annotation in the diagram does not add any information. An extra letter or symbol could further classify the map domain as field or object, isolated or connected, with absolute or relative locations. The resource domain appears frequently in the geographic frames. This domain could be annotated with the geometric type — such as point, line, or region — of the resource, when appropriate. If the input and output resource domains are of the same kind, an annotation could state whether the output is a subset of the input, or whether the two sets are disjoint, or other relationships.

9.3.2 Issues revisited

Section 1.3.1 outlined many issues related to the lack of problem analysis when describing the requirements of geographic applications. The following list shows how the analysis addresses each of these issues.

- **understanding the problem before jumping to a solution**

  The analyst often has unstated assumptions about a problem that can be revealed to be incorrect through analysis. For example, in the Algonquin Park case study, the fact that each transmitter is not necessarily covered by only one fire station was a property of the map that violated an unstated assumption. Likewise, in the power supply network case study, the fact that adding a consumer to the network following the stated rules could lead to a network with crossing cables also helped the analyst to understand better the characteristics of the network before proposing a solution.

- **distinguishing the problem from other concerns and symptoms surrounding it**

  The problem frame approach places bounds on the problem domains. The requirements of each subproblem can only refer to phenomena of the problem domains that are parts

of the problem frame. Any other information is irrelevant or extraneous to that particular subproblem. For example, when capturing data for the Algonquin Park map, For example, when creating the Algonquin Park map, it is necessary to record the locations of physical entities like transmitters and fire stations, as well as of logical entities such as the boundary of the coverage regions. Capturing the location of the physical entities involves a data capture problem, which is not concerned with the logical entities. The data capture problem frame makes this fact explicit, since the captured entities must have a correspondence with entities from the real world.

- **defining requirements clearly and achieving good prior understanding of what the system will deliver if it works**

  The separation between machine specification and requirements leads the analyst to think separately about:

  - the effects that the machine is to have over the problem domains; and
  - how the machine interacts with the shared phenomena of the problem domains to achieve these effects.

  This separation guides the informal description, and is especially useful during formal description. In Alloy, a machine specification shows how relations are traversed to reach the desired results from the inputs. In contrast, an Alloy description of requirements describes the properties of the result using predicates. Having two descriptions allows the analysis of whether or not the specification satisfies the requirement, which is not possible with a single description.

- **achieving adequate understanding of domain properties**

  The case studies show several examples of analysis oriented towards understanding domain properties. These properties are related to the map structure and to the domains involved in a problem frame. They were outlined in section 7.4. For instance, in the power supply network case study, the following domain properties were analyzed: any route from a consumer to a transformer includes at least one pole; any route from a consumer to a power plant includes at least one transformer; there can be more than one pole that is the nearest pole to a consumer; and connecting a consumer to a network with no crossing cables may result in crossed cables. In the same case study, the formal description revealed several questions which could be not be answered with the original map domain description: can

the network have cycles? must consumers be leaves in the network? what kind of line connects two transformers?

- **collecting data in the appropriate level of detail given the likely uses of the system**

  Since the approach is problem-oriented, the data analysis — domain properties of the map — is done in the light of problem analysis. As a consequence, only the data that is necessary for a particular problem needs to be collected. The approach enforces this through the separation between machine specifications, domain properties, and requirements, identifying the necessary phenomena of each problem domain to address the problem. For example, in the Algonquin Park problem, information about bodies of water is required in only one subproblem outlined in section 6.4. Analysis of this problem reveals that bodies of water must be represented as region-type entities, and must include the average depth as an attribute. No other information about bodies of water is required.

9.3.3 Alternative approaches

An alternative approach to problem analysis such as goal-oriented analysis could have been extended and specialized to geography in the same fashion as was the problem-frames approach. For example, in the KAOS method [vLLD98] a goal is refined into smaller goals until a set of requirements are reached that, when combined, satisfy the original goal. Classes of goals exist much in the same way as the classes of problems in the problem frames approach. However, specializing problems frames was considered to be more appropriate given that they address the issues discussed in the previous section. KAOS uses formal and informal descriptions, but the formal descriptions are based on temporal logic. Although temporal issues relate well to goals, in this work only geographic problems of a static nature were considered, and the time dimension was not relevant.

The approach from this dissertation provides a clear separation between the informal and formal parts of the problem analysis and description. This separation is beneficial since analysis can be restricted to the informal phase. In contrast, approaches such as Software Cost Reduction (SCR) [HJL96] rely on the use of formal techniques. The informality of the problem frame approach is useful for communication among analysts, users, and clients.

Scenario- and use-case-based approaches to requirements engineering could be used to complement the problem frames approach. Each geographic problem frame that can be described in terms of user interactions could have an associated use case. From this case, various scenarios could be described to add to the richness of the descriptions.

9.4 Evaluation of the formal analysis

Some of the issues revealed through formal analysis could have been discovered by informal analysis. For example, whether cables cross or whether a transmitter is covered by two fire stations. However, humans are fallible and many of these properties are not apparent. Informal property analysis often simply confirms initial assumptions and expectations. Formal analysis allows these questions to be answered in a more reliable and automated fashion.

The following are the types of properties that can be analyzed using the approach:

- **structural properties of the map**

  These properties involve, for example, absolute positioning, metric relations, or topological relations. The power supply map could be analyzed to determine whether or not all consumers are always connected to the network.

- **properties of a problem frame**

  These properties involve the possible outputs generated by the machine for some given inputs. For example, given a new consumer, is it always the case that the output of the "nearest" problem is a single pole? In other words, can there be more than one nearest pole to this new consumer? Section 7.4 outlines properties that can be analyzed for each problem frame.

- **properties of a combination of problem frames**

  Most real-world problems involve multiple problem frames. The approach enables analysis of properties that result from the combination of two or more frames that are concerned with the same problem domains. The power supply case study includes properties of this kind. For instance, when connecting a new consumer to the network, can cables cross? This property involves two instances of the proximity problem frame.

The Alloy notation is appropriate for describing structural relationships [JSS01] and as such is suitable for describing the geographic problem frames and the properties listed previously. The problems addressed by the geographic frames are static, not usually described as state machines. Concurrency is not an issue, nor are safety- or liveness-type properties. Therefore, state-machine-based model checkers were not considered as suitable for this work.

There are different types of properties that cannot be analyzed in the formal descriptions. Some are related to the choice of Alloy as the analysis tool:

- **Reachability analysis**

  Given an initial map, a final map, and a transformation over the map, it is not possible to determine whether it is possible to obtain the final map starting from the initial map and applying the transformation multiple times.

- **Recursion and iteration**

  When adding a new consumer to the power supply network, it may be necessary to add multiple poles to span the distance between the consumer and the network. Since Alloy is a declarative language, this problem cannot be described as an iteration. The problem could have been described declaratively using recursion. However, Alloy does not support recursion. In order to perform analysis of the problem of consumer a consumer, an assumption was made that at most one new pole would have to be added to the network.

- **Computations involving Euclidean distance**

  Model checkers in general are based on finite types. As such, they cannot handle real numbers or the continuous plane required for computations involving Euclidean geometry.

The shortcomings of formal map projections also limit the types of properties that can be analyzed:

- determining the topology of a map given the coordinates of the objects in the map
- determining the boundary of a spatial union in terms of topology or absolute coordinates
- combining map projections; for example, nine-intersection cannot be combined with any Cartesian projection

9.5 Future work

9.5.1 Notations for informal map descriptions

Existing notations for informal map description have some deficiencies in terms of their expressiveness. Studies have been performed comparing these notations, considering their suitability to describe database conceptual models [FCTJ01]. A similar study should be performed comparing

these techniques with respect to their applicability to describe map domains in the light of defining requirements.

9.5.2 A method for informal problem description and analysis

This work provides the artifacts to allow the description and analysis of geographic problems. However, it does not provide a method to guide the process of informal analysis. A method is needed that clearly defines the steps for producing and analyzing the requirements of a geographic application using the geographic problem frames. Such a method should account for the fact that map domain analysis and problem analysis should complement each other, the results of one influencing the description of the other.

9.5.3 Formal abstractions of map projections

The formal map projections presented in this dissertation included abstractions such as taxicab geometry and the nine-intersection model. Each of these abstractions has its applicabilities and limitations, which restrict the kinds of analysis that can be performed. Research is necessary to find other map abstractions that are suitable for use in discrete model checkers and model finders. For example, none of the provided abstractions are suitable for defining the boundary of a spatial union.

9.5.4 Composition of geographic problem frames

Decomposing complex real-world problems into subproblems allows for concentrating on and understanding one problem at a time. However, the ultimate goal is to build one machine that satisfies the requirements of all subproblems. Michael Jackson has studied the difficulties that arise when composing these subproblems [Jac00b]. A similar study should be performed with respect to geographic problem frames. Particular attention is necessary with respect to the matter of merging maps represented in different projections. It would also be interesting to evaluate the feasibility of recent implementation approaches such as aspect-oriented programming [KLM+97] as a platform for modularizing the subproblems at the implementation level, and composing the different map projections.

9.5.5 Requirements through the software lifecycle

The requirements of a geographic application will lead to the phases of design, implementation, and testing. It is necessary to study how geographic problem frames will impact other stages of the lifecycle. For example, geographic problem frames may give insights into designing appropriate software architectures and user interfaces.

Software architectures for geographic problem frames

The relationship between software requirements and software architectures is currently being explored [Nus01] [HJL+02]. In terms of geographic applications, the implementation platforms are evolving from monolithic systems, e.g., early GIS systems to component-based platforms. It would be interesting to study what types of software architectures are more suitable for mapping from each geographic problem frame.

User interfaces for geographic applications

Designing end-user interfaces for geographic applications is not a simple task, since applications often require expertise in the fields of geography and/or cartography. As geographic applications become more accessible to end users, e.g., web-based applications like MapQuest [Map96] or the Region of Waterloo Information Utility [Map00, CMTG98], the user interfaces must become more natural and require less expertise. Since geographic problem frames provide a description of a requirement referring to phenomena that may be observable by end users, perhaps they would also be a useful technique from which to base the design of user interfaces.

Appendix A

The Video Cassette Recorder problem

In this appendix the Video Cassette Recorder (VCR) problem is explored to illustrate the problem frames approach and underlying concepts. This problem was chosen for this purpose since it refers to concepts that are common knowledge and easy to understand.

Informal description of the VCR problem

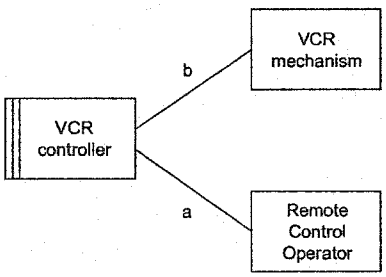
A simple VCR integrated with a TV accepts infrared signals coming from a user through a remote control. The signals are received through the VCR's infrared port which is connected to a computer. A software system for this computer is needed to process the infrared incoming signals and dispatch the appropriate signals to control the mechanism of the VCR. The VCR's remote control has the following buttons: *play, stop, rewind, fast-forward*, and *pause*. The VCR is always on and its mechanisms recognize signals to: start the motor that makes the heads and rollers spin at normal speed; start the motor at fast speed; stop the motor; engage and disengage the head so it reads from the tape; and set the direction of tape movement (forward or backward).

The *play* button should start the motor on normal speed and engage the head to read from the tape. If the *fast-forward* button is pressed while playing the tape, the rotation speed must be increased. *Rewind* works similarly but also requires a change in the direction of rotation. *Pause* should cause the motor to stop without disengaging the head. *Stop* should disengage the head from the tape and stop the motor. A *rewind* or *fast-forward* button pressed while the motor is stopped should cause the tape to advance or return without showing any image.

Besides the signal lines that connect the computer to the VCR mechanism there is also a status line that the VCR uses to share signals with the machine informing whether the VCR is in pausing mode or not. This is necessary since the result of the *pause* button on the VCR depends on the current status of the VCR. If the user presses *play* followed by *pause*, the image should be frozen. However, by pressing *pause* a second time, the VCR should return to playing mode.

A.1 Context and Problem diagrams

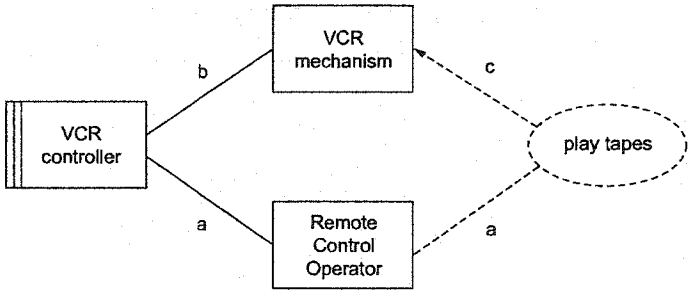
Figure A.1 presents a context diagram for the VCR problem.



a: Play, Rewind, Fast-forward, Pause, Stop
b: SetMOn, SetMFast, SetMOff, EngHead,
DisHead, SetDirFor, SetDirBack,
Pausing

Figure A.1: The VCR context diagram

Figure A.2 shows the VCR problem diagram.



a: OP! {Play, Rewind, Fast-forward, Pause, Stop}
b: VC! {SetMOn, SetMFast, SetMOff, EngHead,
DisHead, SetDirFor, SetDirBack}
VM! {pausing}
c: VM! {Playing, Rewinding, FForwarding,
Rew-image, FF-image, Pausing, Stopped}

Figure A.2: The VCR problem diagram

A.2 The VCR mechanism domain description

The VCR that needs to be controlled is a simple device. It is composed of a variety of mobile rollers and rotating heads as shown in Figure A.3 [Bra01].

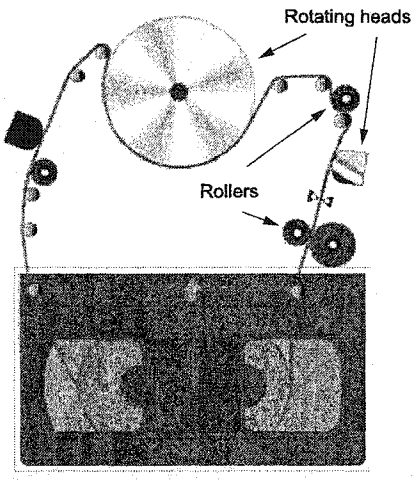


Figure A.3: The VCR mechanism

When the heads are engaged, a long piece of tape is extracted from the cassette. The extracted tape is wrapped around a variety of rollers and heads in order to play the tape. The heads touch the surface of the tape and read the recorded magnetic signals. Disengaging of the head causes the tape to return to its original position inside the cassette.

When the motor is activated, it causes the rollers to spin and pulls the tape in one direction. If the head is engaged, activating the motor allows the head to touch different parts of the tape sequentially, and to read the signals from the tape. If the head is disengaged, the tape is pulled in one direction but no signal is read from the tape. When the motor stops, the rollers cease to spin and the tape remains still. In order to simplify the problem, the end of tape sensor is not considered part of the VCR mechanism.

