The Pennsylvania State University

The Graduate School

College of Earth and Mineral Sciences

THE SEMANTIC GEOSPATIAL PROBLEM SOLVING ENVIRONMENT: AN ENABLING TECHNOLOGY FOR GEOGRAPHICAL PROBLEM SOLVING UNDER OPEN, HETEROGENEOUS ENVIRONMENTS

A Thesis in

Geography

by

Junyan Luo

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2007

The thesis of Junyan Luo was reviewed and approved* by the following:

Mark N. Gahegan
Professor of Geography
Thesis Adviser
Chair of Committee

Brent Yarnal
Professor of Geography

Frederico T. Fonseca
Assistant Professor of Information Sciences and Technology

Ian J. Turton
Research Associate of Geography

Roger M. Downs
Professor of Geography
Head of the Department of Geography

*Signatures are on file in the Graduate School.

ABSTRACT

This thesis presents a conceptual and computational framework for the Semantic Geospatial Problem Solving Environment, an enabling technology for geographical problem solving under today's open, heterogeneous environment. The framework adopts an open software architecture to integrate different geospatial information technologies, employs the dataflow-based visual programming interface to facilitate front-end user task construction, and most importantly, utilizes structurally represented semantic knowledge to assist and automate the construction of geographical applications. The major contribution of this thesis is the development of a semantic model of geographical problem solving, which synthesizes aspects of knowledge representation and reasoning to capture and model the meanings of geospatial information technologies and geographical applications. Three levels of semantics are addressed according to C. S. Peirce's theory of pragmatism, including the first level semantics about individual resources, the second level semantics about relations, and the third level semantics related to geographical applications. Correspondingly, the idea of proxy representation is introduced to signify the meanings of individual resources under different interpretation contexts, and the first order model theory is adopted to build the logical structures that connect individual resources together. Then automated problem solving tools are developed based on cognitive models of problem solving and automated planning techniques, which support three useful modes of reasoning in geographical application construction, including incremental problem solving, prototype refinement, and task decomposition. After that, the proposed semantic model is implemented based on the World Wide Web Consortium's Web Ontology Language to facilitate the sharing of semantics on the Semantic Web. Finally, a reference implementation is described to illustrate the concepts and ideas discussed in the thesis.

Keywords: *Geographical Problem Solving, Semantics, Automated Planning, Knowledge Representation, Problem Solving Environment, Semantic Geospatial PSE.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

AKNOWLEDGEMENTS

The completion of this thesis would be impossible without the help of many people. First of all, I would like to address special thanks to my advisor, Dr. Mark Gahegan, for his kind support, insightful advice, and enormous patience in improving my writing.

I would like to thank Dr. Fred Fonseca, whose original research has provided the initial motivation of this study.

I would like to express my gratitude to Dr. Brent Yarnal for his multi-disciplinary insights on human-environment relations, which have helped me remain focused on real problems.

I am especially grateful to Dr. Ian Turton for his kind assistance and generous help to me in all aspects of my work at the GeoVISTA center.

I also would like to thank Dr. David O'Sullivan for his comments and suggestions during the early stage of this study.

I owe tremendous gratitude to Dr. Alan MacEachren and Dr. Donna Peuquet for their guidance in various research projects I have participated in. Those projects have offered me invaluable research experience and many useful materials, without which this study would not have been possible.

I would like to thank all my current and past colleagues at Penn State, including Ritesh Agrawal, Ola Ahlqvist, Tawan Banchuen, Isaac Brewer, Aju Badardeen, Jin Chen, Xiping Dai, Jianwei Dou, Sven Fuhrmann, Diansheng Guo, Frank Hardisty, Scott Pezanowski, Bill Pike, Anthony Robinson, Kean Huat Soon, Stephen Weaver, Biliang Zhou, and many others not in the list, for the great time of working together.

Finally, my best thanks go to my family. My parents and brother are always offering me unconditional support and understanding throughout my life, even though they are far away. My fiancée, Guo Chen, has been always on my side during the years I spent at Penn State. She gave me the strength to finish this work.

Chapter 1

INTRODUCTION

1.1 Background

In today's information era, computer technologies have become not only more advanced and powerful but also more specialized and sophisticated. Most of today's computational techniques, including those related to computer hardware, software, and algorithms, tend to be understandable and accessible only to small groups of experts (Houstis and Rice 2002). As the demands for digital computing grows, this becomes a significant barrier to scientific research, which discourages the broader scientific communities from grasping those complex yet useful techniques, and precludes scientists who use them from sharing their work with others (Rice and Boisvert 2000).

In geography, this barrier can be quite severe due to the interdisciplinary nature of the subject, where different theories, expertise, and techniques are usually required to solve real-world problems. In many cases there is "a gap in knowledge between the abstract functioning of these (computational) tools … and their successful deployment to the complex applications and datasets that are commonplace in geography" (Gahegan 1998); this gap handicaps the spread and applications of GI (Geospatial Information) technologies, and increases the cost of geographical problem solving. For today's geographical information science (GISci), which aims to enhance geographical research with GI technologies (Goodchild 1992), it is necessary to develop appropriate "enabling technologies" for geographers to better utilize the power of digital computing.

The importance of enabling technologies to geography has been recognized for a relatively long while. In 1983, Dobson proposed his idea of "Automated Geography", which envisioned the development of an integrated computational infrastructure for

geographical problem solvers (Dobson 1983). Although regarded by many as a testimony for the rising technology of geographical information systems (GIS) at the time (Dobson 1993), the idea of Automated Geography was in fact not just an account of the particular technology of GIS, but also an agenda for computer-assisted geographical problem solving in general. As Dobson later pointed out, the increasing importance of digital information in all aspects of geography had created the necessity to develop a "coordination center" or "geographical information network" that should combine computer cartography, computer graphics, digital remote sensing, GIS, spatial statistics, quantitative spatial modeling, and various other technologies to help geographers solve problems within their domains of interest (Dobson 1985). By now the idea of Automated Geography has been experimented with and implemented in various GIS-based applications (Dobson 1993).

However, it should be recognized that the dominance of one technology (such as GIS) could restrict if not mislead geographical research, especially when the technology itself has considerable limitations (Gahegan 1998). The enabling infrastructure for geography, therefore, should combine all GI technologies rather than rely on a particular subset as most existing GIS packages do. With the latest advances in computer science and information technologies and the prevailing status of the Internet in social and academic life, the development of comprehensive problem solving infrastructures for geography presents both opportunities and challenges. On the positive side, advances in areas such as software engineering, distributed computing, database management systems, and the World Wide Web (WWW) have created the possibilities of direct connections between different data sources and software tools. Compared with the past, the technical barriers for geographers to build a geographical information network as suggested by Dobson are at least partly removed, and there is no excuse for us to be restricted to 'stove pipe' systems such as GIS, without fully exploiting the potential of other useful technologies. On the negative side, however, the increased needs of technology

integration have created the danger of widening the knowledge gap between geographers and digital computing, which has already been a big problem. In contrast to traditional software systems governed by centralized designs, information processing in today's technological environment must respect and utilize multiple data sources and software tools developed from different origins, potentially based on different conceptual models, and perhaps delivered over an open computer network. Even when crafted to work in a closed offline environment, a system must be flexible enough to analyze different kinds of datasets and support different stages of geographical inquiry (e.g. hypothesis generation, analytical modeling, and results presentation). In brief, the scope of potentially useful computational resources has dramatically expanded (and is still expanding), and how to make use of them in an integrated manner is a challenge for both domain geographers interested in technology use and GI scientists concerned with technology development.

The origins of the above-mentioned challenges can be traced back to the problem of *heterogeneity*, which refers to the conceptual and structural differences between information entities produced by different sources (Bishr 1998). Generally speaking, this issue can be divided into two separate but inter-connected problems: the first being *syntactic heterogeneity*, which describes the incompatibility of information entities at the data format level, and the second being *semantic heterogeneity*, which refers to disagreements at the conceptual level, i.e. the different ways various information entities conceptualize and represent realities (Heiler 1995).

Syntactical heterogeneity can be overcome by adopting appropriate data sharing standards and employing a proper software inter-operation platform. Data sharing standards specify the common formats of data exchanging between information sharing parties. For example, two well-known geospatial data sharing standards are SDTS (Spatial Data Transfer Standard) defined by USGS (The United States Geological Survey) (USGS 1998) and GML (Geography Markup Language) brokered by OGC (Open

Geospatial Consortium) (OGC 2004). On the other hand, a software inter-operation platform defines the protocols or interfaces between inter-operating software modules or services. The most widely used software inter-operation platforms include CORBA (Component Object Request and Brokerage Architecture) from OMG (Object Management Group) (OMG 1997), DCOM (Distributed Component Object Model) from Microsoft™ (Microsoft 1996), Java Beans from Sun Microsystems™ (Sun Microsystems 1997) and various software bus technologies such as the recently developed Web Service Architecture (W3C 2004).

In contrast, resolving the heterogeneities at the semantic level is not so straightforward. Semantics implies meaning, and the ultimate goal of semantic integration is to ensure that heterogeneous information entities can work together *harmoniously and meaningfully* (Heiler 1995). Obviously this cannot be simply done by syntactic data conversion or low-level system inter-operation. In addition to that, semantic integration requires the data exchange and software inter-operation among participating parties to be correct and meaningful in a high-level, conceptual sense. In order to achieve this, heterogeneous computational resources must be semantically mediated at the conceptual level, during which their meanings should be clearly articulated, correctly interpreted, and appropriately processed.

How to overcome the semantic heterogeneity among geospatial information entities has been intensively studied in GISci during the past decade, primarily motivated by the demands of information sharing among different proprietary GIS platforms in the market (Bishr 1997). Unfortunately, up to now the emphasis in this area has been largely placed on the integration of data, with little having been done to explore the sharing of analytical functionalities, despite a few scattered exceptions (Ungerer and Goodchild 2002; Crosier, Goodchild et al. 2003). Furthermore, the study of how to combine shared data and functionalities into complete working solutions seems to have stalled. For example, imagine a crisis manager who is planning to discover traveling patterns to

Washington DC during the past couple of years that may provide novel information about potential terrorist attacks. In this task, the analyst must connect to different data sources and apply different visualization and / or spatial data mining tools to identify the hidden clusters in the datasets. State-of-the-art technology in geospatial information sharing can be applied to retrieve information from different databases and convert it into the analyst's own operational data model. However, few methods are currently available to help the analyst (who is a crisis manager rather than a GIS expert) retrieve the most suitable tools for the given datasets, and configure them to accomplish the desired analysis work. In other words, the existing work answers the question of "what data is most suitable to my analysis and how can I get it?" but provides no suggestions regarding the issue about "what functionalities should I choose to analyze this dataset in this particular task, and how can I correctly configure and successfully use them?" This is a significant deficiency from the perspective of real world geographical problem solving, which limits the effectiveness and efficiency of the technologies we are currently using, and compromises our abilities to share the useful new tools we create.

The central purpose of this thesis is to build the conceptual and computational foundation for resolving the problem of semantic heterogeneity from an analytical perspective. To achieve this goal, the concept of a Semantic Geospatial PSE (Problem Solving Environment) is introduced by this study based on software development and application practices from scientific computing in general and GISci in particular. A Semantic Geospatial PSE is an open problem solving software system that models, manages, and processes the meanings of heterogeneous computational resources, including both data and analytical functionalities, and thus facilitates their semantic integration and senseful applications. It focuses on capturing the problem solving aspects of different GI technologies, including what high-level tasks computational resources can accomplish and how they should be integrated with each other to solve novel problems. Such kind of knowledge, referred to as "geographical problem solving semantics" in this

study, can be presented in high-level terms understood by geographical analysts, and at the same time connected back with the low-level implementation and runtime details recognized by digital computers.

Consider the crisis management example mentioned above, where an analyst attempts to analyze traveling patterns related to the area of Washington DC. In this application, the analyst's central interest is focused on exploring spatial-temporal trends of recorded traveling activities and analyzing how such trends can be used to monitor an anticipated security threat. Although the integration of data sources and analytical methods is an indispensable part of this task, from the analyst's point of view, knowing such technical details would help little in finding meaningful patterns in the data. In an open, heterogeneous environment where data sources and analytical functionalities are created by different individuals and organizations, the complicated technical details involved in integrating them may easily become a distracting factor to the analyst, whose primary concern is crisis analysis rather than tool integration. With the Semantic Geospatial PSE, the analyst can use intuitive high-level concepts from their familiar domains (e.g. spatial-temporal analysis) as a shell to retrieve heterogeneous resources (with which they are not necessarily familiar) and convert them into useful applications in a consistent, automated and transparent manner. As a result, he / she can focus on the intended analysis work instead of worrying about how to integrate heterogeneous resources correctly and meaningfully. The Semantic Geospatial PSE, therefore, offers a promising enabling technology for today's geography and GISci.