The VCR domain responds to certain shared phenomena originated by the machine by causing other phenomena to happen in an orderly manner. The VCR is a *causal* domain.

There are different types of phenomena. In this case, the phenomena shared between the VCR and the machine are *events*. These events cause changes in the private phenomena of the VCR. The private phenomena of the VCR are states of the mechanism. They are not shared with any other domain. The events shared with the machine may cause a change of state in the VCR.

The VCR domain has the following private state phenomena:

- MOn means that the motor is activated in normal speed. MFast means that the motor is activated in fast speed. MOff is defined as:
  $\mathsf{MOff} \triangleq \neg\mathsf{MOn} \wedge \neg\mathsf{MFast}$
  The three states are mutually exclusive, and this is expressed by the above definition and these two other properties:
  $\mathsf{MOn} \leftrightarrow \neg\mathsf{MFast} \wedge \neg\mathsf{MOff}$
  $\mathsf{MFast} \leftrightarrow \neg\mathsf{MOn} \wedge \neg\mathsf{MOff}$

- HOn means that the heads are touching the tape. HOff is defined as the negation of HOn:
  $\mathsf{HOff} \triangleq \neg\mathsf{HOn}$

- DFor means that the tape is moving in the forward direction. DBack is defined as the negation of DFor:
  $\mathsf{DBack} \triangleq \neg\mathsf{DFor}$.

The VCR mechanism shares the following events with the machine as shown in the problem diagram in Figure A.2: SetMOn, SetMFast, SetMOff, EngHead, DisHead, SetDirFor, and SetDirBack.

A state transition diagram is an appropriate notation to describe the behaviour of causal domains because it shows the relationship between events and states. Each round box represents a state. The upper part of the box identifies the state and is marked with the phenomena that hold in that state. The middle part of the box contains events that can occur but do not cause a transition to another state. The lower part of the box contains events that lead the VCR mechanism to an unknown state.

The machine should not generate commands that will put the VCR in an unknown state. However, this is part of the specification of the machine, and right now the concern of the description regards the domain properties.

The domain properties are essential to argue that the specification of the machine satisfies the requirements.

Figure A.4 shows the state machine diagram capturing the domain properties of the VCR. This diagram covers one uninterrupted power episode. When the VCR is turned on, the mechanism begins at the initial state.

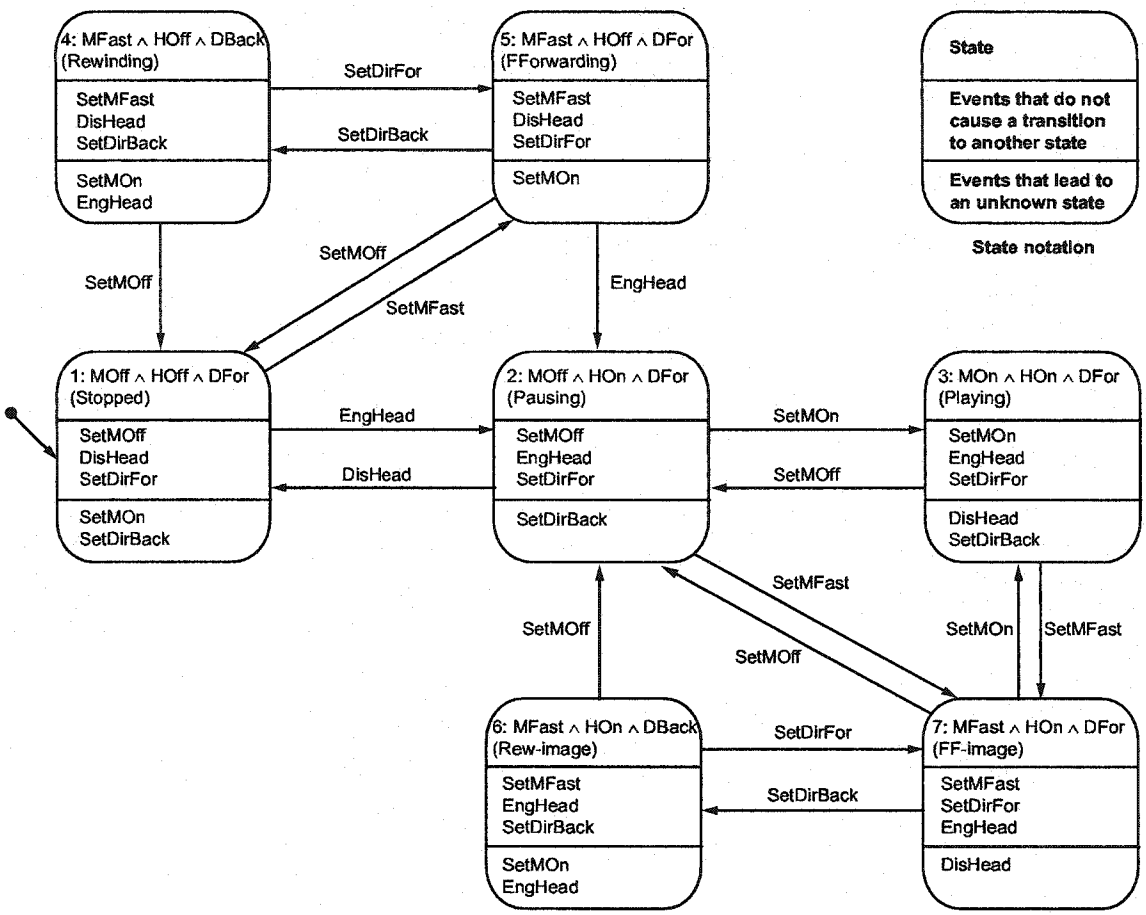


Figure A.4: The VCR mechanism: domain properties

As seen in the VCR problem diagram in Figure A.2, the requirements refer to and constrain phenomena from the VCR mechanism. These phenomena are: Playing, Rewinding, FForwarding, Rew-image, FF-image, Pausing, Stopped. They are defined as follows in terms of the private state phenomena of the domain presented earlier:

- Playing $\triangleq$ MOn $\wedge$ HOn $\wedge$ DFor
- Rewinding $\triangleq$ MFast $\wedge$ HOff $\wedge$ DBack
- FForwarding $\triangleq$ MFast $\wedge$ HOff $\wedge$ DFor
- Rew-image $\triangleq$ MFast $\wedge$ HOff $\wedge$ DBack

- FF-image $\triangleq$ MFast $\wedge$ HOn $\wedge$ DFor
- Pausing $\triangleq$ MOff $\wedge$ HOn $\wedge$ DFor
- Stopped $\triangleq$ MOff $\wedge$ HOff $\wedge$ DFor

The Pausing phenomena shared with the machine is the same phenomena in the requirement reference.

This completes the domain properties description for the VCR mechanism.

A.3 The remote control operator domain description

The operator of the remote control is instructed to press the remote control buttons in a certain order to produce the desired effects. However, there is no guarantee that the user will press the buttons in the suggested order. This domain is not causal unlike the VCR domain. The operator does not need external stimulus to press the buttons. The operator is autonomously active and is called a *biddable* domain.

The only set of phenomena that is relevant to the problem description in this domain are the shared phenomena with the machine. These phenomena are classified as events and are described as the action of pressing one of the following buttons: Play, Rewind, Fast-forward, Pause, Stop. Note from the problem diagram in Figure 2.3 that the requirements also refer to the same set of phenomena.

A.4 The requirements description

The requirements are the effects that the customer wants to observe in the controlled domain given the commands initiated by the operator. The VCR problem diagram in Figure A.2 shows the requirements and the phenomena to which they refer. The description of the requirements is detailed here.

As discussed before, the operator is a biddable domain and his/her actions cannot be controlled. The operator of the remote control receives instructions explaining how commands should be issued, but there is no way to enforce that he/she follows the procedure. For example, he/she may be instructed not to press the button to rewind twice, since the second command will have no effect on the VCR's behaviour. The requirements should describe which commands do affect the behaviour of the VCR and in which context. It should also describe what the effects are. The

requirements description is used to derive the machine specification. Commands that do not make sense in a particular context are not sensible and should be discarded by the machine[1]. However, that is part of the machine specification, not the requirements. The requirements only define the sensible commands and their effects.

Table A.1 defines which commands are sensible given the preceding issued command.

Preceding command	**Sensible commands**
Stop	Play, Rewind, Fast-forward
Play	Stop, Pause, Rewind, Fast-forward
Pause	Pause, Stop, Play, Rewind, Fast-forward
Rewind	Pause, Stop, Play, Fast-forward
Fast-forward	Pause, Stop, Play, Rewind

Table A.1: Sensible commands in each context

The state machine diagram in Figure A.5 shows the requirements of the VCR problem. It shows the effects that should be observable in the VCR given that a sequence of sensible commands is issued. It refers to the sensible commands and constrains the VCR's behaviour.

A.5 The specification description

A specification describes what the machine has to do in order to bring to the problem domain the desired effects described in the requirements. A specification is a description of the behaviour of the machine that must be built considering only its interface with the problem domains. It talks about the phenomena shared with the problem domains.

In the VCR problem, the machine must:

- recognize the commands issued by the remote control operator via the infrared port. The commands that the machine should detect are: Play, Rewind, Fast-forward, Pause, Stop
- discard the commands that are not sensible as described by the requirements in Table A.1
- generate appropriate events that are shared with the VCR mechanism so that the desired effects can be produced considering the properties of the mechanism. Table A.2 shows

[1]The machine also ignores commands that are not *viable*, that is, that are not allowed given the current state of the domain; the VCR problem described here does not have such commands.

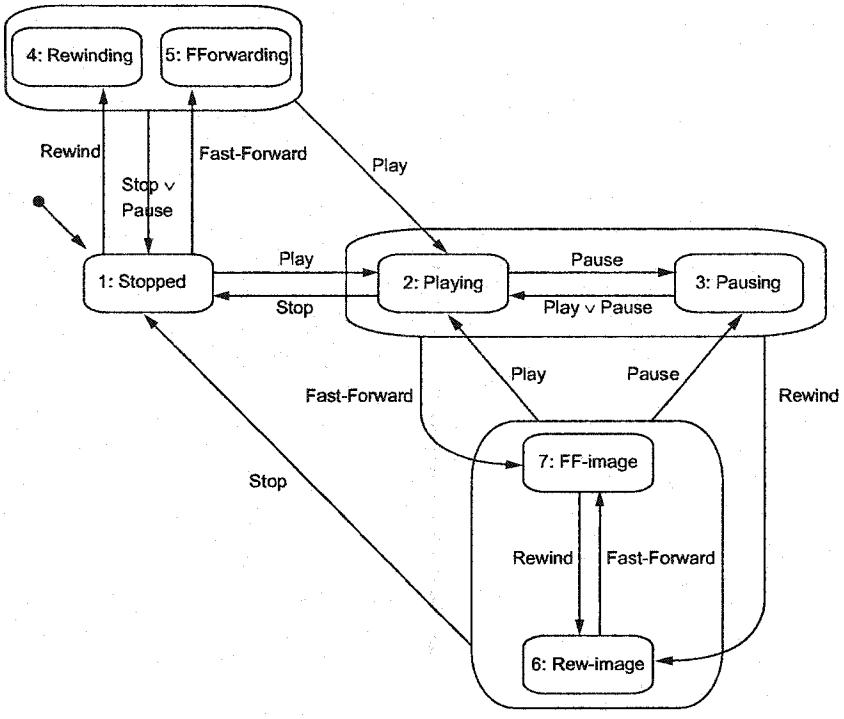


Figure A.5: A state machine diagram describing the requirements for the VCR problem

which events of the VCR mechanism should be generated by the machine according to the commands issued by the remote control operator. Since for each command more than one event has to be generated, the order in which these events are issued also matter.

Context and command	**Event shared with the VCR mechanism**
Stop	SetMOff; DisHead; SetDirFor
Play	SetDirFor; EngHead; SetMOn
Rewind	SetMFast; SetDirBack
Fast-forward	SetMFast; SetDirFor
¬Pausing ∧ Pause	SetDirFor; SetMOff
Pausing ∧ Pause	SetMOn

Table A.2: Specification describing the correspondence and order in which the events have to be generated in response to the remote control commands

Appendix B

Geographic Concepts

In this appendix, some geographic concepts are defined and explained. Terms like scale, datums, map projections, and coordinate systems are discussed in order to lay the foundations for describing geographic applications. They also clarify the terminology of the field.

B.1 About geography

Eratosthenes, an ancient Greek scholar in the 3rd century BC, coined the term *geography*. The word is made up of Greek roots: *ge* means the earth and *grafia* refers to writing. Thus, geography in Greek literally means "writing about the earth".

As a discipline, the meaning of geography has been evolving along the centuries. For example, Ptolemy, circa AD 150, defined the purpose of geography as "to provide a view of the whole earth by mapping the location of places" [Sto86]. The definition has evolved to acknowledge the human contributions and interactions with the earth's environment as part of the discipline's field of study. This is reflected in a recent definition by Martin Kenzer: "...concerned with the locational or spatial variation in both physical and human phenomena at the earth's surface." [Ken89]

B.2 Spatial versus geographic(al)

The terms spatial and geographic(al) are frequently used interchangeably. However, their meanings are slightly different. The following definitions were selected from the Oxford English Dictionary and are appropriate for the purpose of this work.
spatial - "of, pertaining, or relating to space"

space - "denoting area or extension"
geographic(al) - "of or pertaining to geography"
geography - "the science which has for its object the description of the earth's surface"

The term *spatial* has a broad meaning, while the term *geographic* refers specifically to the earth's surface. The term geographic can be applied in a subset of the situations where the term spatial can be applied. According to David Mark, the term geographic refers to "relations at scales ranging from a few meters to the size of the earth's planet" [Mar93]. He also provides examples that would be included under 'spatial' but not 'geographic': shapes of molecules, typical CAD (computer-assisted design) applications, and the relative positions of stars and other bodies in astronomy.

'Geographic' and 'geographical' are both acceptable spelling alternatives in British and American usage, however, 'geographic' is the preferred form in North America and 'geographical' in Europe [Mar93].

This work concentrates on geographic rather than spatial applications. Although there is a difference in meaning between spatial and geographic, in this work both terms will be used interchangeably with the more restrictive meaning of *geographic*.

B.3 Map

A map representing the real world is an entity of concern in a geographic application. For the purpose of this work, a map is defined as "a set of symbols recorded in spatial relationship to each other, so that the position of the symbols acts as an integral part of the message" to the reader of the map [Chr97]. The set of symbols represents concrete or abstract features that occur on or near the surface of the earth [Cam98]. Roads, rivers and cities are examples of concrete features while examples of abstract features may range from political divisions to spoken language or levels of income. In general, concrete features are related to physical geography while abstract features are related to social and cultural geography.

The reader of the map has to decode the symbols and interpret the message or meaning contained in them. In order to do that, the reader needs information about the context of the map. Scale, projection, coordinate system, datum, and accuracy are all important characteristics to consider when interpreting a map.

B.4 Scale

Map scale is "the ratio between the dimensions of the map and those of reality" [RMM+95]. It depends on the purpose of the map. The scale can be expressed as [MM98]:

- a word statement: for example, *1 millimeter to 1 kilometer*
- a representative fraction or an arithmetic ratio: for example, 1/100,000 or 1:100,000 means that one unit of distance on the map represents 100,000 of the same units on the earth's surface. Note that if the scale is represented as a ratio or fraction, it does not have a unit. The numerator of the fraction is always 1 and represents the map distance while the denominator indicates the distance on the ground.
- a graphic symbol: for example, a small ruler printed on the border of a map and subdivided into units appropriate to the scale of the map is a bar scale. In some maps where the scale may vary from one part to another, a variable graphic scale is used.

Maps are often referred to as large or small scale. Although there is no standard classification, the following definitions are generally acceptable:

large scale ranges from 1:50,000 to any larger scale. Large scale maps present great detail but only cover a small area of the world.

medium scale ranges from 1:50,000 to 1:250,000.

small scale ranges from 1:250,000 to 1:7,500,000. Small scale maps cover large areas but with very little detail.

B.5 Datums, coordinate systems and projections

The earth has a spherical shape flattened at the poles, which makes its shape similar to an ellipsoid. Its shape is difficult to represent on a flat map sheet. To create a map of the real world, two operations must be performed. The first one is *georeference*, meaning that the geographic features must be located over a spheroid or an ellipsoid. The second operation is *projection*, meaning that the spheroid or ellipsoid must be projected onto a flat sheet. Figure B.1 illustrates both operations.

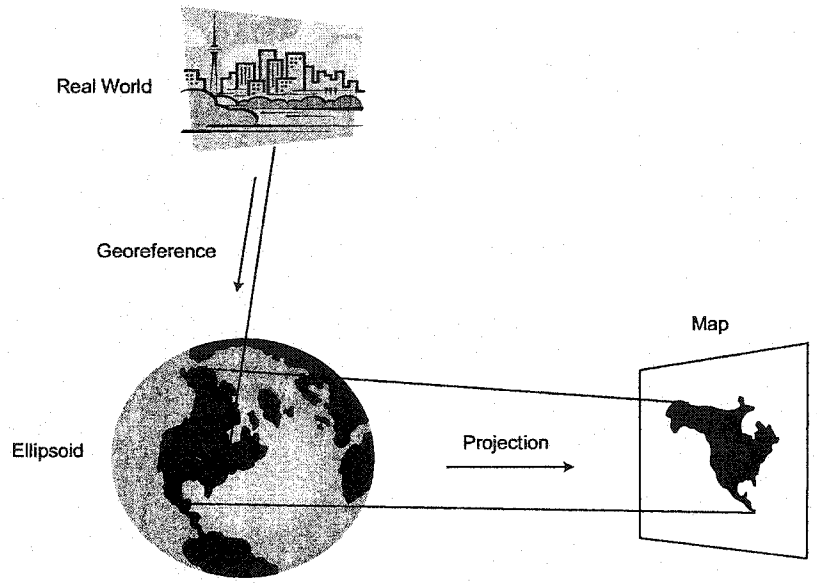


Figure B.1: Georeferencing and projecting the real world onto a map

B.5.1 Datums

An ellipsoid is a geometric model of the shape of the earth. There are many different ellipsoids that were created with the purpose of representing the earth's shape in different or more accurate ways. Each country or agency uses different ellipsoids to make an approximation of the earth's surface [Dan97b].

Georeferencing assigns real world locations to three dimensional points on an ellipsoid. Georeferencing is done with a datum and a geographic coordinate system. A datum[1] is a mathematical model of the earth that is used to calculate coordinates. Basically, a datum refers to a particular ellipsoid and defines the position (origin point and direction) of the ellipsoid relative to the earth. Different datums were created to best represent different regions of the world. Figure B.2 shows the difference between local and global datums [2].

A global datum is used for any worldwide application. It positions the ellipsoid at the center of mass of the earth and uses an ellipsoid that best approximates the shape and size of the earth. A local datum is used for a particular region of the world. It positions the ellipsoid relative to a surface point defined for a national or regional survey. It approximates the ellipsoid to the actual

[1]The plural of the term *datum* is *datums*, when used in its geographic context, as opposed to meaning "a fact or proposition" (Oxford English Dictionary).

[2]adapted from [SNKR01].

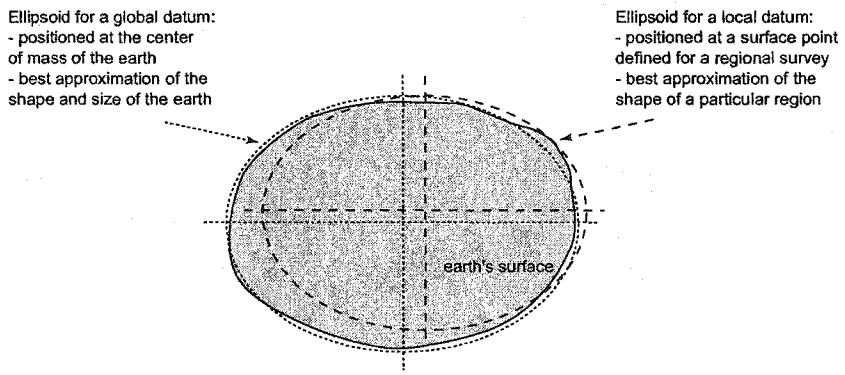


Figure B.2: Differences between datums

surface of the region better than a global datum.

Three different datums have commonly been used to map North America. The North American Datum of 1927 (NAD27) uses a single surface control point at Meade's Ranch, Kansas, central USA. NAD83 is centered at the earth's center of mass. The World Geodetic System (WGD84) is currently the best fitting datum for worldwide applications [SNKR01]. The Global Positioning System (GPS) is based on the WGS84[Dan97b].

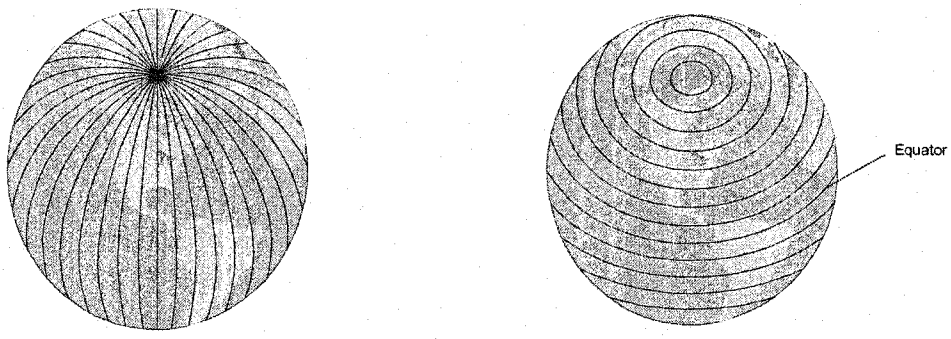
Conversion between datums may involve operations of translation, rotation and re-scaling the points on the source datum to achieve their correspondent points on the target datum. Referencing coordinates to the wrong datum can result in position errors of hundreds of meters.

B.5.2 Geographic coordinate system

The most common earth coordinate system is the latitude-longitude system, also called geographic coordinate system. This is a spherical system which provides angular coordinates used to locate any position on the earth's surface.

Each particular geographic coordinate system is defined by an ellipsoid and a datum that are used as a reference to determine latitude and longitude coordinates [WSC95]. This system allows each point and feature on the earth's surface to be uniquely positioned.

In the latitude-longitude coordinate system, several lines are drawn on the earth's surface. *Meridians* or lines of longitude are placed from pole to pole as shown in Figure B.3(a). *Parallels* or lines of latitude are placed at right angles to the meridians. A *prime meridian* is the starting point of the coordinate system. Since all the meridians are identical, the meridian of Greenwich,



(a) Meridians (b) Parallels

Figure B.3: Lines of Longitude and Latitude

England is usually chosen arbitrarily as the prime meridian. The equator is the central parallel located midway between poles. The parallels are concentric small circles that are parallel to the equator, as seen in Figure B.3(b)

If we imagine that the earth has a spherical shape, the Geodetic latitude of point P is the angle λ from P to the center of the sphere to the equator as shown in Figure B.4. Since the earth is not completely spheric the angle is not measured precisely at the center of the earth, but using a normal line from the point P to the equator [RMM+95]. If λ is positive it means that the point is north of the equator. λ is measured in degrees, minutes and seconds. The equator is at latitude 0° while the north pole is at 90° and the south pole at $-90°$.

If we divide the equator into 360°, the longitude of point P is the angle ϕ starting from the

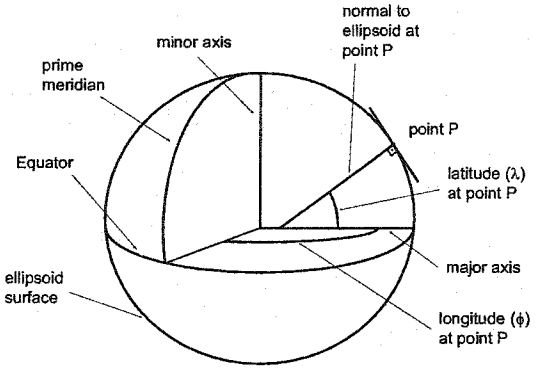


Figure B.4: Latitude and Longitude angles

intersection point between the meridian that passes through P and the equator, to the center of the sphere, to the intersection between the prime meridian and the equator. Figure B.4 also shows the longitude angle. Longitude ranges from $-180°$ west of the prime meridian to $+180°$ east of the prime meridian.

Having the latitude-longitude coordinate system means that any point can be located on the earth's surface (approximated by the ellipsoid surface). However, this surface is not flat, and in order to be able to draw a flat map a projection is necessary.

B.5.3 Projections

"Map projections are attempts to portray the surface of the earth or a portion of the earth on a flat surface" [Dan95]. Map projections are used to transform two angles (latitude and longitude) in three dimensions, to x and y Cartesian coordinates in two dimensions. As a result of the projection process, distortions of distance, direction, scale, area, and conformality may happen. Conformality is the property of the retention of correct shapes on a map [Cam98]. Some projections try to minimize the distortion in only one of these properties. This has the effect of maximizing the distortions in other properties. Some projections are attempts to minimize the distortion in all of these properties. It is important to take the map projection into account when dealing with map applications, since different map projections result in different spatial relationships between regions.

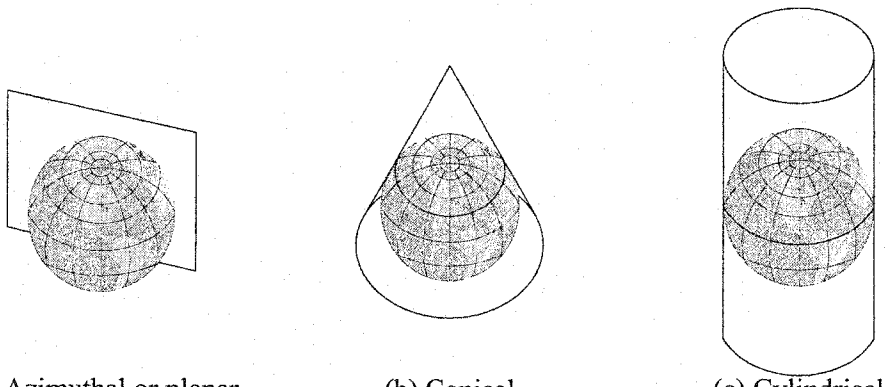
There are a variety of map projections, but they can be classified as belonging to three basic types:

Azimuthal projections - Azimuthal or planar projections are constructed by placing a plane tangent to the earth's surface and projecting the area to be mapped onto the plane as in Figure B.5(a). Examples of this type of projection are the Azimuthal Equidistant and Lambert's Azimuthal Equal area projections.

Conical projections - Conical projections are achieved by wrapping an imaginary cone around the earth, as shown in Figure B.5(b). In the conical projections, parallels and meridians are projected onto a cone tangent to the sphere, along a circle which is usually a mid-latitude parallel. As the locations move away from this tangent parallel, the distortion produced by the projection increases. Conical projections are best suited for maps of mid-latitude regions that are longer in the east-west direction than in the north-south direction. Canada and the USA meet these qualifications and are frequently mapped on conic projections [Cam98].

Alber's conical equal area projection and Lambert's conformal conic projection are examples of conic projections.

Cylindrical projections - Cylindrical projections result from wrapping a cylinder around the globe, as shown in Figure B.5(c). The cylinder may be positioned to touch the globe at the equator (regular), or at a central meridian (transverse) resulting in different projections.



(a) Azimuthal or planar (b) Conical (c) Cylindrical

Figure B.5: Map projections

There are a variety of projections in each of these categories, often with different rules for positioning the projecting surface. For instance, in each of the three categories, the projecting surface may be secant to the globe instead of tangent. Some common projections that are used today are variations of the Mercator and Transverse Mercator projections which are both cylindrical projections.

Mercator Projection

In the Mercator projection, the sphere is projected onto a cylinder tangent to the equator. All parallels and meridians are straight lines. Parallels meet meridians at right angles. Meridians are equally spaced, while spacing between parallels increases towards the poles. The scale is not distorted at the equator. Angles and shapes within any small area are not distorted, and as a result this projection is considered conformal. The projection is often used for marine navigation, since all straight lines in the map are lines of constant bearing [RMM+95]. Figure B.6 shows a world map using the Mercator projection[3]. Observe how the Mercator projection enlarges the areas close to the poles. Alaska in North America

[3] generated using the Online Map Creation Tool [OMC96].

appears about the same size as Brazil in South America, but in fact Brazil is over five times larger than Alaska.

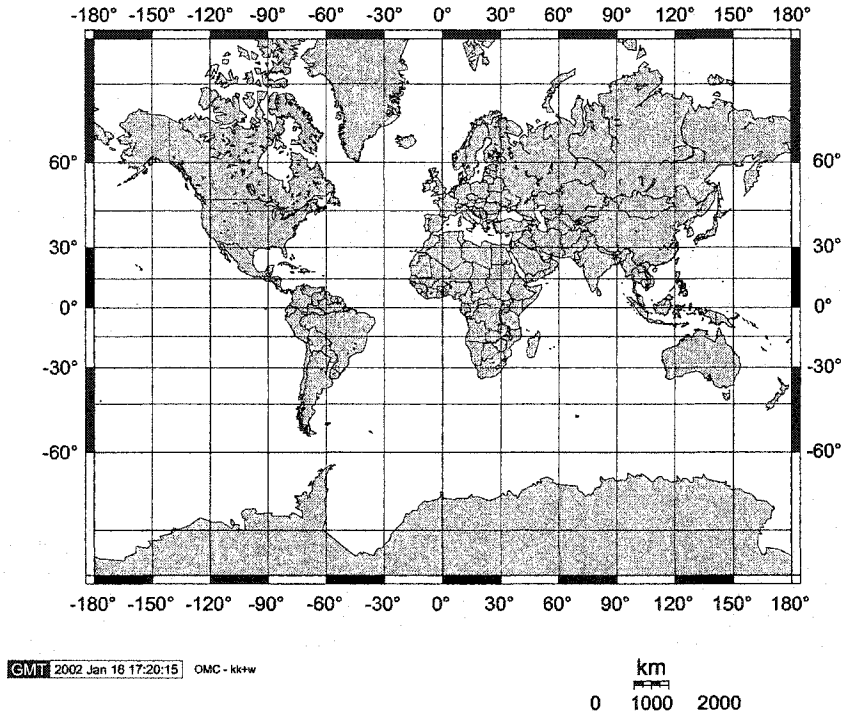


Figure B.6: A Mercator projected world map

Transverse Mercator Projection

Transverse Mercator projections result from projecting the sphere onto a cylinder tangent to a central meridian [RMM+95]. The central meridian and parallels are straight. Other meridians become complex curves. Transverse Mercator maps are conformal since the angles and shapes within any small area are not distorted. Transverse Mercator maps are often used to portray areas with larger north-south than east-west extents. Distortions of scale, distance, direction and area increase away from the central meridian.