1.2 Research Question and Objectives

Modeling and processing geographical problem solving semantics must be based on synthesizing a wide range of studies, including those from software engineering, geospatial data handling, cognitive problem solving, AI (Artificial Intelligence), and knowledge representation. The general research question in this study is: *can elements*

from those diverse areas be woven together to create a computational platform, upon which geographical domain knowledge can be captured, modeled and represented to expedite the applications of heterogeneous computational resources? In other words, is there an essence of geographical problem solving lurking behind the technical details, and if so, can we integrate the aforementioned studies based on this essence to build a better enabling technology that hides the plethora of technical details from geographers and thus makes geographical problem solving much easier?

Specifically, the research objectives of this thesis can be divided into four levels, including those at the system level, the conceptual level, the automation level, and the representation level.

1. **System Level.** The main objective at the system level is to define the overall software architecture of a Geospatial PSE, identify its major components, and specify their roles. Based on that, a Semantic Geospatial PSE can be conveniently defined as a Geospatial PSE that handles geographical problem solving semantics.
2. **Conceptual Level.** At the conceptual level, it is necessary to clearly identify the semantic problems encountered in geographical problem solving, and carefully examine how meanings of heterogeneous computational resources should be modeled to resolve such problems. This study employs a semiotic-logical approach to analyze the issue of semantic heterogeneity. Based on that, a semantic model of geographical problem solving is developed to layout the conceptual foundation for capturing, modeling and processing the semantics of heterogeneous computational resources in a Semantic Geospatial PSE.
3. **Automation Level.** This level concerns the procedural nature of geographical problem solving and examines how such procedures can be modeled and automated in a computer. The central objective at this level is to develop computational methods that reason about geographical analytical

functionalities, and combine them with the semantic model described at the conceptual level to implement automated problem solving tools for a Semantic Geospatial PSE.

4. **Representation Level.** At the representation level, the main research question is how to incarnate the semantic model into a feasible knowledge representation that can be shared in an open, heterogeneous environment such as the Internet. With this in mind, this study defines a portable language based on OWL (Web Ontology Language), the W3C (World Wide Web Consortium) knowledge representation standard for the Semantic Web (McGuinness and Harmelen 2004), to represent and share knowledge created from the proposed semantic model.

1.3 Major Contributions

This thesis initiates a first step towards removing the technical details that obfuscate and hinder the analysis of geospatial information. The major contribution of this thesis is the development of an integrated framework that synthesizes a number of different approaches that can represent and reason with the semantics of geographical analysis within the context of geographical problem solving. It reviews the technical background of today's GISci and geographical problem solving, examines how geographers conceptualize and utilize computational resources to solve real problems, and studies how a problem solving system should be built to offer geographers easy and semantically meaningful access to the latest geospatial information technologies. The specific contributions can be categorized into four areas with respect to the four levels of objectives listed above.

Firstly, the concept of Semantic Geospatial PSEs provides a new technology to support geographical problem solving in an open, heterogeneous environment, which

allows geographical research to extend beyond the technical boundaries of traditional desktop GIS.

Secondly, this thesis creates a semantic model of geographical problem solving, which examines the roles of semantics in geographical applications, defines a solid framework to model geographical problem solving semantics, and extends the scope of semantic integration to the level of software functionalities.

Thirdly, this study examines and develops methods of automated geographical problem solving, which provides a novel and useful fusion of cognitive science, semantic studies, and AI under the contexts of geography and GISci.

Finally, this work creates a feasible knowledge representation strategy for geographical analysis, which supports knowledge sharing and processing over the Semantic Web. Combining the semantic model of geographical problem solving with various contemporary knowledge development methods, the representation framework outlines a knowledge-based methodology for geographical analysis, which can lead to various interesting future research directions. Note that the emphasis of this thesis is placed on the integration of analytical *methods* because semantic *data* integration, as mentioned before, has already been thoroughly studied in GISci. The development of method integration adds the final missing piece to the semantic integration of geospatial information, with which data and methods can be combined into end-to-end solutions that can be directly delivered to users.

1.4 Organization of the Thesis

This thesis first reviews the general picture of computational problem solving in both scientific computing and GISci, and then combines relevant studies in geography, philosophy, semantic studies, cognitive science, and artificial intelligence into a coherent conceptual and computational model of geographical problem solving. A wide swath of

literature needs to be introduced to cover both the general ideas invoked, and the specific details of representing and reasoning with semantics. Instead of combining all of this background into a single, rather disconnected, literature review, some aspects of the literature are introduced in their most relevant chapters, as needed (specifically in chapters 3, 4 and 5). Chapter descriptions follow:

1) Chapter one has described the necessary background information, states the main research problems, and outlines the specific objectives of this thesis.
2) Chapter two reviews the pros and cons of existing GI technologies (especially GIS), introduces the ideas of scientific PSEs, and presents a general design of a Geospatial PSE.
3) Chapter three analyzes the problems of semantic heterogeneity from a logical-semiotic perspective, discusses the role of explicitly specified semantics in resolving these problems, and presents a semantic model of geographical problem solving.
4) Chapter four discusses automated methods of geographical problem solving based on results from cognitive and AI studies.
5) Chapter five presents an OWL-based knowledge representation language to encode and share geographical problem solving semantics defined by the proposed semantic model.
6) Chapter six describes a reference implementation that demonstrates the development and application of a Semantic Geospatial PSE.
7) Finally, chapter seven contains concluding remarks and points out several directions of future research.

Chapter 2

GEOSPATIAL PSE: THE ENABLING TECHNOLOGY FOR GEOGRAPHICAL PROBLEM SOLVING

2.1 Introduction

Computer programs are the basic units for humans to utilize the computer's power, and software systems such as GIS have been the primary platforms for geographers and others to pursue digital computing. Due to the inter-disciplinary nature of geography, it is commonplace for geographical analysts to draw upon a wide range of software tools to address different issues arising from real world problem settings. Consequently, it is important to take an inclusive view to comb all computational technologies relevant to geographical problem solving, evaluate their pros and cons, and integrate useful past experience with the latest technology development to provide a framework that helps geographical research move forward.

Two categories of software technologies are reviewed in this chapter. The first includes the current generation of GI technologies, especially GIS, as they have been developed and applied as the *de facto* problem solving tools in geography and related fields. The second is a family of systems known as scientific problem solving environments (PSEs), which have been recently recognized by the general scientific computing community as the key enabling technology to scientific problem solving. The central purpose of these discussions is to identify the pros and cons of GI technologies from the perspective of GISci, and clarify why the idea of scientific PSEs is important to scientific research in general and geographical problem solving in particular. Accordingly, the final part of this chapter describes a blueprint for Geospatial PSEs,

which are essentially software systems engineered towards PSE standards and tailored for geospatial information and geographical purposes.

2.2 GI Technologies and Geographical Problem Solving: A GISci Perspective

During the past several decades, geospatially referenced information (or geospatial information in short) of various kinds has become a key resource for human analysts, including both academic researchers and practitioners from other fields, to observe, model, and understand different geospatial phenomena. GI technologies, including GIS, are a collection of computational tools that support the creation, management, manipulation, analysis, and presentation of geospatial information (Goodchild, Sheppard et al. 1999). The term "geographical problem solving", therefore, can be generally defined as any activities that use GI technologies to process geospatial information, with the purpose of attaining certain geographical application goals. This section reviews the relations of GIS, GI technologies, and GISci from a historical perspective, and then describes the challenges faced by existing GI technologies in supporting today's GISci and geographical problem solving.

2.2.1 The Origins of GIS

GIS is a family of information systems specially designed to handle geospatial information (Goodchild 2000). Consequently, strictly speaking GIS is only one kind of GI technologies (although the most common one). However, historically the most notable achievements of geospatial data handling have largely been made under the banner of GIS, and due to GIS's market prevalence and social impact, it is not uncommon for people to use the term "GIS" to denote the entire family of GI technologies (Goodchild, Sheppard et al. 1999). Therefore, understanding the origins of GIS can help us better comprehend the general picture of today's GI technologies.

The inception of GIS in late 1950s and early 1960s was a direct response to the emerging demands for geospatial data handling in both academic institutes and governmental agencies (Foresman 1998). The early years of GIS coincided with the period in geography known as the "quantitative revolution", when quantitative methods were introduced to geography and adopted by many geographers as the main approach of scientific inquiry. Such a trend created a new need for tools that could support digital information manipulation and data analysis, which inspired the development of many useful computational methods and algorithms. Joined by geospatial interests from other fields, including computer science, urban planning, environmental studies, and geology, those early geospatial data handling techniques constituted a multidisciplinary intellectual foundation for the fledging GIS technology in the academic sector (Chrisman 1998). In the mean time, as computers began to show potential in processing large amounts of digital information, practitioners outside universities, especially policy makers in government agencies, started to search for information systems that could facilitate the management of geospatial data originating from various application settings. For example, the Canadian GIS (CGIS), which is regarded by many as the first prototypical GIS in North America, was originally built to support the Canadian government's national land-use and land inventory management program (Tomlinson 1967; Tomlinson 1968). In the United States, the increasing concerns about demographic census, natural resource monitoring, and land-use management in the 1960s and 1970s motivated a significant amount of system development and application efforts in both government agencies and private companies (Behrens, Moyer et al. 1974; Hoffer and staff 1975; Marx 1986). Similar trends can be observed in other continents around the world (Garner and O'Callaghan 1998; Rhind 1998). The software systems produced by those practices became the prototypes of the first generation GIS.

2.2.2 The Formalization of GIS and Its Limitations

Since the middle 1970s, the development of GIS entered a formalization period, during which most key techniques in GIS were stabilized, and software systems such as Arc/Info became mature and gradually evolved into the prevailing platforms in the market. The academic contributions to the formalization of GIS were remarkable, having pushed forward advances in critical areas such as geographical data modeling, spatial data indexing, topological spatial relations, and digital cartography (Goodchild 1992). In addition, the wide spread of GIS inspired the idea of computational geography, which regarded digital computation not only as a tool, but also as a new methodology of inquiry (Openshaw 1994; Macmillan 1997).

Unfortunately, academic researchers were never able to compete with government and commercial sectors in terms of developmental resources. As a result, the general shape of GIS was driven by larger market forces and broader application needs, even though they often did not meet the exact demands of academic researchers. From a geographer's point of view, the most intriguing feature of GIS was the tremendous analytical possibilities the systems can offer. For information managers in public and private sectors, on the contrary, the power of the technology mainly resided in its data management capability. It has been recognized since the mid-1980s that despite the tremendous growth of GIS's data management capability, the analytical dimension of the technology was largely ignored or under-represented in mainstream systems, at least by researchers' standards (Goodchild 1992; Batty and Xie 1994). Consequently, as geographical analysis became more advanced and specialized, more and more analysts failed to find the analytical power they wanted from GIS. For example, the main fruits of geographical analysis, especially those advocated by the quantitative side of the geographical discipline, were, by and large, not incorporated well into GIS. When people talked about "analysis" in GIS, they were either referring to basic GIS operations such as those for distance calculation, map overlay, buffering, and spatial query, or implying

specialized functionalities provided by the mainstream systems to support immediate societal and governmental needs, e.g. those for remote sensing imagery and risk planning (Eastman 1987). Although the "classical" geographical analytical methods, e.g. those for spatial autocorrelation estimation, location allocation, and spatial predication, could be added to some systems as software extensions (e.g. Arc/Info spatial analysts), they had gradually been excluded from GIS's technological core. Furthermore, since most GIS packages were developed around formalized but proprietary data structures and data models, adding an entirely new spatial data model, such as one based around Voronoi diagrams, was simply impossible. It was not accidental that some of the most important progress in geographical analysis during this time, for instance, the geographical analysis machine (Openshaw 1987), the local statistics of spatial association (Getis and Ord 1992), and complex models of geographical processes (Batty and Longley 1986; Goodchild and Mark 1987), to list only a few, were in fact developed in parallel to the main body of the GIS technology. As the original designs of mainstream systems had not foreseen those techniques, incorporating them was (and is still) a non-trivial and oftentimes difficult task, usually requiring intensive new development and expensive re-engineering.