Universal Transverse Mercator (UTM) Projection

This projection is a special case of the Transverse Mercator Projection where the central meridian is moved around the earth by rotating the cylinder around the poles.

The earth is divided into 6° longitudinal strips called *zones* which are identified by a number. Each zone extends from 80° South latitude to 84° North latitude. Each zone has a

central meridian that is used as the tangent circle for the projection. Each zone is projected individually using a Transverse Mercator projection. The central meridian is the area of least distortion. Since there are 60 different zones, each central meridian at each zone is an attempt to correct the distortions around that area of the world. The UTM projection is used for world-wide applications. Letters are used to designate horizontal 8° sections extending north and south from the equator. Figure B.7 shows the UTM zones[4]. The province of Ontario in Canada, for example, lies in the UTM zones 15, 16, 17 and 18.

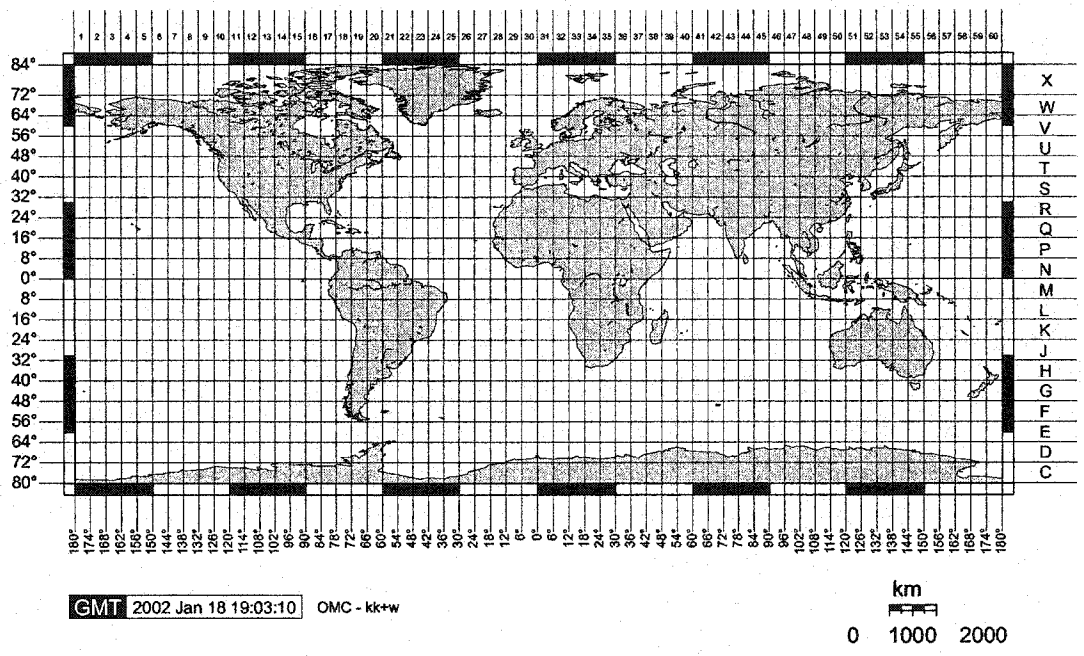


Figure B.7: UTM zones

The choice of map projection depends on the location under study, the properties (distance, area, direction, etc.) for which distortion should be minimized, and other factors. Maling [Mal92] suggests several principles and methods for choosing a suitable map projection. These are beyond the scope of this work.

It is important to note that besides using a map projection to flatten the earth's surface, a rectangular coordinate system is also necessary to identify the points of the flat surface. A map projection simply defines how the surface of the earth is represented on a flat surface. A rectan-

[4]redrawn from [Sny87] using the Online Map Creation Tool [OMC96].

gular coordinate system needs to be superimposed on this projected surface and referenced to the datum and ellipsoid of the original latitude-longitude coordinates [Cam98].

B.5.4 Rectangular coordinate systems

Rectangular coordinate systems[5] use two-dimensional Cartesian coordinates along the x and y axes, and are defined by four aspects: a map projection referenced to a datum and ellipsoid, an orientation, a point of origin, and a unit of measurement [WSC95]. A map projection is only part of what needs to be considered to define a rectangular coordinate system.

There are local and global rectangular coordinate systems. Local rectangular coordinate systems were developed by many nations and define grid systems based on coordinates that cover their territory. Examples of local coordinate systems are the British National Grid, the Irish National Grid (both are based on the Transverse Mercator projection), and the State Plane Coordinate System in the USA, which is based on Lambert conformal conic and transverse Mercator projections [Mal92]. The UTM coordinate system is the most common global rectangular coordinate system.

Universal Transverse Mercator (UTM) coordinate system

The UTM system is a rectangular coordinate system used with the UTM projection. As previously described, a UTM zone has a central meridian. The UTM coordinate of an arbitrary point P is a pair of numbers representing the "Easting" (offset measured from the central meridian to P in kilometers or meters) and "Northing" (offset measured from the equator to P in kilometers or meters) coordinates [SMB85]. The central meridian is assigned the arbitrary "Easting" coordinate of 500 kilometers to avoid negative offsets. The equator is considered the 0km horizontal axis for the northern hemisphere. Figure B.8 shows the coordinate grid for each zone[6].

In order to avoid negative numbers, the southern hemisphere coordinates may consider the equator at 10,000 km and use a letter character designating the horizontal section to which the coordinate belongs. If negative coordinates are used, the letters designating horizontal sections become redundant information. However, they are usually part of the UTM coordinate system to ensure only positive coordinates. Figure B.7 shows the UTM zones and horizontal sections in a world map.

There are some advantages of using UTM coordinates instead of latitude-longitude [Car01]:

[5] also known as *projected* or *grid* coordinate systems.

[6] redrawn from [Car01].

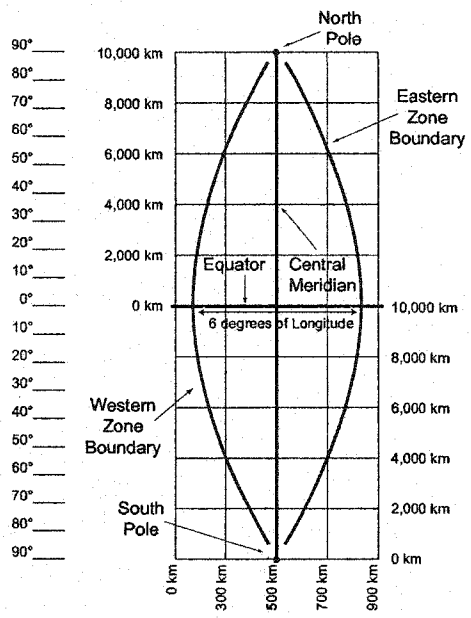


Figure B.8: A UTM zone

- The coordinates are a base-10 metric system instead of base-60 system with degree, minute and second as units

- The rectangular UTM coordinate system provides a constant distance. relationship anywhere on the map. For example, a point which is in UTM zone 10 and has coordinates (559.7,4282.1) is exactly 1 kilometer away from the point in zone 10 (559.7,4281.1). This is true for any zone in any part of a map. In angular coordinate systems like latitude-longitude, the distance covered by a degree of longitude differs as you move towards the poles. Besides, only at the equator is the distance covered by a degree of longitude equivalent to the distance covered by a degree of latitude.

The drawback of the UTM coordinate system is that it is difficult to calculate distance or perform any calculation involving points that are in two or more different UTM zones.

Other types of coordinate systems that do not fit under the rectangular category but are worth mentioning are the postal code systems, such as the Canadian postal code and American ZIP code. They have also proven to be useful for some applications.

Table B.1 shows coordinates of the star in the hand of the Goddess of Liberty atop the Capitol building in Austin, Texas, expressed using different coordinate systems and datums [Dan97a].

Datum	Coordinate System	Coordinates	Units
NAD 83	Geodetic Latitude, Longitude	30:16:28.82 N, 97:44:25.19 W	deg:min:sec
NAD 27	Geodetic Latitude, Longitude	30:16:28.03 N, 97:44:24.09 W	deg:min:sec
WGS 72	Geodetic Latitude, Longitude	30:16:28.68 N, 97:44:25.75 W	deg:min:sec
NAD 83	UTM Easting, Northing, Zone	621160.98, 3349893.53 14 R	meters
NAD 27	UTM Easting, Northing, Zone	621193.18, 3349688.21 14 R	meters
NAD 83	Military Grid Reference System	14RPU2116149894	meters
NAD 27	Military Grid Reference System	14RPJ2119349688	meters
WGS 72	World Geographic Reference System	FJHA1516	deg. and min.
	U.S. Postal Zip Code (5-digits)	78705	

Table B.1: A single location described by a variety of coordinate systems

It illustrates how the actual coordinates can be affected by the different datums and coordinate systems.

B.5.5 Summary

Datums, projections, and coordinate systems are extremely important map characteristics. When relating two maps or features from different maps, it is important to know their datums, projections and so on in order to combine the information appropriately. Maps created using two different datums may place the same location up to 1,000 meters apart. Combining such maps without careful datum conversion may result in completely inaccurate results. Maps using different projections should not be combined before the transformations to convert from one projection to another are performed. However, in the rush of preparing geographic data for map use and analysis, this is a frequent omission in geographic applications.

B.6 Map classification

Maps can be classified according to different criteria. For example, the scale of the map may be used as the criterion, and maps can fall into categories of large or small scale, as discussed previously. Maps can also be classified according to their function, or according to the data model used to create the map. Reference and thematic maps serve two distinct functions. Field and object maps differ in the way they gather and manipulate the data.

B.6.1 Reference versus thematic maps

Reference or general-purpose maps are designed to accurately show the position of real world phenomena. Thematic or special-purpose maps are designed to show the variation of the distribution of a real world phenomenon over the geographic space [MM98].

The purpose of reference maps is to provide the location of relevant features or phenomena from the real world. The focus when creating the map is on representing spatial relationships faithfully so that the reader can rely on the map to derive metric information. This type of map is useful for determining locations of geographic features. No particular feature receives more importance over the others in reference maps. Road maps are an example of reference maps, since they usually provide accurate locations of cities, roads, rivers, lakes, and parks.

Thematic maps emphasize one feature or a single phenomenon from the environment. The purpose is to show the spatial variation in the form of geographic distribution of a particular feature, not to show the location of the feature. Examples of thematic maps include maps of precipitation, temperature, population density, or average annual income. These maps show the variation of these phenomena across a region, a country or the world [RMM+95]. Many thematic maps show abstract phenomena that do not physically exist in the real world.

The fact that a map emphasizes one phenomenon does not mean that the map is thematic. The focus has to be the spatial distribution of the phenomena rather than its precise location. Most thematic maps present some type of locational reference such as political divisions and province borders. However, the reader of the map should not use this spatial reference to find specific locations or derive other measurements. This is not the purpose of thematic maps, but of reference maps.

B.6.2 Field versus object model

There are two opposing types of models to gather, store and manipulate geographic information: field-based and object-based models. These two types of models are used by geographers and cartographers to produce maps [Wor95].

Field-based models treat the information as continuous geographic phenomena distributed over the space. Patterns of rainfall and temperature are examples of geographic phenomena that can be represented appropriately by the field model.

In this approach, the information is captured and stored as a collection of fields. Each field may be formalized as a mathematical function from a set of locations to a set of attribute values that characterize the geographic phenomena. The set of locations is created, for example, by dividing

a given region into a finite grid of squares where each individual square is a location approximated by a point. Then, the phenomenon under consideration is sampled at the determined locations. The spatial fields are constructed as functions mapping each location to the sampled attribute value.

Object-based models treat the geographic information as collections of discrete, identifiable entities that populate the space. Each of these entities has a georeference and attributes describing a geographic phenomenon. For example, cities can be represented in the object model as discrete entities with associated location, average temperature, average rainfall, population, etc.

Instead of identifying the attribute value of the phenomena for each location as in the field model, the object model clumps together all the attribute values for one specific object. It also maps this discrete and identifiable object to its location.

The object model can be seen as the inverse of the field model. In the field approach, the functions (fields) map from locations to the attributes. In the object approach, the function maps from entities (a collection of attributes) to their locations.

Although the field and object models are concepts independent of implementation, the field model is more naturally implemented by raster data structures and the object model more suitable to be implemented using a vector format. However, the field/raster or object/vector approaches are quite independent.

Object-models have two subcategories depending on whether or not the relationship between objects is captured and stored in the model.

Isolated object model

In this model objects are described using simple geometric primitives such as point, line and area. Each object is described as a geometric whole, since it occurs in isolation from the others. The only relationship in this model is between the object and a position. The object contains also any non-geographic attributes. All computer aided design (CAD) packages treat their objects in this manner.

Connected object model

In this model objects are described in relation to each other. For example, a point may be the end point of a line or a point of intersection between two lines. A line may divide an area into two polygons. These relationships are not recognized at the isolated model. The connected model is, however, mostly concerned with the description of these relationships which are called topological. "Topology is a branch of mathematics concerned with those properties of geometry that

are unchanged by any continuous deformation" [Chr97]. As used in cartography, topology is concerned with "the relative location of geographic phenomena independent of their exact position" [Inf96] and it belongs to a branch of topology called combinatorial topology [Cor79].

Although topology can be mathematically defined it is better understood by intuition. Consider an unbounded thin sheet of rubber that can be stretched, contracted or twisted to any degree but not torn or cut. Given, for example, a polygon and a point inside it, drawn in this rubber sheet, after any amount of deformation the point would be still inside the polygon. This "insideness" relationship is an example of a topological relationship. The transformation induced by the deformation of a rubber sheet is called a topological transformation. Topological relationships are preserved under any topological transformation.

To capture the topological relationships, features need to be stored using node, edge and polygon primitives. Nodes represent the beginning, the ending and any intersection of linear features which are, in the connected model, represented by edges. Nodes bound edges and edges bound polygons. From these simple relationships, more complex ones can be built. For example, if two polygons share a common edge, then they are said to be adjacent. Connectedness and contiguity are other examples.