To be fair, the emphasis of GIS on data management is not really a fault so much as a limitation, and one can argue that there is in fact no need for GIS to become a sophisticated analysis environment, since the major motivations behind most mainstream systems are to manage large amounts of geospatial data. The formalization of GIS is essentially a market-oriented selection and generalization process that evaluates many competing GI technologies according to the dominant market forces, and assimilates the most demanded ones into GIS's technological core. Nevertheless, the problem is, as GIS becomes a key technology in government affairs and industrial production, there is a reasonable risk that the prevalence of GIS could "force" academic researchers to adapt to the technology despite the nature of their studies, which will eventually lead to a harmful technology-driven trend in geography (Taylor and Overton 1991). One can easily

question why a GIS-based study is or should be "GIS-based": is it because the GIS technology truly provides new perspectives or novel methods that cast light on previously unsolvable problems? Or, in the worst sense, is it just due to the fact that our datasets are available on GIS platforms? If the answer is the latter, it means that all the subsequent analyses from the study are potentially subject to the limitations of the given GIS platform. Since GIS do not incorporate many useful methods for geographical analysis, it is not surprising that GIS do not always match the specific requirements of different problem solving scenarios. Such concerns were one of the reasons behind the heated debates in 1990s regarding the nature and suitability of GIS (Pickles 1993; Sheppard 1993; Pickles 1997), the appropriate use of the technology (Smith 1992; Curry 1995; Curry 1995), and the alleged epistemological naivety or poverty of GIS-based studies (Lake 1993; Sui 1994). A thorough and useful review of the critiques of GIS, especially those launched by the social-critical side of geography, can be found in Shuurman (2000).

2.2.3 Geographical Information Science

Following the formalization of the GIS technology and the initial observations of its deficiencies, the early 1990s saw a scientific turn in the geospatial information community, which aimed to emphasize and establish the "scientific aspects" of GIS and other geospatial data handling technologies (Goodchild 1992). The term "Geographical Information Science", or GISci in abbreviation, was coined at this time and has been widely used since then. The main arguments of GISci are two fold.

First, the concept of GISci provides a timely clarification and clear definition for the research endeavors conducted by the geospatial information community. It clearly states that GISci is not a narrowly focused technical area that only studies the GIS technology itself, but should be a diverse research field that examines the theories, development, and applications of geospatial information handling, including GIS and various other computational methods not supported by GIS (Goodchild 1992). GISci

defines a general computational approach, with or without GIS, to geospatially related problems, which focuses on 1) the cognitive, mathematical, and computational studies that make digitized geospatial information processing possible, and 2) new ways of "doing science" entailed by the applications of different GI technologies (Goodchild, Sheppard et al. 1999).

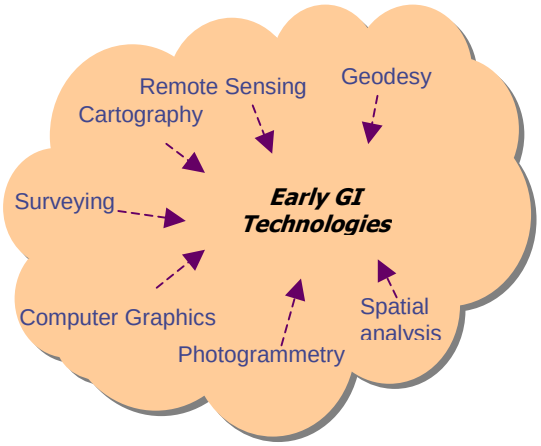
Second, GISci offers a possible reconciliation between GIS and its dissenters by pointing the latter to the wider vision of GI technologies, where one can find whatever functionality is absent from GIS in other (potential) computational methods. It is important to note that appropriate applications of GI technologies should always be problem-centered rather than technology-driven. Instead of searching for applications to a pre-assumed technology such as GIS, human analysts must have the obligation and freedom to select the most suitable technologies for their specific studies. Since GI technologies have been clearly treated as tools, the epistemological "crisis" of GIS raised earlier by human geographers (Smith 1992) becomes less a problem (at least from a GI scientist's perspective), because the epistemological stance of an application is ultimately decided by the nature of the study rather than the tools it uses.

To summarize, the concept of GISci has cleared the confusion about the relations between GIS and other GI technologies, defined the roles that GI technologies should play in geographical research, and shifted the main research focus of the geospatial information community from the tools themselves to more fundamental scientific issues that underpin their development and applications.

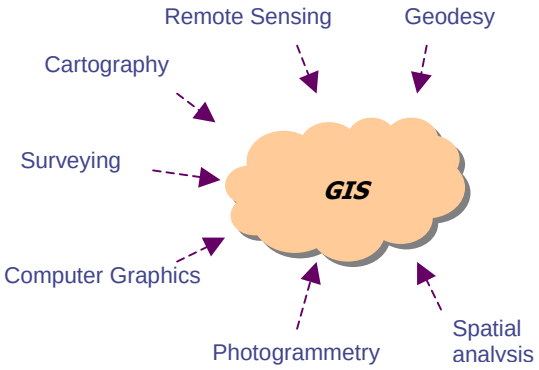
2.2.4 Geographical Problem Solving: The Challenges

Figure 2.1 shows the evolution of GI technologies from their early years to today's information era. As depicted in the picture, the formalization of GIS has generalized and advanced the early GI technologies into mature systems. However, new technical boundaries have also been drawn (by defining what are parts of GIS and what

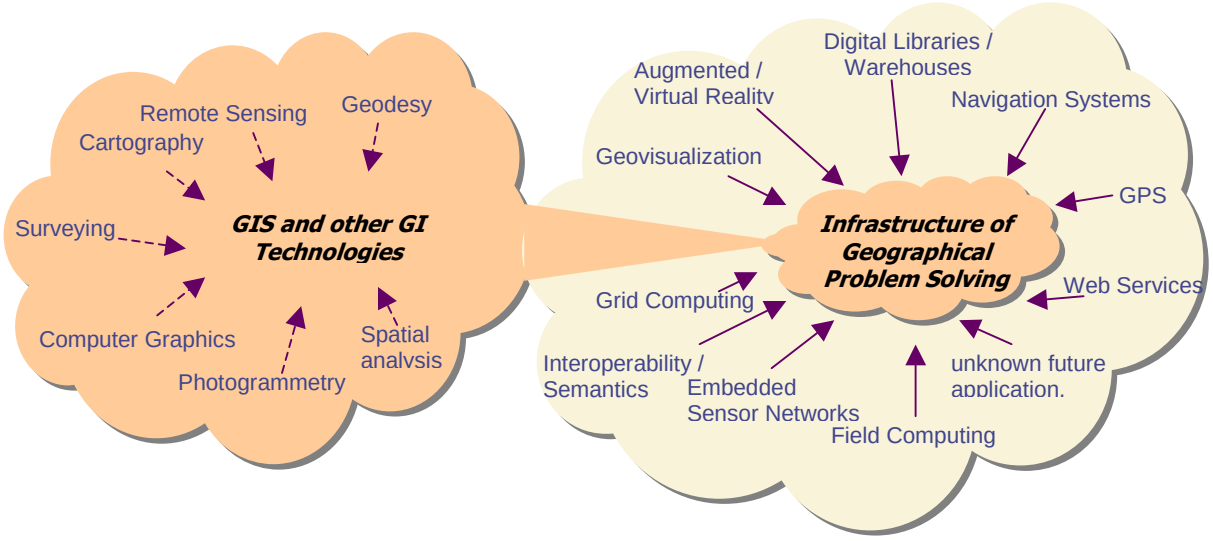
are not), and many domain-specific details are abstracted away. The problem-centered application paradigm defined by GISci requires an infrastructure of geographical problem solving that breaks the boundaries of GIS to support the flexible selection and seamless integration of different GI technologies, while providing on-demand access to other relevant data sources, computing services, and analytical methods produced by the latest scientific and technological developments.



Inception of GIS: The inputs from different disciplines help the incubation of early GI technologies



Formalization of GIS: Generalization results in mature GIS systems, but also sacrifices many domain-specific details and creates new boundaries for the technology.



Geographical Problem Solving: The integrated infrastructure of geographical problem solving should break technical boundaries between GIS and other GI technologies. In addition, new technologies should be easily added to geographers' toolbox.

Figure 2.1 Technological transitions for GISci and geographical problem solving. The diagram is based on NRC (2006)

Unfortunately, none of existing software systems, including GIS and other software packages used by geographers, meet such requirements of geographical problem solving. Currently the development of a problem solving infrastructure as described above presents the following challenges:

1. The challenge of open software integration. While software systems such as GIS are built by implementing a set of pre-defined geospatial data handling tasks, the main purpose of a geographical problem solving system is to provide users with integrated access to functionalities either provided by the system itself or developed by other sources. Furthermore, as scientific discovery is usually achieved by trying alternative approaches, using innovative methods, and conducting novel experiments, the system should be able to incorporate new functionalities and support new ways of thinking in a seamless and timely manner. This means a geographical problem solving system must be open and highly extensible. How to build and maintain such an open system with integrated support of external functionalities is a new challenge for software engineering.

2. The challenge of flexibility versus ease of use. To enable users, especially non-experts, to fully utilize the power of digital computing, a geographical problem solving system must provide an intuitive yet powerful front-end interface, through which users can easily create customized computational tasks to control highly subtle behaviors of the system. From the perspective of user interface design, neither intuitiveness nor controllability is a big issue when considered separately; however, they become a difficult goal when combined. For example, while Graphical User Interfaces (GUIs) have proved to be an effective way to improve the intuitiveness and usability of software systems, they limit the flexibility of task control by only allowing users to access the functionalities "exposed" by the given GUI widgets. On the other hand, although batch-processing command shells (e.g. the Unix command-line console) are extremely flexible in controlling system operations, such flexibility is usually only available to more

technically salient users who understand the command system well and know how to write a batch-processing script. In summary, how to achieve both intuitiveness and flexibility at the same time is another main challenge for the design and development of geographical problem solving systems.

3. The challenge of knowledge production and understanding. One of the ultimate goals of GISci is to facilitate meaningful geographical analysis, where GI technologies are used "in conjunction with knowledge rather that as a substitute for it" (Goodchild 1991). While geographical knowledge construction is an important research topic in its own right, from the perspective of problem solving, it leads to a critical issue regarding how to better connect tools with high-level research questions originating from application domains. For example, what tools can be used to support data mining analyses performed on the epidemiological cancer dataset of the Appalachian region collected at the county level? Which visualization tools can best show the hidden spatial-temporal patterns in the information of travelers to Washington D.C. during the past several years? Those kinds of problems raise a new challenge of knowledge-based geographical analysis support, with which the deployment of GI technologies can be guided and / or automated by stored high-level knowledge. Although it has been long recognized that methods from fields such as knowledge engineering and AI can be adopted to achieve those goals (Macmillan 1997), the design and implementation of knowledge-based geographical analysis support have never been fully addressed by current GI technologies.

2.3 Scientific Problem Solving Environments

In scientific computing, one technology that answers the above challenges is known as scientific problem solving environments (PSEs), which are described by Gallopoulos and his colleagues as computational systems that "integrate powerful functionalities necessary to accomplish high-level tasks" and interact with users on

"domain terms" (Gallopoulos, Houstis *et al.* 1994). According to Houtis and Rice (2002), as specialization becomes a matter of fact in today's scientific research and education, it is unrealistic to expect most scientists to possess the technical expertise to integrate and exploit the latest technologies by direct programming or complex software operations. Consequently, a scientific PSE must provide a software environment that integrates different technologies for the users and guides or enforces them towards best practices (Houstis and Rice 2002).

The goals of scientific PSEs well match the requirements of geographical problem solving. A geographical problem solving system, therefore, can be designed as a scientific PSE tailored for geospatial information and geographical applications, or more simply stated, as a ***Geospatial PSE***. This section reviews the historical background of scientific PSEs, examines the main elements of a PSE system, and discusses useful experience from the scientific computing community that may help the design and development of Geospatial PSEs.

2.3.1 The Historical Background of Scientific PSEs

The development history of scientific PSEs began in the early 1960s, shortly after the emergence of high-level programming languages such as ALGOL and FORTRAN. The picture of PSEs was painted in Culler and Fried's work towards an "online computing center for scientific problems" (Culler and Fried 1963), which is regarded by many as the earliest work on scientific PSEs. At a time when users needed to write numerous lines of arcane code to perform digital computing, scientific PSEs was an inspiring and ambitious idea: since advanced programming languages could liberate programmers from wrestling with the unintuitive machine instructions, why not build high-level software systems for scientists that require no user programming at all? Such systems should integrate all the necessary functionalities implemented by specialized technicians, and provide "natural" and intuitive user interfaces that employ "scientific

languages" rather than conventional programs to control the operations of the systems. Thus, PSEs enable researchers to concentrate on their specific problems rather than struggling with the machine.

However, in the 1960s such an idea appeared to be far ahead of its time. For example, what constitutes "human terms" or "scientific languages"? Some researchers suggested that natural languages plus scientific terminologies might be the solution (Houstis and Rice 2000). But in an era when the theory and practice of natural language processing were still incubating, it was impossible to build usable systems that could truly understand natural languages. Performance was another major handicap, as the comprehensiveness and complexity of a PSE were beyond the hardware and software capacities at the time. In the late 1960s, it was realized by most scientists that PSE was a vision that could hardly make any realistic impact given the theoretical and technical contexts at the time.

Since the mid 1980s, the once deceased idea of PSE has been rejuvenated thanks to the advances of technologies. With the dramatically improved hardware and software capabilities, the impossibilities encountered in the 1960s suddenly became something worthy again of realistic consideration. As the pace of technology advance keeps accelerating and the demands for computer power in the hard and soft sciences continue to grow, researchers have begun to re-embrace the concept of PSE. As a result, software systems that are explicitly designed as scientific PSEs have emerged and been applied in different domains. For example, Parallel ELLPACK is an PSE for solving partial differential equations (PDEs), which provides users with semi-automated, "intelligent" access to high-performance PDE solvers (Rice and Boisvert 1985; Houstis, Rice et al. 1989). MATLAB (Matrix Laboratory) is a popular PSE for mathematical problem solving, which is characterized by its powerful command shell that "expresses problems and solutions as mathematical notations" (Mathworks 2003). Cantata is a PSE for digital image processing that uses a visual language to help users construct complex image

processing tasks (Young, Argiro et al. 1995). One can visit the website at http://www.cs.purdue.edu/research/cse/pses/research.html to see more PSEs that have been developed and applied in different scientific fields. Today, scientific PSEs have rapidly grown into a new, active, and promising field that crosses computer science and different scientific domains, with numerous successful software products built under its banner.

2.3.2 The Core Elements of Scientific PSEs

In order to construct robust scientific PSEs under a realistic development budget, researchers have summarized a set of software design guidelines for PSE development. Generally speaking, the core elements of a scientific PSE can be described using the following formulae (Grzechynka and Harezlak 2000; Houstis and Rice 2000; Houstis and Rice 2002):

PSE = User Interface + Libraries + KnowledgeBases + Integration

In other words, a scientific PSE needs to adopt appropriate software integration strategies to combine reusable functionalities implemented by shared software libraries, and present them to scientists through an appropriate user interface, where the software usage can be guided and automated based on computationally represented knowledge. Such design guidelines provide feasible answers to the challenges raised by geographical problem solving.

1. Software Reuse and Loose-coupling Integration. Most scientific PSEs employ software reuse strategies and adopt loose-coupling software integration technologies to address the massive software integration demands imposed by scientific computing.

Software reuse is motivated by the fact that different software systems usually have similar, potentially shareable features and functions. Since software development is an extremely costly practice, reusing those commonalities rather "re-inventing the wheel"

can significantly reduce the cost of software development (Krueger 1992). The idea of software reuse is particularly important to the development of scientific PSEs, because the cost of building analytical functionalities for a PSE-like software system is extremely high, and reusing existing implementations is an important way to keep the development cost under control. Historically, numerical libraries such as LAPACK (Linear Algebra PACKage; http://www.netlib.org/lapack/) and NAG (Numerical Algorithm Group; http://www.nag.co.uk/) have been indispensable parts of scientific computing in general and PSE development in particular. Although these libraries are not front-end systems that can be directly used by scientists, they usually constitute the foundation of most PSEs' underlying implementations. For instance, the system Parallel ELLPACK is based on the numerical library ELLPACK (Houstis, Rice et al. 1989), and MATLAB's implementation has heavily drawn upon numerical libraries such as LAPACK. One can visit http://www.netlib.org to explore the long list of shared scientific computing libraries in the public domain and their usage in PSE development.

Software functionalities implemented in shared libraries are usually integrated by scientific PSEs through a loose-coupling software integration infrastructure. The relation between reusable libraries and integration infrastructure is like stamps and glue, where pieces of functionalities from reusable modules communicate with each other using the runtime services and protocols provided by the integration platform (Keye 2003).

Loose-coupling integration contrasts with tight-coupling integration, which refers to any software coupling technique that combines software modules together by creating dependencies among them. Such dependencies could be shared data structures, common system models, or direct references to each other in the implementing code. On the contrary, the loose-coupling approach integrates relatively independent software units through a commonly agreed communication mechanism. The only things shared by the integrated parts are the protocols of data exchanging and conventions of software inter-operation. Individual software units can have their own data structures and system

models, and direct referencing in the implementing programs is deliberately avoided. Generally speaking, a tight-coupling integration can achieve better performance and reduce the development cost in the short term, since all elements of the system can be directly controlled and centrally managed by a dedicated group of developers. However, in the long run it will cause severe problems of software evolution, because changing any single module may require subsequent modifications of other system components connected with it. On the other hand, loose-coupling integration normally requires more development effort to make software modules independent of each other, and it might result in some immediate performance drop due to the overhead of data communications. But it is rewarding in the long run due to its convenience in replacing, modifying, or upgrading individual modules. A scientific PSE, therefore, should be developed based on a loose-coupling software integration strategy, which flexibly integrates functionalities from different reusable libraries.

Different loose-coupling techniques have been successfully utilized in scientific PSE development. For example, a component-based scientific PSE is illustrated by PSEware, a scientific PSE for symbolic mathematics (Lakshman, Char et al. 1998). A PSE platform based on a software bus is developed by Weerawarana and his colleagues to solve partial differential equations (Weerawarana 1994; Weerawarana, Houstis et al. 2000). In conclusion, the software reuse model and loose-coupling integration strategies illustrated by scientific PSEs provide a feasible solution to the challenge of software integration faced by today's geographical problem solving.

2. Constructive User Interface. The development of scientific PSEs provides valuable experience and useful examples in designing user-friendly and highly flexible user interfaces. Since the inception of PSEs, scientists have deliberately distanced a PSE system from software tools that only allow users to access fixed set of functionalities. In the early years of PSE, the user interface of a PSE system was described as an "intelligent" assistant to scientists that "talks" with humans and automatically writes

programs for them (Gallopoulos, Houstis et al. 1994). Although this ideal vision turned out to be unrealistic, an intuitive user interface that help scientists control detailed computational processes, create novel tasks, or even generate new programs, is still regarded as one of the defining characteristics of scientific PSEs (Houstis and Rice 2000; Houstis and Rice 2002). Generally speaking, two less ambitious (compared with natural language processing) but more practical approaches have been successfully used in PSE's user interface design.

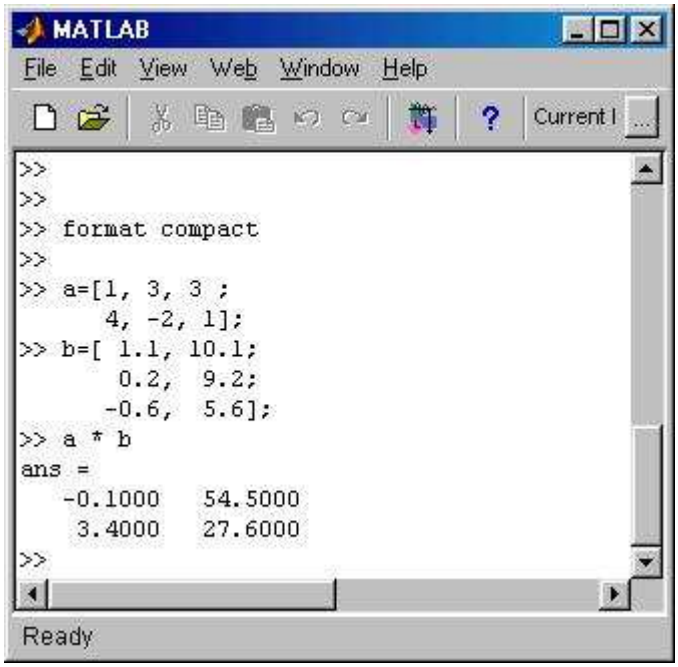


Figure 2.2 Matrix multiplication in MATLAB.

The first approach is to develop a domain-centered command shell, which has been widely used in mathematical PSEs such as MATLAB and MATHEMATICA. The command shells of those systems are equipped with commands constructed from commonly recognized mathematical notations. For example, as depicted in figure 2.2, in the MATLAB command shell, the multiplication of matrices **a** and **b** is expressed as **a*b,** a straightforward format to scientists with adequate linear algebra background. With the batch-processing script interpreter provided by the command shell, users can write script programs mostly comprised of mathematic notations to perform sophisticated analyses.

Moderate programming skills are still required, but the syntax of script programming is normally much simpler than conventional programming.

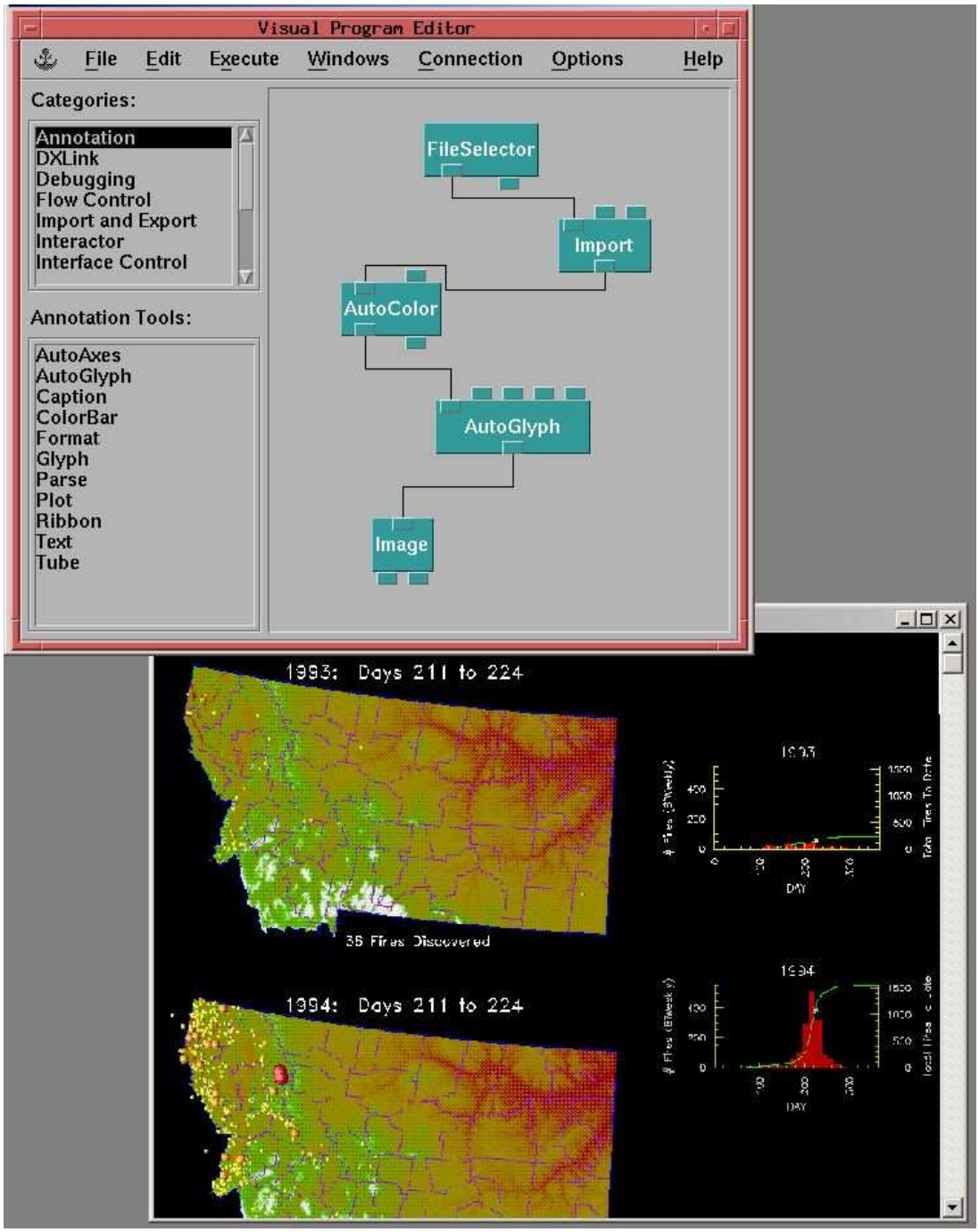


Figure 2.3 An OpenDX visual program that displays color maps. The example visualizes a dataset about the historical wide fire in the western U.S. (Duce, Voth et al. 1996).

The second approach is demonstrated by those PSEs that incorporate Visual Programming (VP) into their front-end interface designs. VP is a computer programming paradigm that aims to construct programs visually rather than textually (Golin 1992). There are a variety of VP approaches, including those based on iconic visual languages (Chang 1986), the form-based tools that generate programs according to user inputs (Wilde and Lewis 1990), and the dataflow-based visual programming paradigm used in GeoVISTA *Studio* (Takatsuka and Gahegan 2002). But the central idea of VP remains the same, which is to exploit the advantages of graphic-based human computer interaction to improve or even replace the text-based computer programming (Chorafas 1996). Figure 2.3 depicts a visual program constructed in OpenDX, an information visualization PSE based on the open source tool DataExplorer (OpenDX.org 2000), which in this example visualizes the historical wildfire records for the west coast of the United States. As shown in the visual program editor (the top window), the block diagram describes the dataflow structure of the visual program, where rectangular boxes are processing modules and black lines denote dataflow channels among them.