A map of a subway network is usually made using the connected model. The important message to convey regards which stations are adjacent and connected to each other, so that a person can find a path from one station to another. These subway network maps are usually diagrams that emphasize the connectivity of the stations over their precise locations.

B.6.3 Summary

These classifications are often orthogonal and do not have a one to one correspondence. Reference maps that use the object model are, for example, the ones found in most atlases and road maps. An aerial photograph provides a good example of a reference map that uses the field model. Thematic maps captured with the field model are very common. Most of the climate maps shown on newspapers and magazines fall into this category. A map where each county of a province is an object with a color that represents the average income level of its population is an example of a thematic map using the object model.

Appendix C

The Alloy language

The purpose of this Appendix is not to be a complete reference for the Alloy notation, but to provide enough information to understand the formal descriptions presented in this dissertation. The information contained here was adapted from [Jac].

The Alloy language consists first-order predicate logic, sets, and relations. Logic statements are used to describe constraints over the sets and relations.

C.1 Sets and relations

A *signature* designates a set. It may also define *attributes*. In the example below, the signatures *A* and *B* are sets, and *C* is an attribute of *B*, such that each member of *B* will be related to a set of members of *A* via relation *C*.

```
sig A { }
sig B { C: set A }
```

A description of how to add constraints over the signatures can be found in Section C.2.4.

The multiplicity of attributes may fall into three categories:

Declaration	Semantics
`sig A { B: C }`	B refers to exactly one C
`sig A { B: option C }`	B refers to zero or one elements of C
`sig A { B: set C }`	B refers to zero or more elements of C

Attributes may be relations. In the following example, the attribute C is a relation between elements of set A.

```
sig A { }
sig B { C: A -> A }
```

Three different multiplicity markings may be added to relation declarations:

Symbol	Semantics
question mark "?"	zero or one
exclamation "!"	exactly one
plus sign "+"	one or more
default (no markings)	zero or more

Multiplicity markings may be added to both sides of a relation symbol. The following are some of the possibilities of combining multiplicities:

Declaration	Semantics
`R: S -> T`	relation
`R: S ->! T`	total function
`R: S ?-> T`	partial injective relation
`R: S ?->? T`	partial injective function
`R: S ->+ T`	total relation
`R: S +-> T`	surjective relation
`R: S !->! T`	bijection

Different signatures are disjoint sets, unless specified using the `extends` keyword, which designates a subset. In the example below, sets B and C are subsets of set A.

```
sig A { }
sig B, C extends A { }
```

The keyword `disj` designates disjoint subsets, and the keyword `part` designates set partition.

```
sig A { }
disj sig B, C extends A { }
part sig D, E extends A { }
```

C.1.1 Set operations

The following are operations that may be performed over sets:

Operator	Semantics
`A + B`	union of A and B
`A & B`	intersection of A and B
`A - B`	A with elements of B removed

C.1.2 Relation operations

The following unary operators can be used on relations:

Operator	Semantics	Result if R = { (a → b), (b → c) }
`~R`	transverse	{ (b → a), (c → b) }
`^R`	transitive closure	{ (a → b), (b → c), (a → c) }
`*R`	reflexive transitive closure	{ (a → b), (b → c), (a → c), (a → a), (b → b), (c → c) }

If *(A: B → C)* is a relation and *b1* is a subset of set *B*, then *b1.A* is the set resulting from applying relation *A* to all elements of set *b1*. The expression *A[b1]* is equivalent to *b1.A*.

C.2 Logic statements

C.2.1 Logical operators

This is the syntax for the first-order logic operators:

Expression	True when
`F && G`	both F and G are true (conjunction)
`F || G`	F or G is true (disjunction)
`F => G`	F is false or G is true (implication)
`F <=> G`	both F and G are true or both are false (if and only if)
`!F`	F is false (negation)

C.2.2 Set expressions

A number of expressions involve sets. These are:

Expression	True when
`A = B`	A and B have the same elements
`A != B`	A and B do not have the same elements
`A in B`	every element of A is an element of B
`A !in B`	some element of A is not an element of B
`some A`	A is non-empty
`no A`	A is empty
`sole A`	A has at most one element
`one A`	A has exactly one element

Sets can be described using set comprehension. The expression $\{ a: A \mid F \}$ denotes the subset of A for which expression F is true.

C.2.3 Quantifiers

Alloy has five logical quantifiers:

Expression	Semantics
`all a: A | F`	Expression F is true for all elements of A
`some a: A | F`	Expression F is true for at least one element of A
`no a: A | F`	Expression F is false for all elements of A
`one a: A | F`	Expression F is true for exactly one of the elements of A
`sole a: A | F`	Expression F is true for at most one of the elements of A

C.2.4 Constraints

A description may contain a number of constraints, or *facts*, over the sets and relations. A fact is a statement that must be true in any instance of the model. The fact below states that if a is in set A, it is also in either set B or set C, or both.

```
fact { all a: A | a in (B + C) }
```

An *implicit fact* is an invariant appended to the end of a signature declaration, without the *fact* keyword. This is invariant applies to all instances of the signature. For example, the following specifications are equivalent:

```
sig A { a: set B }
{ some a }
```

```
sig A { a: set B }
fact { all this: A | some this.a }
```

C.2.5 Functions

A *function* groups a number of logical statements. Functions are not automatically evaluated as constraints over the model, like facts. They can be used in other logical statements to simplify descriptions. A function can return a result; in this case, it can be used as part of an expression, if the function is deterministic. The keyword *fun* declares a function, and the keyword *result* denotes the returned value. The following function *f* takes a parameter of type *A* and returns an element of set *B*.

```
fun f(x: A) : B { result = x.a }
```

If a function does not constrain the value of *result* using an equality statement such as the one above, it is considered to be nondeterministic. However, if the function is, in fact, deterministic, the keyword *det* can be prepended to the function declaration as a way to inform the analyzer that this is a deterministic function. For instance, in the example below, if it is known that there is always only one element that has the desired constraints, the function can be stated to be deterministic.

```
det fun f(x, y: A) : B { result in x.a && result !in y.a }
```

Functions can be restricted to work on specific types, using the :: notation. The following function *f* is attached to type *A*. The keyword *this* refers to the recipient element of the function. Such functions are accessed using the double-dot notation (..).

```
A::f( ) { this in B }
// The following fact illustrates the use of function f
fact { all x: A | x..f() }
```

Appendix D

Alloy descriptions of geographic problem frames

This appendix contains the formal descriptions of the geographic problem frames using the Alloy notation. The frames presented here were used in the case studies discussed in Chapter 8. The appendix also contains the nine intersection model constraints that were discussed in Section 7.2.4 as well as some supporting modules (*Ord* and *NumericValues*).

D.1 Ord module

The *Ord* module is part of the Alloy distribution. It defines a template that allows you to impose a total order on a given type.

```
module std/ord

static sig Ord [t] {
   first, last: t,
   next, prev: t -> t
}
{
  // we require t=univ[t] in order to preserve the constraint
  // that Ord[t].last is a subset of t and the domain and range of
  // Ord[t].next lie inside t.
  t = univ[t]

  // constraints that actually define the total order
  prev = ~next
  one first
```

```
  one last
  no first.prev
  no last.next
  (
   // either t has exactly one atom,
   // which has no predecessor or successor...
   (one t && no t.prev && no t.next) ||
   // or...
    all elem: t | {
      // ...each element (except the first) has one predecessor, and...
      (elem = first || one elem.prev)
      // ...each element (except the last) has one successor, and...
      (elem = last || one elem.next)
      // ...there are no cycles
      (elem !in elem.^next)
    }
  )
  // all elements of t are totally ordered
  t in first.*next
}

// return the predecessor of elem, or empty set if elem is the first element
fun OrdPrev [t] (elem: t): option t { result = elem.(Ord[t].prev) }
// return the successor of elem, or empty set of elem is the last element
fun OrdNext [t] (elem: t): option t { result = elem.(Ord[t].next) }

// return elements after elem in the ordering
fun OrdPrevs [t] (elem: t): set t { result = elem.^(Ord[t].prev) }
// return elements prior to elem in the ordering
fun OrdNexts [t] (elem: t): set t { result = elem.^(Ord[t].next) }

// two-element comparison functions

fun OrdLT [t] (e1, e2: t) { e1 in OrdPrevs(e2) }
fun OrdGT [t] (e1, e2: t) { e1 in OrdNexts(e2) }
fun OrdLE [t] (e1, e2: t) { e1=e2 || OrdLT(e1,e2) }
fun OrdGE [t] (e1, e2: t) { e1=e2 || OrdGT(e1,e2) }

// return the smallest element out of a set of ordered elements
fun OrdSmallest [t] (elems: set t) : option t {
    result = elems.*(Ord[t].next) - elems.^(Ord[t].next)
}
```

D.2 NumericValues module

The *NumericValues* module describes a set of ordered values with addition and subtraction operations.

```
module map/numericvalues

open std/ord

// Value is a subset of LargeValue
sig LargeValue { }
sig Value extends LargeValue { }
part sig Zero, Positive extends LargeValue { }

fact { some Zero & Value }
fact { some Ord[LargeValue] }
fact { Ord[LargeValue].first = Zero }
fact { all v: Value | OrdPrev(v) in Value }

fun add(x, y: Value) : LargeValue {
   result = { i: LargeValue |
        #OrdPrevs(i) = #OrdPrevs(x) + #OrdPrevs(y)
   }
}

// subtraction returns the absolute value (no negatives)
fun subtract(x, y: LargeValue) : LargeValue {
    result = { i: LargeValue |
         (OrdGE(x, y) && #OrdPrevs(x) = #OrdPrevs(i) + #OrdPrevs(y))
     ||  (OrdLT(x, y) && #OrdPrevs(y) = #OrdPrevs(i) + #OrdPrevs(x))
    }
}

fact { all x, y: Value | some add(x, y) }

det fun min(v1, v2: Value) : Value {
   OrdLT(v1, v2) => result = v1, result = v2
}

det fun max(v1, v2: Value) : Value {
   OrdGT(v1, v2) => result = v1, result = v2
}
```

D.3 Nine-intersection map projection

The nine-intersection map projection is presented in Section 7.2.4. Egenhofer and Herring [EH91] enumerate 23 rules that govern the possible intersections in a nine-intersection matrix. These rules are presented here in the form of Alloy facts.

```
// The interior and boundary of a geometry intersects with
// its own interior and boundary but not with its exterior.
fact { all g: TopologicGeometry | {
       g in g.II
       (g !in D0 => g in g.BB)
       g !in g.(IB + IE + BE)
     }
}

// Condition 1: The exterior of two regions intersect with each other
// Condition 22: Both exteriors must intersect (for points)
// all exteriors intersect with all other exteriors
fact { all g: TopologicGeometry | g.EE = TopologicGeometry }

// Conditions for regions

// Condition 2: If both interiors are disjoint then A's interior intersects
// with B's exterior and vice-versa
fact { all a, b: (D2 + D1) & TopologicGeometry |
    (b !in a.II && a !in b.II) => b in a.IE
}

// Condition 3: If A's interior is a subset of B's closure, then A's
// boundary must be a subset of B's closure as well and vice-versa
fact { all a, b: (D2 + D1) & TopologicGeometry |
    (b !in a.IE && b in a.(IB + II)) => (b !in a.BE && b in a.(BB + BI))
}

// Condition 4: If A's interior intersects with B's boundary then it must
// also intersect with B's exterior, and vice-versa
fact { all a, b: (D2 + D1) & TopologicGeometry |
    b in a.IB => (b in a.IE)
}

// Condition 5: A's boundary intersects with at least one part of B, and
// vice-versa
fact { all a, b: (D2 + D1) & TopologicGeometry |
    (b in a.BB) || (b in a.BI) || (b in a.BE)
}

// Condition 6: If both interiors are disjoint then A's boundary cannot
// intersect with B's interior and vice-versa
```

```
fact { all a, b: D2 & TopologicGeometry |
    (b !in a.II && a !in b.II) => (b !in a.BI)
}

// Condition 7: If A's interior intersects with B's interior and exterior,
// then it must also intersect with B's boundary and vice-versa
fact { all a, b: D2 & TopologicGeometry |
    (b in a.II && b in a.IE) => (b in a.IB)
}

// Condition 8: If both boundaries do not intersect with each other then at
// least one boundary must intersect with its opposite exterior
fact { all a, b: D2 & TopologicGeometry |
    (b !in a.BB && a !in b.BB) => (b in a.BE) || (a in b.BE)
}

// Condition 9: If both interiors do not intersect with each other then at
// least one boundary must intersect with its opposite exterior
fact { all a, b: D2 & TopologicGeometry |
    (b !in a.II && a !in b.II) => (b in a.BE) || (a in b.BE)
}

// Conditions for regions without holes

// Condition 10: If both boundaries intersect with the opposite interiors
// then the boundaries must also intersect with each other
fact { all a, b: D2 & TopologicGeometry |
    (b in a.BI && a in b.BI) => (b in a.BB && a in b.BB)
}

// Condition 11: If A's interior intersects with B's exterior then A's boundary
// must also intersect with B's exterior
fact { all a, b: D2 & TopologicGeometry |
    (b in a.IE) => (b in a.BE)
}

// Condition 12: If the interiors do not intersect with each other then A's
// boundary must intersect with B's exterior and vice-versa
fact { all a, b: D2 & TopologicGeometry |
    (b !in a.II) => (b in a.BE)
}

// Line Conditions
// Condition 13: If A's closure is a subset of B's interior then either A's
// exterior intersects with both B's boundary and B's interior, or not at
// all, and vice-versa
fact { all a, b: D1 & TopologicGeometry |
    (b !in a.(BE + IE) && b in a.(BI + II)) =>
          (b in a.EI && b in a.EB) || (b !in a.EI && b !in a.EB)
}
```

```
// Simple Line Conditions

// Condition 14: Each boundary can intersect with at most two opposite parts
fact { all a, b: D1 & TopologicGeometry |
    !((b in a.BE) && (b in a.BI) && (b in a.BB))
}

// Condition 15: If A's boundary is a subset of B's boundary, then the two
// boundaries coincide, and vice-versa
fact { all a, b: D1 & TopologicGeometry |
    (b in a.BB) && (b !in a.EB) && (b !in a.IB) =>
            (b !in a.BE) && (b !in a.BI)
}

// Relations between a Region and a Line

// Condition 3a: If L's interior is a subset of R's closure, then L's
// boundary must be a subset of R's closure as well
fact { all r: D2 & TopologicGeometry, l: D1 & TopologicGeometry |
    (r !in l.IE && r in l.(IB + II)) => (r !in l.BE && r in l.(BB + BI))
}

// Condition 5a: L's boundary intersects with at least one part of R
fact { all r: D2 & TopologicGeometry, l: D1 & TopologicGeometry |
    (r in l.BB) || (r in l.BI) || (r in l.BE)
}

// Condition 6a: If both interiors are disjoint then L's boundary cannot
// intersect with R's interior
fact { all r: D2 & TopologicGeometry, l: D1 & TopologicGeometry |
    (r !in l.II) => (r !in l.BI)
}

// Condition 7a: If L's interior intersects with R's interior and exterior,
// then it must also intersect with R's boundary
fact { all r: D2 & TopologicGeometry, l: D1 & TopologicGeometry |
    (r in l.II && r in l.IE) => (r in l.IB)
}

// Condition 16: The interior of a region A always intersects with the
// exterior of a line B
// Condition 17: The boundary of a region A always intersects with the
// exterior of a line B
// Three positions of the 9-intersection matrix between
// a line and a region are always filled
fact { all l: D1 & TopologicGeometry, r: D2 & TopologicGeometry |
    r in l.EI && r in l.EB && r in l.EE
 }
```

```
// Condition 18: The interior of a line B must intersect with at least
// one of the three parts of a region A
fact { all l: D1 & TopologicGeometry, r: D2 & TopologicGeometry |
    (r in l.IE) || (r in l.IB) || (r in l.II)
 }

// Conditions for points

// A point has no boundary
fact { all p: D0 & TopologicGeometry | no p.BB && no p.BE && no p.BI }

// Condition 19: Interior, boundary and exterior of a non-point object A
// intersect with the exterior of a point B
// The exterior of a point intersects with everything
fact { all p: D0 & TopologicGeometry | {
       p.EI = TopologicGeometry - p
       p.EB = TopologicGeometry - D0
   }
}

// Condition 20: The interior of a point can only intersect with a single
// part of another object
// Condition 21: The interior of a point must be a subset of one of the
// three parts of another object.
// The interior of a point intersects with the exterior
// of another geometry, or its interior, or its boundary (exclusive or)
// Trivial cases between two points
// Condition 23: The interior of a point instersects with exactly one
// opposite object part
// The interior of a point intersects the exterior of
// another point, or with its interior, but not both
fact { all p: D0 & TopologicGeometry, g: TopologicGeometry |
  (    (p in g.II && p !in (g.BI + g.EI) )
    || (p in g.BI && p !in (g.II + g.EI) )
    || (p in g.EI && p !in (g.II + g.BI) )
  )
}
```