Both domain-centered command shells and VP have their pros and cons. A command shell usually enjoys the compactness and expressiveness offered by textual programming, and advanced users can write large, complex batch-processing scripts to create new functionalities. For example, a significant number of MATLAB's modules and extensions are directly written in its own shell scripting language. However, the intuitiveness of a command system is highly dependent on the symbolic nature of the targeted domain. For mathematically related fields where symbolic notations are widely used and commonly understood, it is relatively easy to develop an intuitive command system such as the one used in MATLAB. For less symbolic fields such as geography, on the contrary, one can barely find such a command system because the commonly agreed symbolic notations simply do not exist in those fields. Consequently, software designers have to invent symbolic terms and notations themselves, which often results in

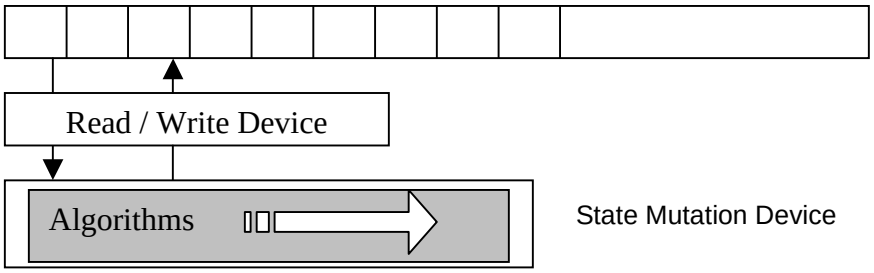
less intuitive or even arcane commands (A good example is the 3000+ commands provided by Arc/Info Workstation). On the other hand, the graphic and (sometimes) iconic representations used by VP constitute a more intuitive interface for a broader audience, because the nature of the targeted domain usually has little impact on the understanding of visual metaphors. The main disadvantage of VP is that it does not have the compactness offered by textual programming, and a visual program of modest size may result in huge, complex, and unreadable visual structures. This makes VP unsuitable for large-scale software construction. However, for front-end user interfacing, where task structures are generally much simpler than programs generated during software implementation, VP provides a neat combination of friendly GUI and powerful operational control.

To summarize, while neither command shell nor VP is the invention of scientific PSEs, their applications in PSE development provide useful experience for interface designers of geographical problem solving systems. Since geography is not a symbolic field such as mathematics, it is reasonable to regard VP as the more feasible design choice than command shell.

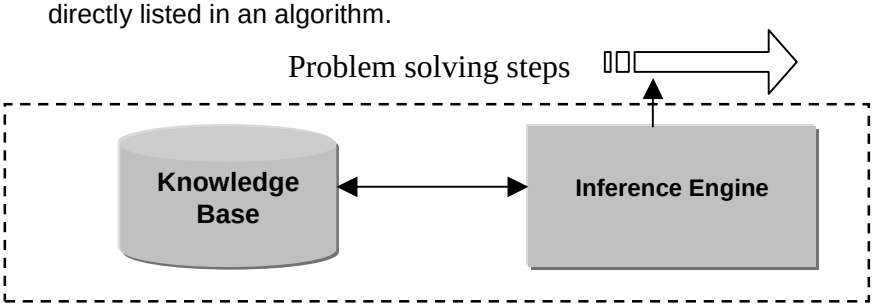
3. Knowledge-based Support. Scientific PSEs recognize the importance of computationally represented knowledge in scientific problem solving, and have explored the possibility of building intelligent and automated problem solving tools using knowledge-based systems. Early effort on intelligent problem solving tools can be traced back to the idea of "polyalgorithms", which described automated algorithm selectors that could find the most suitable algorithms according to a set of criteria extracted from the problem settings (Rice 1968). Since then, knowledge-based support has been regarded as a key element of a scientific PSE, which plays an central organizational role in the system by transparently connecting different pieces of software with high level understandings (Houstis and Rice 2002).

The key technology in knowledge-based support is knowledge-based systems. Conceptually, the basic idea of knowledge-based systems is to simulate the human decision-making processes by encoding *expertise* (the knowledge possessed by experts), and use such information to automate decision making (Schreiber, Akkermans et al. 1999). For example, when a spatial analyst performs cluster analysis over a given epidemiological dataset, (s)he normally relies on personal understandings about the same kind of spatial analyses and past experience with similar datasets to select and configure the clustering method to use. If such expertise can be captured and computationally represented in a knowledgebase, we can develop computer tools to select and configure spatial clustering methods automatically, which saves considerable time and effort for users, especially non-experts.

Algorithmically engineered software tools and knowledge-based systems represent two contrasting models of digital computing: the procedural and declarative computing models. As described by the top picture in figure 2.4, algorithms are essentially procedural Turing Machines (Davis 2000), which can be intuitively described as a *pre-defined sequence* of instructions that can be performed by a state mutation device (e.g. CPU) to modify linearly organized data storage units (e.g. memory). On the contrary, a knowledge-based system declares relevant information about digital computing in a knowledgebase, and uses a generic program known as the *inference engine* to derive the problem solving procedures. Generally speaking, an algorithmically engineered tool is more efficient than a knowledge-based system, since fundamentally all modern digital computers are procedural Turing Machines. However, since users do not need to write complicated procedures themselves, knowledge-based systems are much more powerful in terms of automation, especially when algorithmic solutions are too difficult for human programmers to build.



A. Procedural computing model, where problem solving steps are directly listed in an algorithm.



B. Declarative computing model, where problem solving steps are derived by an inference engine.

Figure 2.4 The procedural and declarative digital computing models.

While the early development of knowledge-based systems was mainly under the banner of expert systems, contemporary studies on this subject tend to general, where the knowledge encoded by a system is no longer restricted to expertise, but can be any kinds of qualitative concepts, relations, and rules (Schreiber, Akkermans et al. 1999). Scientific PSEs exploit the automation power of knowledge-based systems by combining algorithmic solutions with carefully designed knowledge models, which capture the relationships between the algorithm space and the problem space (Rice 1976). The algorithm space is a model that describes all algorithmic solutions, while the problem space is an abstract knowledge structure related to the problem settings. A knowledge model for scientific problem solving, therefore, can be constructed by defining its algorithmic space, problem space, and the relationships between them. Based on the knowledge model, the inference engine can search the algorithmic space to find the most suitable algorithms for given problem settings. A successful example can be seen in PYTHIA, a knowledge-based PSE that automatically selects scientific algorithms for

particle physics (Weerawarana, Houstis et al. 1996). In future PSE development, the role of knowledge-based systems can be extended to automate the full span of scientific problem solving, with which users only need to sketch the conceptual structures of high-level tasks, and the system will solve the problems automatically (Houstis and Rice 2000).

2.4 The Geospatial PSE

Based on the discussions above, the enabling technology for geography and GISci can be designed as a Geospatial PSE, which should be developed by:

1. Designing an appropriate open software architecture based on shared GI technologies and a loose-coupling software integration infrastructure;
2. Providing a visual programming interface for users to create, control, and monitor computational tasks;
3. Creating knowledge-based support for users to select methods, configure tools, and generate tasks.

The rest of this section describes these three software development goals, and presents the basic blueprint of a Geospatial PSE.

2.4.1 Open Software Architecture

Software sharing and reusing are not new concepts in the geospatial information community. Although the proprietary nature of the mainstream GIS has created barriers for users to exchange data among different systems, this problem has been well addressed thanks to the latest progress in interoperable GIS (Bishr 1997). Furthermore, the recent adoptions of software inter-operation technologies such as software components and Web-based computing have largely removed the syntactical handicap to sharing and reusing software functionalities, even on proprietary systems (for example, recent versions of Arc/Info have included a set of software components under the name of ArcObjects that encapsulate most of its core functionalities). In addition, since the 1980s,

large-scale open source development efforts such as the GRASS (Geographical Resources Analysis Support System) project have resulted in a number of shared geospatial data formats and handling tools in the public domain (Reiter 2002). The current initiatives proposed by OGC (Open Geospatial Consortium), a non-commercial organization devoted to creating open standards for GIS, have published various specifications for sharing and reusing different aspects of GI technologies, e.g. The Simple Feature Model for geographical data modeling (OGC 1999), GML (Geography Markup Language) for sharing geospatial data on the Web (OGC 2004), the WMS (Web Map Service) for online mapping (OGC 2004), to list just a few . Supporting tools for these standards have been developed by major GIS vendors to facilitate system integration, and free solutions have been built and distributed as open source libraries such as GeoTools (http://docs.codehaus.org/display/GEOTOOLS/Home), JTS (Java Topology Suite; http://www.vividsolutions.com/jts/jtshome.htm), and the GeoServer (http://docs.codehaus.org/display/GEOS/Home).

However, currently none of existing software systems have provided a software architecture that fully exploits the power of sharing and reusing. Figure 2.5 depicts the evolution of software architectures from the pre-Internet era to today's networked, distributed environment. At the time when computer networks were expensive and distributed computing was rare, the stand-alone system architecture was the primary choice of software design, which implements all functionalities, including input / output operations, data processing, and user interfaces, as a monolithic application. A good example is the early versions of Arc/Info Workstation, which provided its own implementations for everything ranging from data management to digital mapping. With the advance of the networking infrastructure and the consequent demands for distributed systems, the client-server architecture became a popular model of distributed computing. It divides the stand-alone application into a dyadic structure, where clients pass the high-end requests from user applications to servers, and servers retrieve information and send

the results back to clients. Since the 1990s, many GIS vendors have adopted the client-server architecture, which delegates information management to back-end data servers driven by more powerful and more specialized DBMS and thus allows the client applications to focus on front-end user interactions

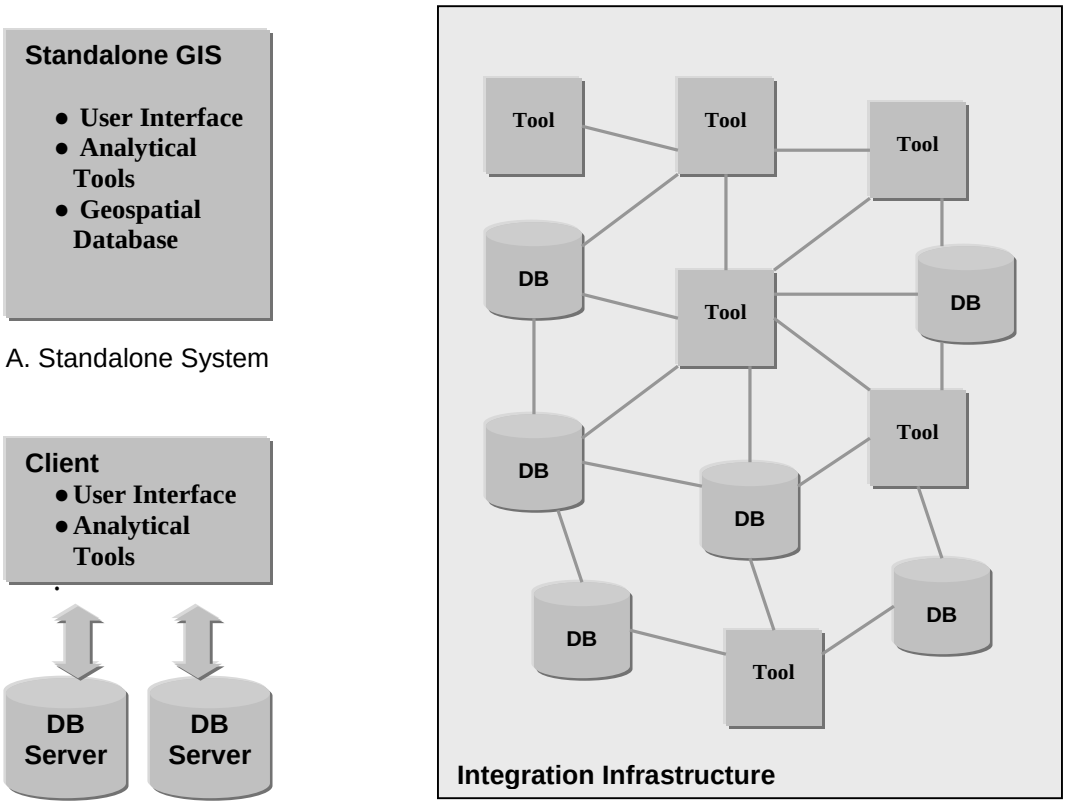


Figure 2.5 The evolution of software system structure.

In more recent years, as the scale and complexity of distributed systems grow, the classical roles of client and server have blurred and the dependencies between information entities have become more complicated than the simple request-and-respond relation. For example, it is commonplace for a module in a large distributed application to be both a client and a server to other modules. The dissolving relations between clients and servers are eventually transforming the structures of software systems into a more decentralized grid (or network) structure, where clients and servers become peers.