D.4 Proximity problem frame

This section presents the generic framework for the proximity problem frame as well as machine specification and requirements specific for finding resources within a given distance from the input resources. The map used in this problem frame follows the metric taxicab projection presented in Section 7.2.3.

```
module geoframes/ProximityFrameTaxicab

open map/metrictaxicab
open map/cartesian
open map/map
open map/numericvalues
open std/ord

//-------------------------------------------------------------------------
// GENERAL SIGNATURES
//-------------------------------------------------------------------------

sig ProximityFrame {
    machine: ProximityMachine,
    req: ProximityRequirement,
    domains: ProximityDomains
}
{  machine.ProximityMachine$domains = domains
   req.ProximityRequirement$domains = domains
}

sig ProximityDomains {
    // domains
    M: MetricTaxicabMap,
    sourceFeature, desiredFeature: set Feature,
    inputResource, proximalResource: set Resource
}
{ inputResource + proximalResource in M.O }

sig ProximityMachine {
   validSource: set Resource,
   validTarget: set Resource,
   domains: ProximityDomains
}
// The facts below represent the weakest constraint over the features that
// the source and target resources must have. In this case, this constraint
// is that the valid resources must have at least one of the desired features.
// Stronger constraints may be added to specific machines.
{
   validSource = { r: Object | r in domains.inputResource &&
                   (some r.F & domains.sourceFeature) }
```

```
   validTarget = { r: Object | r in domains.M.O &&
                   (some r.F & domains.desiredFeature) &&
                    r !in domains.inputResource }
}

sig ProximityRequirement {
   domains: ProximityDomains
}
{
   // proximal resources are in the map
   domains.proximalResource in domains.M.O
   // resources have the desired features
   all r: domains.proximalResource | some r.F & domains.desiredFeature
}

//-------------------------------------------------------------------------
// SPECIFIC FRAME
// "Within distance"
//-------------------------------------------------------------------------

sig WithinDistanceProximityFrame extends ProximityFrame { dist: LargeValue }
{ find_resources_within_distance(machine, dist)
}

// specific machine specification
// find the resources that are within a given distance from the input
// resources
// Given map is a MetricTaxicabMap
fun find_resources_within_distance (m: ProximityMachine, dist: LargeValue)
{
  // Find the points where each of the validSource resources are located
  let sourcePoints = m.domains.M.Location[m.validSource] |
  // Find the points where each of the validTarget resources are located
  let targetPoints = m.domains.M.Location[m.validTarget] |
  // We are computing distances from the locations where the objects are
  // anchored, not taking into account the objects' geometries.
  m.domains.proximalResource = points_within_distance(dist, sourcePoints,
                   targetPoints).~(m.domains.M.Location) & m.validTarget
}

det fun points_within_distance(dist: LargeValue,
                               sp, tp: set Point) : set Point {
    let dists = { d: LargeValue | OrdLE(d, dist) } |
    result = { p1: tp | some p2: sp | taxicabDistance(p1, p2) in dists }
}
```

```
// Specific requirement for the "within distance" problem
// returns the set of resources that are located within distance
// from the input resources
fun within_distance_requirement(pr: ProximityRequirement, dist: LargeValue) {
  // Every resource in the solution
  // Determine which of the input resources are valid
  let sources = { r: pr.domains.inputResource |
          (some r.F & pr.domains.sourceFeature) && r in pr.domains.M.O } | {
  (some sources) => (
          all r: pr.domains.proximalResource | {
                // Input cannot be in the output
                r !in pr.domains.inputResource

               // All resources in the solution set must have distance
               // from the valid sources smaller than "dist"
                OrdLE(smallest_distance(pr.domains.M, r, sources), dist)
           }
           &&
           // No resource outside the solution set must be within distance
           // to the valid sources
           no candidate: pr.domains.M.O - pr.domains.proximalResource
                                        - pr.domains.inputResource | {
               some candidate.F & pr.domains.desiredFeature
               OrdLE(smallest_distance(pr.domains.M,
                                       candidate, sources), dist )
           }
   )
   (no sources) => no pr.domains.proximalResource
 }
}

//-------------------------------------------------------------------
// ASSERTIONS
//-------------------------------------------------------------------

assert frameConcern_within_distance {
    all f: WithinDistanceProximityFrame, d: LargeValue |
          find_resources_within_distance(f.machine, d) =>
                              within_distance_requirement(f.req, d)
}
```

D.5 Site selection problem frame

This section presents the generic framework for the site selection problem frame as well as machine specification and requirements specific for finding empty regions within a given distance from the input resources. The map used in this problem frame follows the metric projection presented in Section 7.2.3.

```
module geoframes/SiteSelectionFrame

open map/metric
open map/metrictaxicab
open map/metricdistance
open map/cartesian
open map/map
open map/numericvalues
open std/ord

//-------------------------------------------------------------------------
// GENERAL SIGNATURES
//-------------------------------------------------------------------------

sig SiteSelectionFrame {
    machine: SiteSelectionMachine,
    req: SiteSelectionRequirement,
    domains: SiteSelectionDomains
}
{  machine.SiteSelectionMachine$domains = domains
   req.SiteSelectionRequirement$domains = domains
}

sig SiteSelectionDomains {
    // domains
    M: MetricMap,
    sourceFeature: set Feature,
    inputResource: set Resource,
    outputRegion: Region
}
{ inputResource in M.O
  outputRegion.P in M.P
}

sig SiteSelectionMachine {
   validSource: set Resource,
   domains: SiteSelectionDomains
}
{ validSource  = { r: Object | r in domains.inputResource &&
                               some r.F & domains.sourceFeature } }
```

```
sig SiteSelectionRequirement {
   domains: SiteSelectionDomains
}

//-------------------------------------------------------------------------
// SPECIFIC FRAME
// Find empty regions within a given distance
//-------------------------------------------------------------------------

sig EmptyRegionsWithinDistanceSiteSelectionFrame extends SiteSelectionFrame {
   dist: LargeValue
}
{ find_empty_regions(machine, dist) }

// Specific for finding points in the map that have distance from the
// the locations of the valid sources greater than zero and smaller than or
// equal to "dist".
// If the set of valid sources is empty, the output region is empty.
fun find_empty_regions(m: SiteSelectionMachine, dist: LargeValue) {
     m.domains.outputRegion.P = { p: m.domains.M.P |
        (some m.validSource) &&
        (  (m.domains.M in MetricDistanceMap
            && p.d[m.domains.M.Location[m.domains.M.O]] in Positive
            && (all p2: m.domains.M.Location[m.validSource] |
                      OrdLE(p.d[p2], dist)
               )
           )
         || (m.domains.M in MetricTaxicabMap
           && (all p2: m.domains.M.Location[m.domains.M.O] |
                 taxicabDistance(p, p2) in Positive
               )
           && (all p2: m.domains.M.Location[m.validSource] |
                     OrdLE(taxicabDistance(p, p2), dist )
               )
           )
        )
     }
}
```

```
// specific requirement
// Check if all points inside the region r are empty and within "dist"
fun empty_regions_requirement (r: SiteSelectionRequirement,
                               dist: LargeValue) {
   all p: r.domains.M.P |
        p in r.domains.outputRegion.P iff (
           (some r.domains.inputResource.F & r.domains.sourceFeature) &&
           (all res: r.domains.M.O |
                (r.domains.M in MetricDistanceMap &&
                     p.d[r.domains.M.Location[res]] != Zero) ||
                (r.domains.M in MetricTaxicabMap &&
                     taxicabDistance(p, r.domains.M.Location[res]) != Zero)
           ) &&
           (all res: r.domains.inputResource |
                (some r.domains.sourceFeature & res.F) =>
                ((r.domains.M in MetricDistanceMap &&
                     OrdLE(p.d[r.domains.M.Location[res]], dist)) ||
                 (r.domains.M in MetricTaxicabMap &&
                     OrdLE(taxicabDistance(p, r.domains.M.Location[res]),
                           dist))
                )
           )
        )
}

//-------------------------------------------------------------------------
// ASSERTIONS
//-------------------------------------------------------------------------

// Frame concern: The domains are included in the frame, and a fact in the
// frame makes the machine domains equal to the requirement domains
// S ^ D => R
assert SiteSelectionFrameConcern {
    all f: EmptyRegionsWithinDistanceSiteSelectionFrame, d: LargeValue |
          find_empty_regions(f.machine, d)
                => empty_regions_requirement(f.req, d)
}
```

D.6 Resource topology problem frame

This section presents the generic framework for the resource topology problem frame as well as machine specifications and requirements specific for: finding region resources (D2) that fully contain the input resources (D0), finding the resources (D0 or D1) that are inside the input resource (D2), finding the region resources (D2) that are crossed by the input resource (D1). The map used in this problem frame follows the nine-intersection projection presented in Section 7.2.4.

```
module geoframes/TopologyFrame

open map/nineintersection
open map/map

//-------------------------------------------------------------------------
// GENERAL SIGNATURES
//-------------------------------------------------------------------------

sig ResourceTopologyFrame {
    machine: TopologyMachine,
    req: TopologyRequirement,
    domains: TopologyDomains
}
{  machine.TopologyMachine$domains = domains
   req.TopologyRequirement$domains = domains
}

sig TopologyDomains {
    // domains
    M: NineIntersectionMap,
    sourceFeature, desiredFeature: set Feature,
    inputResource, outputResource: set Resource
}
{ inputResource + outputResource in M.0 }

sig TopologyMachine {
   validSource: set Resource,
   validTarget: set Resource,
   domains: TopologyDomains
}
// The facts below represent the weakest constraint over the features that
// the source and target resources must have. In this case, this constraint
// is that the valid resources must have at least one of the desired features.
// Stronger constraints may be added to specific machines.
{  validSource = { r: Object | r in domains.inputResource &&
                               (some r.F & domains.sourceFeature) }
   validTarget = { r: Object | r in domains.M.0 &&
                               (some r.F & domains.desiredFeature) }
}
```

```
sig TopologyRequirement {
   domains: TopologyDomains
}
{
   // proximal resources are in the map
   domains.outputResource in domains.M.O
   // resources have the desired features
   all r: domains.outputResource | some r.F & domains.desiredFeature
}

//-------------------------------------------------------------------
// SPECIFIC FRAME
// This frame finds the region resources (D2) that fully contain the
// input resources (D0).
//-------------------------------------------------------------------

sig FindEnclosingRegion extends ResourceTopologyFrame { }
{  domains.inputResource.G in D0
   domains.outputResource.G in D2
   find_enclosing_region(machine)
}

fun find_enclosing_region(m: TopologyMachine) {
   m.domains.outputResource = { r: m.validTarget | some p: m.validSource
                                                 | ObjectInRegion(p.G, r.G) }
}

fun find_enclosing_region_requirement(req: TopologyRequirement) {
   all r: req.domains.outputResource | {
       some r.F & req.domains.desiredFeature
       some p: req.domains.inputResource | {
           some p.F & req.domains.sourceFeature
           ObjectInRegion(p.G, r.G)
       }
   }
   no r: req.domains.M.O - req.domains.outputResource | {
       some r.F & req.domains.desiredFeature
       some p: req.domains.inputResource | {
           some p.F & req.domains.sourceFeature
           ObjectInRegion(p.G, r.G)
       }
   }
}
```

```
//-------------------------------------------------------------------------
// SPECIFIC FRAME
// This frame finds the resources (D0, or D1) that are inside the input
// resource (D2)
//-------------------------------------------------------------------------

sig FindResourcesInRegion extends ResourceTopologyFrame { }
{  domains.inputResource.G in D2
   find_resources_in_region(machine)
}

fun find_resources_in_region(m: TopologyMachine) {
   m.domains.outputResource = { p: m.validTarget | some r: m.validSource
                                                 | ObjectInRegion(p.G, r.G) }
}

fun find_resources_in_region_requirement(req: TopologyRequirement) {
   all p: req.domains.outputResource | {
       some p.F & req.domains.desiredFeature
       some r: req.domains.inputResource | {
           some r.F & req.domains.sourceFeature
           ObjectInRegion(p.G, r.G)
       }
   }
   no p: req.domains.M.O - req.domains.outputResource | {
       some p.F & req.domains.desiredFeature
       some r: req.domains.inputResource | {
           some r.F & req.domains.sourceFeature
           ObjectInRegion(p.G, r.G)
       }
   }
}

//-------------------------------------------------------------------------
// SPECIFIC FRAME
// This frame finds the resources (D2) that are crossed by the input
// resource (D1).
//-------------------------------------------------------------------------

sig FindCrossedRegions extends ResourceTopologyFrame { }
{  domains.inputResource.G in D1
   domains.outputResource.G in D2
   find_crossed_regions(machine)
}

fun find_crossed_regions(m: TopologyMachine) {
   m.domains.outputResource = { r: m.validTarget |
                     some l: m.validSource | LineCrossesRegion(l.G, r.G) }
}
```

```
fun find_crossed_regions_requirement(req: TopologyRequirement) {
   all r: req.domains.outputResource | {
       some r.F & req.domains.desiredFeature
       some l: req.domains.inputResource | {
           some l.F & req.domains.sourceFeature
           LineCrossesRegion(l.G, r.G)
       }
   }
   no r: req.domains.M.O - req.domains.outputResource | {
       some r.F & req.domains.desiredFeature
       some l: req.domains.inputResource | {
           some l.F & req.domains.sourceFeature
           LineCrossesRegion(l.G, r.G)
       }
   }
}

//-------------------------------------------------------------------------
// ASSERTIONS
//-------------------------------------------------------------------------

// Frame concerns: The domains are included in the frame, and a fact in the
// frame makes the machine domains equal to the requirement domains
// S ^ D => R

assert FindEnclosingRegionFrameConcern {
    all f: FindEnclosingResource |
          find_enclosing_region(f.machine) =>
               find_enclosing_region_requirement(f.req)
}

assert FindResourcesInRegionFrameConcern {
    all f: FindResourcesInRegion |
          find_resources_in_region(f.machine) =>
               find_resources_in_region_requirement(f.req)
}

assert FindCrossedRegionsFrameConcern {
    all f: FindCrossedRegions |
          find_crossed_regions(f.machine) =>
               find_crossed_regions_requirement(f.req)
}
```