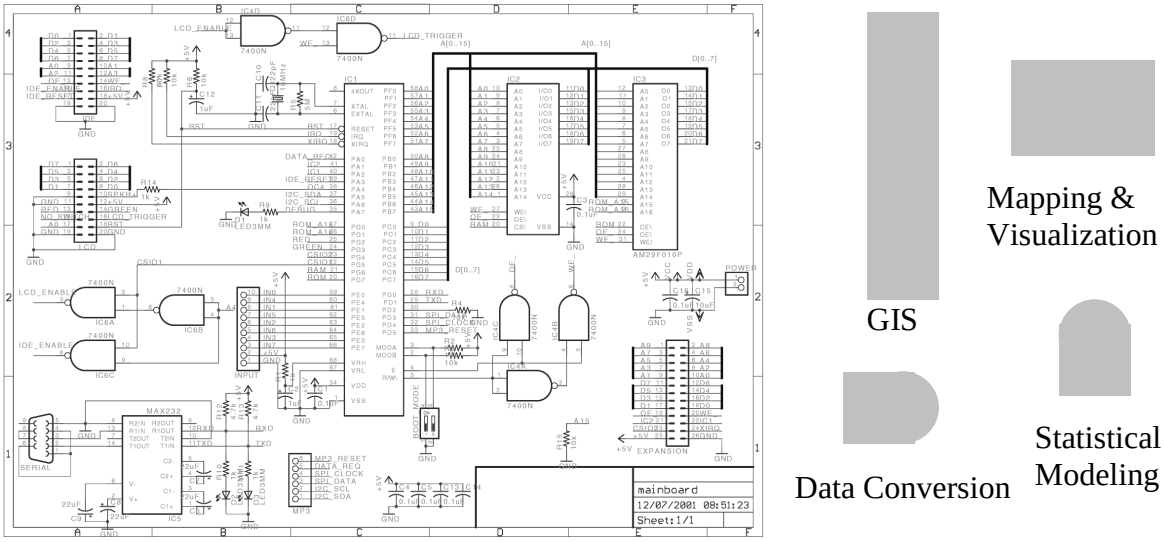
Each node in the grid is a self-contained unit of information processing, and the construction and evolution of the system are extremely flexible and intrinsically open, as new nodes can be added to the system without interfering with existing ones.

From a system architectural point of view, a grid-based structure is most suitable for the software reuse and integration goals required by geographical problem solving. However, since historically most GI technologies have been implemented as stand-alone systems, in order to gain continued benefits from legacy software frameworks, most software vendors tend to treat open software technologies only as "hot-fixes" to newly emerged problems of system inter-operation, rather than embracing them at the architectural level to re-design their standalone products. While it is not unusual for a GIS package to provide additional tools to access open data sources or utilize software extensions from third parties, it is rare for a system to design its core functionalities completely around software reuse and integration. As result, most systems are still standalone in nature.

The first step of this study, therefore, is to adopt an open software architecture for Geospatial PSEs, the structure of which is grid-based in nature. More specifically, the framework proposed in this study elaborates the component-based approach presented in GeoVISTA *Studio*, which has been developed and applied as a Java-based problem solving workbench that allows users to construct highly customized applications from reusable software components (Takatsuka and Gahegan 2002). GeoVISTA *Studio* relies on the JavaBeans component standard, a loose-coupling software integration technology from Sun Microsystems (Sun Microsystems 1998). The software architecture of GeoVISTA *Studio* can be divided into two parts:

- A main software framework that implements the basic software integration infrastructure according to the data communication and method invocation standards defined by the JavaBeans specification;

- An open collection of software components conforming to the JavaBeans component specification, which are also known as “Java beans”.

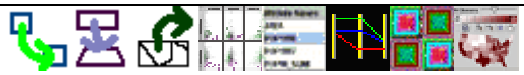
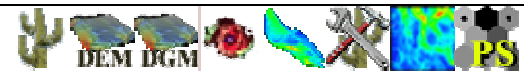
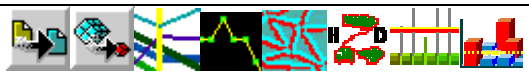


A. The Main Software Framework B. Software Components

Figure 2.6 The component-based software architecture. The software system is analogous to integrated circuit design, where the software framework is the “main board” and software components are circuit units plugged on it.

One can refer to Szyperski’s work for a detailed discussion of component technologies, including JavaBeans (Szyperski 1998). As shown in figure 2.6, the relation between the main framework and software components is similar to that of a main board and individual circuit units in integrated circuit design, where components that implement different GI technologies can be plugged into the main framework to form a new application. Table 2.1 provides a list of selected Java beans currently incorporated in GeoVISTA *Studio’s* tool repository, which cover various aspects of geographical problem solving, ranging from basic data accessing to complex analysis and visualization. Moreover, the repository is open in nature, as new Java beans can be added and immediately integrated without any additional development work.

Table 2.1 Components in GeoVISTA *Studio* categorized according to their domains of origin. The icons listed in the right column are pictorial depictions of JavaBeans components. For a complete list of components and the detailed information about which components those icons stand for, one can visit the web site at http://www.geovistastudio.psu.edu. Moreover, the website http://java.sun.com and the book by Eckstein, Loy, and their colleagues (2002) provides a thorough reference to Java Swing components in the table.

Categories	Domains of Origin	Sample JavaBeans Components
Swing	Basic GUI components from the javax.swing package.	
GeoViz	General components for geographical visualization.	
Java3D	3D visualization components based on Java3D technology.	
SOM	Pattern clustering and classification tools based on self-organization map.	
SpatialDM	Specialized data analysis and information visualization tools tailored for spatial data mining.	
GeoTools	Geospatial data handling tools based on OpenGIS standards and the GeoTools API.	

2.4.2 Visual Programming Interface based on Open Dataflow

Visual programming and related approaches have also been applied in geospatial communities for a long time. Early attempts include the interface to Map II developed by Kirby and Pazner (1990), and the flowcharts gateway to Arc/Info provided by Lanter and Essinger (1991). The latest version of Arc/GIS also provides a built-in visual interface for users to edit analytical workflows (ESRI 2004). Egenhoffer and Bruns (1995) describe an approach called visual map algebra, which provides a visual interface for users to directly manipulate map overlay (Figure 2.7). A general visual programming shell to GIS is described by Albrecht (1996), which is based on a task-oriented generalization and categorization of existing GIS operations. With a set of "universal

GIS operations", Albrecht (1997) proposes the idea of "Virtual GIS" (VGIS), which is a visual programming environment that uses iconic depictions of platform-independent universal GIS operations to visually construct analytical processes. VGIS itself is only a front-end shell that does not implement any functionalities, but it can be mapped into concrete procedures on back-end GIS platforms such Arc/Info (Figure 2.8).

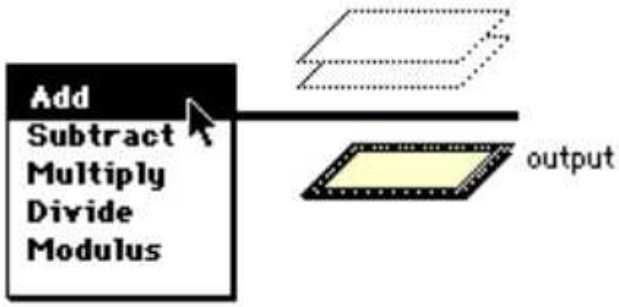


Figure 2.7 Visual map algebra proposed by Egenhofer, with which users can directly manipulate map layers using a mouse (Egenhofer and Bruns 1995).

The problem of these approaches lies in the fact that all of them are built upon a specific abstraction of geospatial data handling. For example, the Arc/Info gateway described by Lanter and Pazner and Arc/Info's own workflow editor are both designed to only work on Arc/Info. Visual map algebra is based on a layered view of geospatial data, which is not supported by a large number of tools, especially those built around object-oriented geospatial data models. While Albrecht's "Virtual GIS" is not dependent on any specific systems, it relies on its own abstraction of GIS operations, and whether such an abstraction is truly "universal" is questionable. As a result, it is impossible to use the interface to support an open geographical problem solving system that can potentially encounter arbitrary data and functionalities.

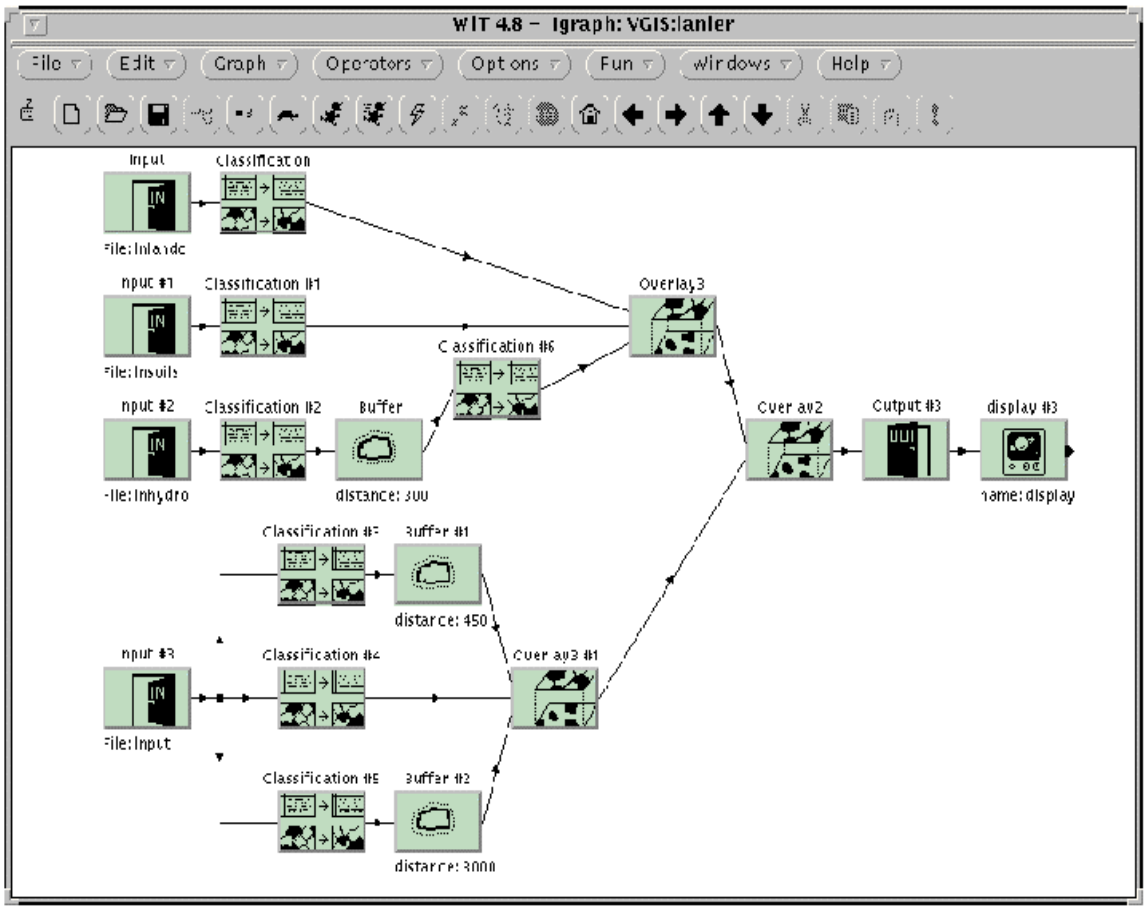


Figure 2.8 A screen shot of VGIS proposed by Albrecht (1997), in which different GIS operations are pipelined to perform a larger analytical task.

This study adopts an open, dataflow-based visual programming model, which is the approach currently employed by GeoVISTA *Studio*. More specifically, a dataflow network can be generally defined as a directed graph *(V, E)*, where *V* is a set of nodes representing units of digital computing, and *E* is a set of directed edges denoting data streams between the nodes. During runtime execution, each edge maintains a FIFO (First In and First Out) queue of data tokens. Each node gets a data token from the queue, executes a process, and puts the resulting data into its outgoing edges as new tokens. In addition, a node usually is able to perform multiple computational actions and uses a set of "firing-rules" to decide which actions should be performed under what conditions. These firing rules can be based on the global state of the dataflow network, the internal

state of the node, and the state of data streams that connect with the node (Lee and Parks 1995).