D.7 Routing problem frame

This section presents the generic framework for the routing problem frame. It also provides a machine specification and requirement specific for finding a route from a given node to another node in the network that has the desired feature. The resulting route only contains one such node that has this feature. The map used in this problem frame follows the network projection presented in Section 7.2.4.

```
module geoframes/RoutingFrame
open map/network
open map/map
//-------------------------------------------------------------------------
// GENERAL SIGNATURES
//-------------------------------------------------------------------------

sig RoutingFrame {
    machine: RoutingMachine,
    req: RoutingRequirement,
    domains: RoutingDomains
}
{  machine.RoutingMachine$domains = domains
   req.RoutingRequirement$domains = domains
}

sig RoutingDomains {
    // domains
    M: NetworkMap,
    sourceFeature: set Feature,
    nodeResource: set Node,
    route: option UndirectedNetwork,
    desiredFeature: set Feature
}
{ nodeResource in M.Net.nodes
  route.nodes in M.Net.nodes
  route.edges in M.Net.edges
  M.Net in UndirectedNetwork
}

sig RoutingMachine {
   validSourceNodes, validTargetNodes: set Node,
   domains: RoutingDomains
}
{  validSourceNodes = { n: Node | n in domains.nodeResource &&
            (no domains.sourceFeature || some n.F & domains.sourceFeature) }
   validTargetNodes = { n: Node | n in domains.M.Net.nodes &&
            (no domains.desiredFeature || some n.F & domains.desiredFeature) }
}
```

```
sig RoutingRequirement {
   domains: RoutingDomains
}
{  // some node in the route must have the desired features
   some domains.desiredFeature =>
         (some n: domains.route.nodes | some n.F & domains.desiredFeature)
}

//-------------------------------------------------------------------------
// SPECIFIC FRAME
// This frame finds a route from a given node to another node in
// the network that has the desired feature. The route will only
// contain one such node that has this feature.
//-------------------------------------------------------------------------

sig FindRouteFromGivenNodeToNodeWithFeatureRoutingFrame
                                                    extends RoutingFrame { }
{ some domains.desiredFeature
  no domains.sourceFeature & domains.desiredFeature
  find_route(machine)
}

// Specific for finding a route from a starting node given in
// domains.nodeResource to a node that contains the desired feature
fun find_route(m: RoutingMachine) {
   // Precondition: there is only one starting node
   one m.validSourceNodes

   // Find the adjacency relation for the route
   // The relation is a subset of the map's adjacency relation
   m.domains.route.adj in m.domains.M.Net.adj
   // Find the nodes that have a desired feature. This
   // is the node where the route terminates, or the "end" node
     let end = { n : Node | n in m.validTargetNodes &&
                 no n.F & m.validSourceNodes.F &&
                 n in m.domains.route.adj[Node] } - m.domains.nodeResource | {
         // There is only one such node in the route
         some end & m.domains.route.nodes => (
                 one end & m.domains.route.nodes
                 // Every node in the map (except the first and last nodes
                 // in the route) is either not adjacent to any node in the
                 // route (that is, it's not part of the route), or is
                 // adjacent to exactly two other nodes in the route (that
                 // is, it's in the middle of the route).
                 && (all n: Node - m.validSourceNodes - end |
                           no m.domains.route.adj[n] ||
                           two m.domains.route.adj[n]
                    )
```

```
 // The end node can be reached from every node in the route
                && (all n: m.domains.route.nodes |
                                        end in n.*(m.domains.route.adj))
                // Only one node is adjacent to the end node
                && one m.domains.route.adj[end]
                // Only one node is adjacent to the starting node
                && one m.domains.route.adj[m.validSourceNodes]
            )
         no end & m.domains.route.nodes => no m.domains.route
   }

   // The nodes in the route are those in the route's adjacency relation
   m.domains.route.nodes = m.domains.route.adj[Node]
}

// Requirements for the specific routing problem
// Specific for finding a route from a starting node given in
// domains.nodeResource to a node that contains the desired feature
fun routing_requirement(r: RoutingRequirement) {
   // Find the node in the input that have a source feature - the start node
   let sourcenode = { n: r.domains.nodeResource |
                     (no r.domains.sourceFeature ||
                      some n.F & r.domains.sourceFeature) } |
   // Find the nodes in the solution that have a desired feature
   let targetnode = { n: r.domains.route.nodes |
                      some n.F & r.domains.desiredFeature &&
                      no n.F & sourcenode.F } - r.domains.nodeResource | {
   some targetnode => ( {
     // There can be only one target node
     one targetnode
     // The starting node must be in the route
     sourcenode in r.domains.route.nodes
     all n: r.domains.route.nodes | {
        // If a node in the route is the targetnode or the starting node,
        // it must have exactly one adjacent node in the route. Otherwise,
        // it must have exactly two adjacent nodes.
        n = targetnode || n = sourcenode =>
             (one r.domains.route.adj[n]),
             (two r.domains.route.adj[n])
        // It must be possible to reach the targetnode from every node
        // in the route
        targetnode in n.*(r.domains.route.adj)
        // It must be possible to reach the starting node from every node
        // in the route
        sourcenode in n.*(r.domains.route.adj)
     }
   } )
   no targetnode => no r.domains.route
   }
```

```
   // The route must be a subset of the network in the map
   r.domains.route.nodes in r.domains.M.Net.nodes
   r.domains.route.edges in r.domains.M.Net.edges
}

//-------------------------------------------------------------------------
// ASSERTIONS
//-------------------------------------------------------------------------

// Frame concern: The domains are included in the frame, and a fact in the
// frame makes the machine domains equal to the requirement domains
// S ^ D => R
assert RoutingFrameConcern {
    all f: FindRouteFromGivenNodeToNodeWithFeatureRoutingFrame |
          find_route(f.machine) => routing_requirement(f.req)
}
```

D.8 Spatial definition problem frame

This section presents the generic framework for the spatial definition problem frame as well as machine specification and requirements specific for determining the region that encloses the given regions. The map used in this problem frame follows the general definition of a map since the specific machine specification and requirements may refer to any type of projection and map phenomane to describe the new defined region. In the specific problem presented here the nine-intersection projection is used.

```
module geoframes/SpatialDefinitionFrame

open map/nineintersection
open map/map

//-------------------------------------------------------------------------
// GENERAL SIGNATURES
//-------------------------------------------------------------------------

sig SpatialDefinitionFrame {
    machine: SpatialDefinitionMachine,
    req: SpatialDefinitionRequirement,
    domains: SpatialDefinitionDomains
}
{  machine.SpatialDefinitionMachine$domains = domains
   req.SpatialDefinitionRequirement$domains = domains
}

sig SpatialDefinitionDomains {
    // domains
    M: Map,
    inputRegion: set Region,
    outputRegion: Region
}
{ (inputRegion + outputRegion).P in M.P }

sig SpatialDefinitionMachine {
   domains: SpatialDefinitionDomains
}


sig SpatialDefinitionRequirement {
   domains: SpatialDefinitionDomains
}
```

```
//-----------------------------------------------------------------------
// SPECIFIC FRAME
// Determines the region that encloses the given regions
// Similar to a "union" operation
//-----------------------------------------------------------------------

sig TopologicRegionUnionFrame extends SpatialDefinitionFrame { }
{  domains.inputRegion + domains.outputRegion in TopologicRegion
   domains.inputRegion.G in D2
   domains.outputRegion.G in D2
   // Precondition: parts are disjoint
   all p1, p2: domains.inputRegion.G | p1 != p2 => p1 !in p2.(II + BI)
   find_region_union(machine)
}


fun find_region_union(m: SpatialDefinitionMachine) {
   m.domains.outputRegion.G = BoundingRegion(m.domains.inputRegion.G)
   m.domains.outputRegion.P = m.domains.inputRegion.P
}


fun find_region_union_requirement(req: SpatialDefinitionRequirement) {
    IsBoundingRegion(req.domains.inputRegion.G, req.domains.outputRegion.G)
    all p: req.domains.outputRegion.P |
         some r: req.domains.inputRegion |
              p in r.P
    all p: req.domains.M.P - req.domains.outputRegion.P |
         no r: req.domains.inputRegion |
              p in r.P
}

//-----------------------------------------------------------------------
// ASSERTIONS
//-----------------------------------------------------------------------

// Frame concern: The domains are included in the frame, and a fact in the
// frame makes the machine domains equal to the requirement domains
// S ^ D => R
assert TopologicRegionUnionFrameConcern {
    all f: TopologicRegionUnionFrame |
          find_region_union(f.machine) =>
               find_region_union_requirement(f.req)
}
```

Bibliography

[ACG96] Joanne M. Atlee, Marsha Chechik, and John D. Gannon. Using model checking to analyze requirements and designs. *Advances in Computers*, 43:141–178, 1996.

[ACG+97] P.S.C. Alencar, D.D. Cowan, T.R. Grove, C.I. Mayfield, and M.A.V. Nelson. An approach to hypermap-based applications. In *Environmental Software Systems / Proceedings of the Second International Symposium on Environmental Software Systems – ISESS'97*, volume 2, pages 244–251, Whistler, B.C., Canada, April 1997.

[Ant96] Annie Antón. Goal-based requirements analysis. In *Proceedings of the Second IEEE International Conference on Requirements Engineering(ICRE'96)*, pages 136–144, 1996.

[BBdB+] Ren Bal, Herman Balsters, Rolf A. de By, Alexander Bosschaart, Jan Flokstra, Maurice van Keulen, Jacek Skowronek, and Bart Termorshuizen. The TM manual - version 2.0 revision e.

[BCMM96] Y. Bédard, C. Caron, Z. Maamar, and B. Moulin. Adapting data models for the design of spatio-temporal databases. *Computers, Environment, and Urban Systems*, 20(1), 1996.

[BDL01] Karla A. V. Borges, Clodoveu A. Davis, and Alberto H. F. Laender. OMT-G: An object-oriented data model for geographic applications. *GeoInformatica*, 5(3):221–260, 2001.

[Béd99] Y. Bédard. Visual modelling of spatial databases: Towards spatial PVL and UML. *Geomatica*, 53(2):169–186, 1999.

[Ber01] Daniel M. Berry. More requirements engineering adventures with building contractors. Submitted to Requirements Engineering Journal, 2001.

[Ber02] Daniel M. Berry. Formal methods: the very idea — Some thoughts about why they work when they work. *Science of Computer Programming*, 42(1):11–27, 2002.

[BKNS97] Dines Bjørner, Souleimane Koussobe, Roger Noussi, and Georgui Satchok. Michael Jackson's problem frames: Towards methodological principles of selecting and applying formal software development techniques and tools. In Li ShaoQi and Michael Hinchley, editors, *Proceedings of ICFEM'97: Intl. Conf. on "Formal Engineering Methods"*, pages 263–271, Hiroshima, Japan, 1997. IEEE Computer Society Press.

[BLD99] Karla A. V. Borges, Alberto H. F. Laender, and Clodoveu A. Davis. Spatial data integrity constraints in object oriented geographic data modeling. In *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA*, pages 1–6. ACM, 1999.

[Boa99] Mars Climate Orbiter Mishap Investigation Board. Mars Climate Orbiter, Phase I Report. Jet Propulsion Laboratory, California Institute of Technology, NASA, USA, 1999. `ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf`.

[Bra01] Marshall Brain. *How Stuff Works*. Hungry Minds, 2001.

[BRJ98] Grady Booch, James Rumbaugh, and Ivar Jacobson. *Unified Modeling Language User Guide*. Addison-Wesley, 1998.

[Cal96] Hugh Calkins. Entity relationship modeling of spatial data for geographic information systems. *International Journal of Geographical Information Systems*, 10(1), 1996.

[Cam98] John Campbell. *Map Use & Analysis*. WCB/McGraw-Hill, USA, 1998.

[Car01] John Carnes. Using your GPS with the Universal Transverse Mecartor Map Coordinate System. Map Tools, Woodside, California, 2001.

[CFvO93] Eliseo Clementini, Paolino Di Felice, and Peter van Oosterom. A small set of formal topological relationships suitable for end-user interaction. In David J. Abel and Beng Chin Ooi, editors, *Advances in Spatial Databases, Third International Symposium, SSD'93, Singapore, June 23-25, 1993, Proceedings*, volume 692 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1993.

[Chr97] Nicholas Chrisman. *Exploring Geographic Information Systems*. John Wiley & Sons, Inc., USA, 1997.

[CK96] Edmund M. Clarke and Robert P. Kurshan. Computer-aided verification. *IEEE Spectrum*, 33(6):61–67, 1996.

[CMTG98] D. D. Cowan, C. I. Mayfield, F. W. Tompa, and W. Gasparini. New role for community networks. *Communications of the ACM*, 41(4):61–63, 1998.

[Cor79] J. P. Corbett. Topological principles in cartography. Technical report, Technical Paper 48, US Bureau of the Census, Washington D.C., 1979.

[CY91] Peter Coad and Edward Yourdon. *Object-Oriented Analysis*. Prentice Hall, London, 2nd edition, 1991.

[Dan95] Peter H. Dana. Map Projection Overview, The Geographer's Craft Project. Department of Geography, University of Colorado at Boulder, USA, 1995. `http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj.html`.

[Dan97a] Peter H. Dana. Coordinate Systems Overview, NCGIA Core Curriculum in GIScience. Department of Geography, University of Texas at Austin, USA, 1997. `http://www.ncgia.ucsb.edu/giscc/units/u013/u013.html`.

[Dan97b] Peter H. Dana. The Shape of the Earth, NCGIA Core Curriculum in GIScience. Department of Geography, University of Texas at Austin, USA, 1997. `http://www.ncgia.ucsb.edu/giscc/units/u015/u015.html`.