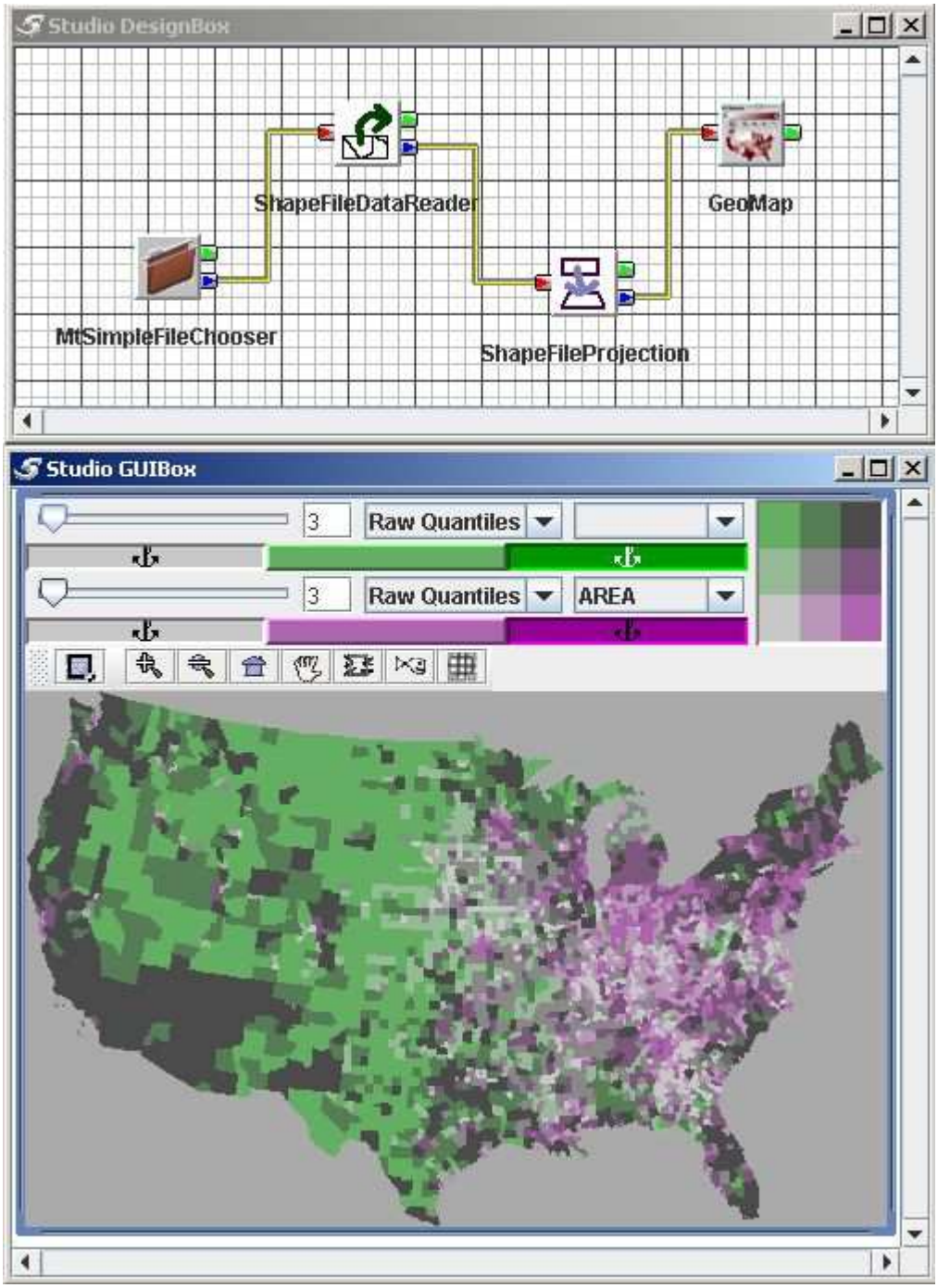


Figure 2.9 A simple visual program in GeoVISTA *Studio*, which shows a choropleth map.

Figure 2.9 shows a simple dataflow application constructed in GeoVISTA *Studio*, which displays two-dimensional maps for user-selected geospatial datasets. Four JavaBeans components are used in the dataflow, including the MtSimpleFileChooser component that allows users to select the data file, the ShapefileDataReader component that parses data from a ESRI shape file, the ShapefileProjection component that performs map projection on the parsed dataset, and the GeoMap component that symbolizes and displays the map on the screen. The pipeline-like edges represent the data flows among the components, which can be created and configured by the user. Compared with the visual programming interfaces provided in systems like VGIS, the open dataflow model used in GeoVISTA *Studio* deliberately avoids any pre-assumptions about the datasets and analytical functionalities used in visual programming. In other words, *any* datasets and analytical functionalities can be visually programmed as long as they are encapsulated in JavaBeans components. As a result, the dataflow model is intrinsically open and can be used to compose and control the behaviors of arbitrary components.

2.4.3 Knowledge-based Automation

How to capture the knowledge of geographical problem solving and use it to facilitate the application of GI technologies is a knowledge engineering problem that contains two inter-related aspects: conceptualization and representation.

1. The Issue of Conceptualization. Conceptualization, or conceptual knowledge modeling, is the process of identifying concepts, relationships, and conceptual structures within the targeted knowledge domain. Since geographical problem solving is essentially a geospatial data handling process, two kinds of conceptualizations are particularly important to a Geospatial PSE:

- Conceptualization of geospatial data, which aims to build high-level descriptions about geospatial datasets. A geospatial dataset is an information carrier that has been structured based on certain data models, and the data

models are essentially conceptualizations of geospatial space-time (Goodchild 1992). However, a conceptualization of geospatial data should not only include descriptions about data models, but also provide additional information about the purposes and uses of data, which help problem solvers apply the right datasets in the right way.

- Conceptualization of geospatial data processing, which should be focused on the procedures or "steps" of pursuing different geospatial analysis work. Notable work on this topic includes DiBiase's four-stage view on geo-visualization (DiBiase 1990), Monmonier's study on graphics scripts (Monmonier 1992), MacEachren's work on geographical knowledge discovery (MacEachren, Masters et al. 1999), and Gahegan and Bodaric's recent work on scientific discovery processes in geography (Gahegan and Brodaric 2002; Gahegan 2005)

2. The Issue of Representation. Compared with conceptualization's focus on conceptual modeling, representation is more concerned with *encoding*, or how to incarnate conceptual models into knowledge constructs that can be comprehended by humans *and* processed by computers. Representation produces the actual content (i.e. computable knowledge) stored in a knowledgebase and later used in automation. The primary knowledge representation techniques used in this study are computational ontologies. In philosophy, ontology is the study of existence. This term is borrowed by knowledge engineering to describe a knowledge representation approach that regards explicitness and portability as the top priority (Gruber 1993). While standards of geospatial data transfer provide the common formats for data sharing, ontologies can be used to define the conventions of geographical knowledge exchange. Furthermore, as structured knowledge processing is explicitly supported, ontologies can be easily imported or directly handled by knowledge-based systems. Recently ontologies have been widely applied in GISci to facilitate information and knowledge sharing in different

application contexts (Fonseca, Borges et al. 2000; Kuhn 2001; Pundt and Bishr 2002; Van de Vlag, Jeansoulin et al. 2005), and the framework of Ontology-Driven GIS (ODGIS) has been proposed to design shareable systems. Unlike the previous data-oriented GIS paradigm, an ODGIS is built on top of portable ontologies and thus shareable in nature, which significantly reduces the complexity and cost of geospatial information integration (Fonseca and Egenhofer 1999). As the portable carriers of conceptual models, ontologies constitute the foundation of today's Semantic Geospatial Web, the infrastructure that delivers geospatial knowledge and information across the Internet (Egenhofer 2002).

With these concerns in mind, the knowledge-based support of a Geospatial PSE must be based on a conceptual model that describes geographical problem solving in terms of geospatial data and analytical processes, and then creates an appropriate ontological framework to represent, store, and share such knowledge over the Semantic Geospatial Web.

2.4.4 The Overall Design of a Geospatial PSE

The overall design of a Geospatial PSE is depicted in Figure 2.10. At the most general level, the system contains three key elements:

- The software inter-operation platform defines the main framework of the open software architecture (or the "main board"). It provides the infrastructure to integrate different software components distributed from the Semantic Geospatial Web, including the components for geospatial data access, the components implementing analytical functionalities, and the "wrapper" components that allow the system to access incompatible legacy tools;
- The knowledgebase and inference engine constitute the built-in knowledge-based system provided by the PSE. The knowledgebase stores ontologies about geospatial data and geographical problem solving processes, while the

inference engine implements intelligent programs that provide different levels of automation by reasoning with the knowledge stored in the knowledgebase.

- The visual programming interface provides the front end for task creation and modification. With the open dataflow model, the visual programming interface uses shared software components rather than a pre-defined set of functionalities as the basic building block to compose user tasks.

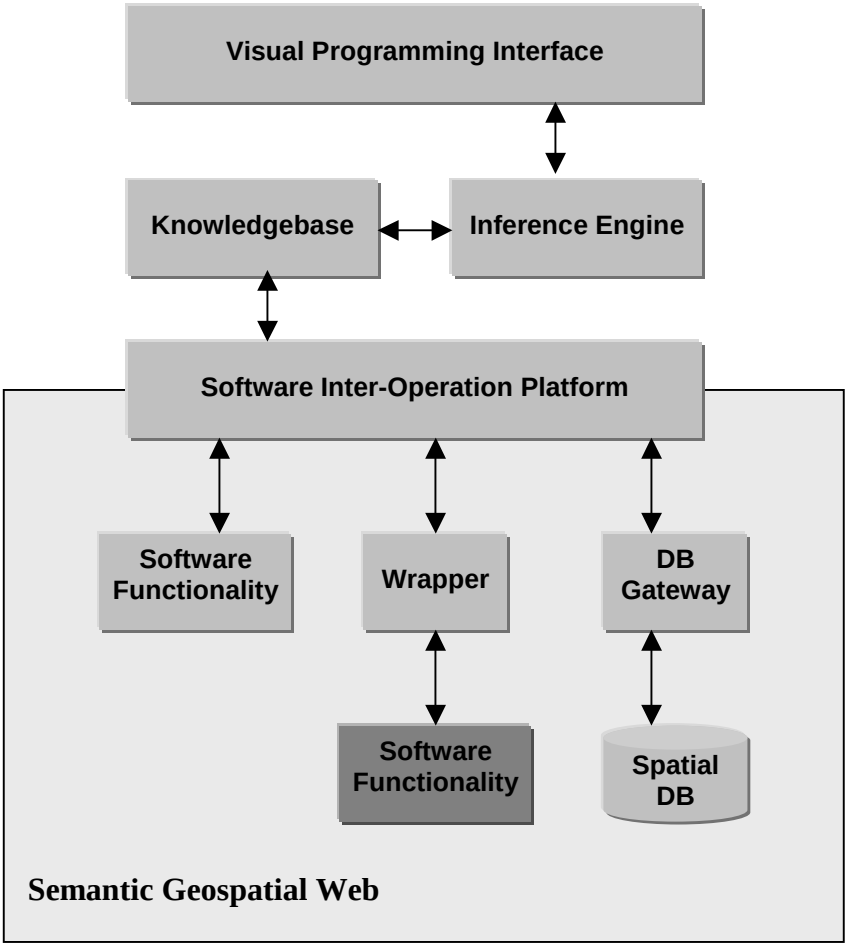


Figure 2.10 The general software architecture for a Geospatial PSE

It is noticeable that the knowledge-based system plays a central role in this framework, which manages the connections among software implementations, visual programs, and geographical understandings. While the implementation of the software

inter-operation platform and the building of visual programming interface can be achieved by drawing upon existing practices (e.g. those used by GeoVISTA *Studio*), the design of the knowledgebase and the development of the inference engine require a novel knowledge model that captures the nature of geographical problem solving. The main focus in the rest of this thesis, therefore, is to develop such a knowledge model.

2.5 Summary

This chapter combines and synthesizes the existing practices of GI technologies and scientific PSEs to present a general framework of Geospatial PSEs. GI technologies are computational methods that handle geospatially referenced information, and "geographical problem solving" includes all activities that use GI technologies to achieve geographical application goals. Despite the rapid development of GI technologies such as GIS and the establishment of GISci as a scientific field in the past two decades, currently an problem-centered problem solving system that provides users with friendly, controllable, and on-demand access to different GI technologies is still lacking, which handicaps the pursuit of geographical research and the dissemination of GISci. In scientific computing, a scientific PSE is a comprehensive software system that integrates different functionalities required by the entire span of scientific inquiry in an open, seamless, and automated manner. The development and application of scientific PSEs provide useful guidelines and valuable experience for the development of geographical problem solving systems, especially 1) the adoption of software reuse and loose-coupling integration technologies; 2) the design of visual programming interfaces; and 3) the use of knowledge-based systems. The enabling technology for GISci, therefore, can be designed as a Geospatial PSE, which is essentially a scientific PSE tailored for geospatial information and geographical purposes. More specifically, this chapter presents a feasible software design for Geospatial PSEs based on GeoVISTA *Studio*, a visual programming environment and problem solving system for GISci. The system design

includes 1) a software main framework based on the JavaBeans component standard, which can be used to integrate different GI technologies distributed as software components; 2) an open dataflow visual programming user interface for component composition and application construction; 3) An ontology-based knowledge modeling, representation and automation framework that conceptualizes both geospatial data and geographical problem solving processes. The next step for this study is to develop a knowledge modeling and representation strategy for geographical problem solving to guide the design of the knowledgebase and facilitate the implementation of automation.