[Dav93] Alan M. Davis. *Software Requirements: objects, functions and states*. Prentice-Hall International, Englewood Cliffs, NJ, 1993.

[Dij76] Edgar W. Dijkstra. *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs, N.J., 1976.

[EC02] S. M. Easterbrook and M. Chechik. Guest editorial: Special issue on model checking in requirements engineering. *Requirements Engineering Journal*, 7(4):221–224, December 2002.

[EF91] Max J. Egenhofer and Robert D. Franzosa. Point-set topological spatial relations. *International Journal of Geographical Information Systems*, 5(2):161–174, 1991.

[EH91] Max J. Egenhofer and John Herring. Categorizing binary topological relationships between regions, lines, and points in geographic databases. Technical report, University of Maine, Department of Surveying Engineering, 1991.

[EM95] M. J. Egenhofer and D. M. Mark. Naive geography. *Lecture Notes in Computer Science*, 988:1–15, 1995.

[EN96] S. Easterbrook and B. Nuseibeh. Using viewpoints for inconsistency management. *Software Engineering Journal*, 11(1):31–43, 1996.

[FCTJ01] Anders Friis-Christensen, Nectaria Tryfona, and Christian S. Jensen. Requirements and research issues in geographic data modeling. In *Proceedings of the Ninth ACM International Symposium on Advances in Geographic Information Systems*, pages 2–8. ACM Press, 2001.

[FI99] Jugurta Lisboa Filho and Cirano Iochpe. Specifying analysis patterns for geographic databases on the basis of a conceptual framework. In *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA*, pages 7–13. ACM, 1999.

[Fis91] P.F. Fisher. Spatial data sources and data problems. In D. J. Maguire, M. F. Goodchild, and D. W. Rhind, editors, *Geographical Information Systems: principles and applications*, pages 9–20. Longman Scientific & Technical, London, 1991.

[FS96] A. Finkelstein and I. Sommerville. The viewpoints FAQ. *Software Engineering Journal*, 11(1), 1996.

[GGJZ00] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, 17(3):37–43, May/June 2000.

[GN02] Vincenzo Gervasi and Bashar Nuseibeh. Lightweight validation of natural language requirements. *Software Practice and Experience*, 32(2):113–133, 2002.

[Goo92] Michael F. Goodchild. Geographical data modeling. *Computers and Geosciences*, 18(4):401–408, 1992.

[GR93] O. Günther and W. F. Riekert. The Design of GODOT: An Object-oriented Geographic Information System. *IEEE Data Engineering Bulletin*, 16(3):4–9, 1993.

[Gro93] CORPORATE The RAISE Language Group. *The RAISE specification language.* Prentice-Hall, Inc., 1993.

[Güt94] Ralf Hartmut Güting. GraphDB: Modeling and querying graphs in databases. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile*, pages 297–308, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[GW91] D. C. Gause and G. M. Weinberg. *Exploring Requirements: Quality Before Design.* Dorset House, New York, 1991.

[Ham01] David Hamil. Your Mission, Should You Choose To Accept It: Project Management Excellence. Spatial News, Geo Community, 2001. `http://spatialnews.geocomm.com/features/mesa1/hamil1.pdf`.

[HJL96] Constance L. Heitmeyer, Ralph D. Jeffords, and Bruce G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, 1996.

[HJL+02] Jon G. Hall, Michael Jackson, Robin C. Laney, Bashar Nuseibeh, and Lucia Rapanotti. Relating software requirements and architectures using problem frames. In *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, 2002.

[Hoa85] C. A. R. Hoare. *Communicating sequential processes.* Prentice-Hall, Inc., 1985.

[HT97] Thanasis Hadzilacos and Nectaria Tryfona. An extended entity-relationship model for geographic applications. *SIGMOD Record*, 26(3):24–29, 1997.

[Ile02] Jerry Iles. Why some watershed plans can end up on the shelf. Buckeye Basins Newsletter, Ohio State University Extension, USA, Summer 2002. `http://east.osu.edu/anr/Buckeye%20Basins%20Summer%2002.htm#WSplans`.

[Inf96] Association For Geographic Information. AGI GIS dictionary. Department of Geography, University Of Edinburgh, Scotland, 1996. `http://www.geo.ed.ac.uk/agidict/`.

[Jac] Daniel Jackson. Micromodels of software: Lightweight modelling and analysis with alloy. Software Design Group, MIT Lab for Computer Science. `http://sdg.lcs.mit.edu/alloy/book.pdf`.

[Jac95a] Daniel Jackson. Structuring z specifications with views. *ACM Transactions on Software Engineering and Methodology*, 4(4):365–389, 1995.

[Jac95b] Michael Jackson. *Software Requirements & Specifications - a lexicon of practice, principles and prejudices.* ACM Press and Addison Wesley, New York, 1995.

[Jac95c] Michael Jackson. The world and the machine. In *Proceedings: 17th International Conference on Software Engineering*, pages 283–292. IEEE Computer Society Press / ACM Press, 1995.

[Jac98] Michael A. Jackson. A discipline of description. *Requirements Engineering*, 3(2):73–78, 1998.

[Jac99] Michael Jackson. Problem analysis using small problem frames. *South African Computer Journal*, (22):47–60, March 1999.

[Jac00a] Daniel Jackson. Automating first-order relational logic. In *Proceedings of the ACM SIGSOFT Conference on Foundations of Software Engineering*, San Diego, November 2000.

[Jac00b] Michael Jackson. *Problem Frames - Analyzing and structuring software development problems.* ACM Press and Addison Wesley, New York, USA, 2000.

[Jon90] Cliff B. Jones. *Systematic Software Development using VDM.* Prentice-Hall, Upper Saddle River, NJ 07458, USA, 1990.

[JSS01] Daniel Jackson, Ilya Shlyakhter, and Manu Sridharan. A micromodularity mechanism. In *Proceedings of the ACM SIGSOFT Conference on Foundations of Software Engineering*, Vienna, Austria, September 2001.

[JZ93] Michael Jackson and Pamela Zave. Domain descriptions. In *Proceedings of the IEEE International Symposium on Requirements Engineering, Re'93*, pages 56–64. IEEE Computer Society Press, January 1993.

[JZ95] Michael Jackson and Pamela Zave. Deriving specifications from requirements: an example. In *Proceedings: 17th International Conference on Software Engineering*, pages 15–24. IEEE Computer Society Press / ACM Press, 1995.

[KBP01] E. Kamsties, D.M. Berry, and B. Paech. Detecting ambiguities in requirements documents using inspections. In *Proceedings of Workshop on Inspections in Software Engineering (WISE'01)*, pages 68–80, Paris, France, 2001. Software Quality Research Lab, McMaster University, Hamilton, Canada.

[Kem93] Karen K. Kemp. Environmental modeling with GIS: A strategy for dealing with spatial continuity. Technical Report 3, National Center for Geographic Information and Analysis, University of California at Santa Barbara, California, 1993.

[Ken89] Martin S. Kenzer. *On Becoming a Professional Geographer*. Blackburn Press, New Jersey, 1989.

[KLM+97] Gregor Kiczales, J. Lamping, A. Mendhekar, C. Lopes, J. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of ECOOP '97*, 1997.

[Kov98] Benjamin Kovitz. *Practical Software Requirements - a manual of content and style*. Manning Publications Co., 1998.

[KP88] G. E. Krasner and S. T. Pope. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. *Journal of Object-Oriented Programming*, 1(3):26–49, 1988.

[KPS96] Georg Kösters, Bernd-Uwe Pagel, and Hans-Werner Six. GeoOOA: Object-Oriented Analysis for Geographic Information Systems. In *Proceedings of the Second International Conference on Requirements Engineering, Colorado*, pages 245–253. IEEE Computer Society Press, April 1996.

[KPS97] G. Kosters, B.-U. Pagel, and H.-W. Six. GIS-application development with GeoOOA. *International Journal of Geographical Information Science*, 11(4):307–335, 1997.

[Kra75] Eugene F. Krause. *Taxicab Geometry*. Addison-Wesley, 1975.

[Mag91] D. J. Maguire. An Overview and Definition of GIS. In D. J. Maguire, M. F. Goodchild, and D. W. Rhind, editors, *Geographical Information Systems: principles and applications*, pages 9–20. Longman Scientific & Technical, London, 1991.

[Mai98] N. A. M. Maiden. CREWS-SAVRE: Scenarios for acquiring and validating requirements. *Automated Software Engineering: An International Journal*, 5(4):419–446, October 1998.

[Mal92] D. H. Maling. *Coordinate Systems and Map Projections*. Pergamon Press, Oxford, England, second edition, 1992.

[Map96] Inc. MapQuest.com. MapQuest, 1996. `http://www.mapquest.com`.

[Map00] Mapcheck. The Region of Waterloo Information Utility, 2000. `http://regionofwaterloo.mapcheck.com/`.

[Mar93] David M. Mark. Human spatial cognition. In David Medyckyj-Scott and Hilary M. Hearnshaw, editors, *Human Factors in Geographical Information Systems*, pages 51–60. Belhaven Press, London, UK, 1993.

[Mar98] D. M. Mark. Spatial representation: a cognitive view. In D. J. Maguire, M.F. Goodchild, D. Rhind, and P. Longley, editors, *Geographical Information Systems: Principles, Techniques, Management and Applications*, volume 1, pages 81–89. Geoinformation International, second edition, 1998.

[MK95] R. McDonnell and K. Kemp. *International GIS Dictionary*. Geoinformation International, Cambridge, 1995.

[MM98] Phillip C. Muehrcke and Juliana O. Muehrcke. *Map Use - Reading, Analysis and Interpretation*. JP Publications, fourth edition, 1998.

[NAC01] M.A.V. Nelson, P.S.C. Alencar, and D.D. Cowan. An approach to formal specification and verification of map-centered applications. *Environmental Modelling and Software*, 16(5):459–465, 2001.

[Nav92] Shamkant B. Navathe. Evolution of data modeling for databases. *Communications of the ACM*, 35(9):112–123, 1992.

[NCA01] M. A. V. Nelson, D. D. Cowan, and P. S. C. Alencar. Geographic problem frames. In *Proceedings of the Fifth IEEE International Symposium on Requirements Engineering*, pages 306–307, 2001.

[NE00] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 35–46. ACM Press, 2000.

[Nel99] Maria A. V. Nelson. Multi-frame approach to formal requirements specification of geographical applications. In *Proceedings of the Doctoral Symposium of Fourth IEEE International Symposium on Requirements Engineering*, pages 31–34, 1999.

[Nus01] Bashar Nuseibeh. Weaving together requirements and architectures. *IEEE Computer*, 34(3):115–117, March 2001.

[OCCB90] Stan Openshaw, Anna Cross, Martin Charlton, and Chris Brunsdon. Lessons learnt from a post mortem of a failed GIS. In *Proceedings of the Association for Geographic Information Conference*. AGI, 1990.

[OMC96] Online map creation, 1996. `http://www.aquarius.geomar.de/omc/omc_intro.html`.

[OPM97] J. L. Oliveira, F. Pires, and C. M. B. Medeiros. An environment for modeling and design of geographic applications. *GeoInformatica*, 1(1):29–58, 1997.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, volume 607 of *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, June 1992. Springer-Verlag.

[PDA89] R. Prieto-Diaz and G. Arango. *Domain Analysis: Acquistion of Reusable Information for Software Construction*. IEEE Computer Society Press, 1989.

[PSZ99] Christine Parent, Stefano Spaccapietra, and Esteban Zimányi. Spatic-temporal conceptual models: Data structures + space + time. In *ACM-GIS '99, Proceedings of the 7th International Symposium on Advances in Geographic Information Systems, November 2-6, 1999, Kansas City, USA*, pages 26–33. ACM, 1999.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1991.

[RMM+95] Arthur H. Robinson, Joel L. Morrison, Phillip C. Muehrcke, A. Jon Kimerling, and Stephen C. Guptill. *Elements of Cartography*. John Wiley & Sons, Inc., sixth edition, 1995.

[SC96] Isabel Santos and João A. Carvalho. An assessment of the applicability of object-oriented analysis to the development of information systems. In *Proceedings: 2nd*

International Conference on Requirements Engineering, pages 238–244. IEEE Computer Society Press, 1996.

[SCG+97] Shashi Shekhar, Mark Coyle, Brajesh Goyal, Duen-Ren Liu, and Shyamsundar Sarkar. Data models in geographic information systems. *Communications of the ACM*, 40(4):103–111, April 1997.

[SMB85] Surveys and Geographical Referencing Section Mapping Branch. *The Ontario Geographical Referencing System (The Universal Transverse Mercator Grid System)*. Ministry of Natural Resources, Ontario, Canada, 1985.

[SNKR01] Juno Song, Ram Naguleswaran, Alan Kirker, and Mashuk Rahman. Map Datums: An Interactive Overview. Produced for the Mapping Analysis and Design Group by student affiliates and the Center for Learning and Teaching Through Technology, University of Waterloo, Canada, 2001. `http://www.pliant.ca/datums.swf`.

[Sny87] John P. Snyder. *Map Projections - A Working Manual*. United States Government Printing Office, Washington D.C., U.S.A., 1987.

[Sto86] D.R. Stoddart. *On Geography: And Its History*. Blackwell, Oxford, 1986.

[Sui98] Daniel Z. Sui. GIS-based urban modeling: practices, problems, and prospects. *International Journal of Geographical Information Science*, 12(7):651–671, 1998.

[Swa01] D.A. Swayne. Design principles for environmental information systems. *Environmental Modelling and Software*, 16(5):417, 2001.

[TJ99] N. Tryfona and C. S. Jensen. Conceptual data modeling for spatiotemporal applications. *GeoInformatica*, 3(3):245–268, 1999.

[TPH97] N. Tryfona, D. Pfoser, and T. Hadzilacos. Modeling behavior of geographic objects: An experience with the object modeling technique. In *Proceedings of the Ninth International Conference on Advanced Information Systems Engineering CAiSE'97*, pages 347–359, Barcelona,Spain, June 1997.

[vdAB91] Bert van den Akker and Henk M. Blanken. Geographic data modeling in TM. In P. Sadanandan and T. M. Vijayaraman, editors, *Advances in Data Management: Proceedings of the International Conference on Information Systems and Management of Data*, pages 107–126, Bombay, India, 1991. McGraw-Hill, New Delhi, India.

[vL00] Axel van Lamsweerde. Formal specification: a roadmap. In *Proceedings of the Conference on The Future of Software Engineering*, pages 147–159. ACM Press, 2000.

[vLLD98] Axel van Lamsweerde, Emmanual Letier, and Robert Darimont. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering*, 24(11):908–926, 1998.

[Wor95] Michael F. Worboys. *GIS: A Computing Perspective.* Taylor and Francis, 1995.

[WSC95] Wisconsin State Cartographer's Office, Madison, Wisconsin. *Wisconsin Coordinate Systems Handbook*, 1995.

[ZJ97] Pamela Zave and Michael Jackson. Four dark corners of requirements engineering. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(1):1–30, 1997.