Chapter 3

SEMANTICAL MODEL OF GEOGRAPHICAL PROBLEM SOLVING

3.1 Introduction

Semantics refers to aspects of meanings that are represented in a language (whether computational or human). In computer and information sciences, semantics is normally defined as the relations between computational artifacts and their underlying conceptualizations (Heiler 1995). While the open Geospatial PSE described in chapter two expands the scope and potential of geographical problem solving, it also raises new challenges in terms of building meaningful applications, because geographers now need to grasp the conceptual rationales of large collections of heterogeneous resources integrated by the system in order to use them. Since it is a non-trivial job for users to explore and learn numerous unfamiliar information sources and software tools, semantic problems may become a major bottleneck that hampers the efficient deployment and successful application of a Geospatial PSE. To overcome such difficulties, this chapter presents the concept of a Semantic Geospatial PSE, which is a system that explicitly captures, stores, and utilizes the semantics of computational resources to facilitate and enforce their meaningful applications. It reduces users' obligation to manually examine the complex conceptual structures of computational resources and allows geographers to use unfamiliar tools in a transparent way, which is particularly important given the open, heterogeneous nature of GI technologies distributed over today's Semantic Geospatial Web.

Based on the system architecture described in chapter two, a Semantic Geospatial PSE is essentially a Geopatial PSE that relies on a specially developed knowledge model

to encode geographical semantics and deliver knowledge-based supports. Therefore, the focus of this chapter is to develop such a semantic knowledge model. Section 3.2 discusses the roles of semantics in a Geospatial PSE, describes the semantic problems that users may encounter during a course of geographical problem solving, and lists the detailed requirements of a Semantic Geospatial PSE. Following that, section 3.3 analyzes the nature of semantics from the semiotic and formal perspectives to layout the theoretical ground of semantic modeling. Then section 3.4 presents a semantic model of geographical problem solving based on the triadic sign structure and first order model theory, and discusses how to uses the model to support meaningful geographical analyses. Finally, summaries are included in section 3.5.

3.2 Semantics of Computational Resources

Semantic problems arise when users cannot easily grasp or accurately comprehend the meanings of computational resources. This section defines three levels of semantics in a Geospatial PSE, describes the problems users may encounter at each semantic level, and proposes the basic approaches to solving them in a Geospatial PSE.

3.2.1 Three Levels of Semantics in a Geospatial PSE

In philosophical studies of semantics, there is a tradition originating from the American philosopher Charles S. Peirce that examines three modes of meanings, including those defined by firstness, secondness and thirdness (Peirce 1931-58). Firstness refers to the mode of being independent of any other subjects or objects. It describes the properties or qualities that define an *individual*, which could be a physical entity, an abstract concept, a dataset, a software tool, or any other independently existing things. Secondness is the mode of being referred to others. It constitutes the foundation of *relations*, which could be any nexus that connects two individuals. Thirdness is about *mediations*, or how individuals are related with each other to create new meanings. For instance, suppose we are using Arc/Info to map an epidemiological dataset about the

Appalachian region. Here the software "Arc/Info" and the given dataset are both individuals, and they are related by the activity of "mapping". The mediation is the entire situation that "Arc/Info is mapping the given dataset", which introduces a new meaning that neither the software nor the dataset alone can provide. Similar analyses can be performed at different scales, as Peirce argues that any secondness and thirdness can have their own firstness that may be related or mediated in other situations. For instance, at a fine-grained semantic scale, individual concepts such as "dataset", "epidemiology", and "Appalachian region" can be all regarded as manifestations of firstness. They are related with each other to define a dataset that describes the epidemiological phenomena in the Appalachian region. This mediation situation itself can be interpreted as firstness in a larger semantic context, e.g. as the particular dataset analyzed by a geographical knowledge discovery application. Based on this approach, the meanings of the world can be eventually modeled as a network that involves individual objects related with each other under different mediating contexts and at multiple semantic scales.

We can follow this general idea to identify different levels of semantics in a Geospatial PSE. More specifically, at the semantic scale that considers computational resources such as data sources and software tools as the basic units of meanings, three levels of semantics can be identified: 1) the first level semantics defines the meanings of individual resources; 2) the second level semantics defines the relations between resources, including those between digital datasets, computational methods, high-level concepts, and any other relevant objects; 3) finally, the third level semantics defines meaningful applications, which can be regarded as situations of mediation that combine a set of computational resources together through certain connections to serve new goals.

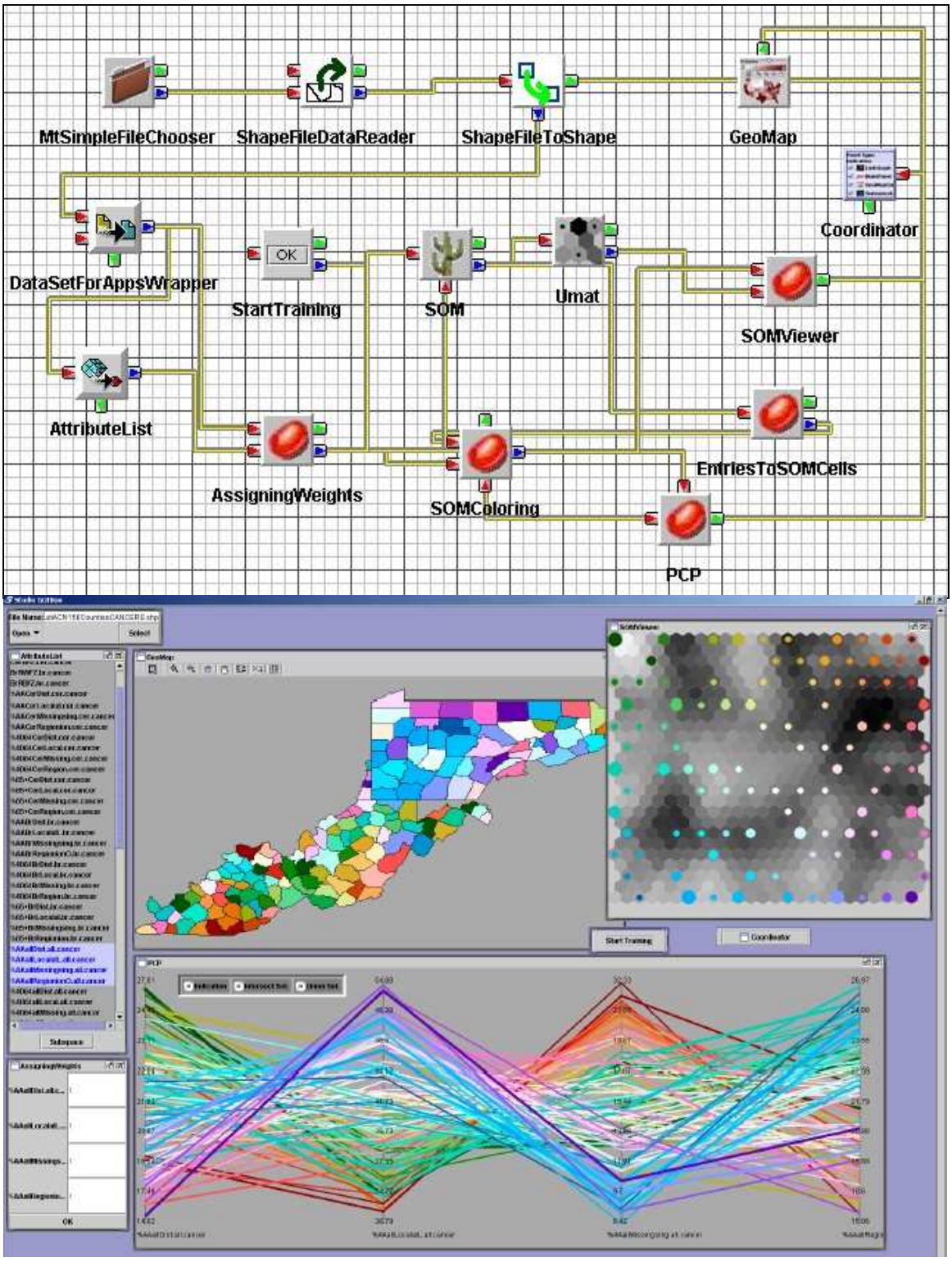


Figure 3.1 A data mining application in GeoVISTA *Studio*. The application analyzes a cancer dataset of the Appalachian region using a map, PCP, and SOM. The upper window depicts the dataflow network of the application (called a design in GeoVISTA *Studio*) in the visual programming interface, and the lower window contains the GUIs of the components.

We can illustrate the three level semantics using a concrete example. Figure 3.1 shows a data mining application constructed in GeoVISTA *Studio*, which combines map, parallel coordinate plot (PCP), and self-organizing map (SOM) to discover patterns of cancer distribution in the Appalachian counties (The GeoVISTA Center 2005). The application imports the cancer dataset stored in the ESRI shapefile format, visualizes it in the GeoMap and PCP components, calculates clusters over selected variables using the SOM component, and shows the results in a SOMViewer. In this example, the first level semantics describes the meanings of individual components such as GeoMap, PCP, and SOMViewer, including what those components can do and how they are distinguished from one another. The second level semantics defines how these individual components can be connected with each other. It may include the input and output data communication interfaces of the components, and other properties about their collective deployment and collaborations. Finally, the third level semantics describes the meaning of the entire data mining application. It states that the situation in which those specific components are connected in this particular way has created a new purpose, which is to perform data mining analysis over the cancer dataset collected for the Appalachian counties.

3.2.2 Semantic Problems of Geographical Problem Solving

The example above shows the main advantages of an open problem solving system, namely the flexibility of integrating heterogeneous tools and the freedom of building highly customized user applications. Nevertheless, while the system hides most syntactical details of Java programming, it still imposes a considerable semantic burden on the user side, requiring users to possess detailed knowledge regarding semantics at all three levels. More specifically, a user may encounter the following three kinds of semantic problems, including the problem of resource articulation, the problem of resource discovery, and the problem of solution construction.

3.2.2.1 The Problem of Resource Articulation

The problem of resource articulation is related to the first level semantics, which is concerned with how to convey the meanings of individual computational resources to their users. Currently the problem of resource articulation in a Geospatial PSE is mainly due to the absence of an effective means to convey the meanings of large amount of heterogeneous resources. In figure 3.1, for example, the visual programming environment provided by GeoVISTA *Studio* uses icons and textual labels to identify individual components. However, while iconic depictions are compact and intuitive visual aides for front-end user interaction, they generally lack the expressiveness and accuracy to describe computational resources in detail. In most cases labeled icons can only help users *recall* the functionalities of computational resources rather than clearly *define* such functionalities (Horton 1994), and the effectiveness of iconic representations usually cannot scale to hundreds or thousands of resources possibly existing in the tool repository of an open Geospatial PSE. In other words, the labeled icons in figure 3.1 can only help users "guess" the functionalities of components without any guaranteed accuracy. As a result, users have to repetitively refer back to software documentations for more detailed explanations, and manually interpret the meanings of individual components before using them. This is extremely inefficient and error-prone in an open, heterogeneous environment, where the amount of tools is huge and the meanings of resources are unforeseeable.

3.2.2.2 The Problem of Resource Discovery

Resource discovery is related to the second level semantics, which is about how to find relevant new resources according to their relations with known concepts or existing resources. Generally speaking, two types of resource discovery problems are commonly encountered: *concept-based discovery*, which looks for resources related to certain high-level concepts, and *context-based discovery*, which finds tools that can be connected with other resources already used in the application context. For example, imagine that we are

constructing the application shown in figure 3.1. The cognitive process of this task can be described as the follows. Firstly, we want to know what components in the repository can satisfy our needs for spatial clustering and cartographical visualization. This is an issue of concept-based resource discovery since we want to find tools performing those high-level tasks (where "performing" can be regarded as a relation). The results would include the SOM and GeoMap components. Now imagine we have already imported the dataset and extracted the polygon layer of the county boundaries using the ShapefileToShape component. The next step is to find which tools can process a polygon layer produced by this particular component. Context-based resource discovery is then needed to list all components whose input interfaces "match" the output interface of ShapefileToShape. In this case, GeoMap would satisfy such requirements. In summary, concept-based and context-based discoveries represent two complementary approaches to resource discovery, with the former focusing on the capability of tools and the latter emphasizing the compatibility of resources with a given application structure. Both of them are based on the relations between high-level concepts, data sources, and analytical tools. Due to the lack of resource discovery support, users have to manually search a large collection of resources and evaluate their complex relationships to locate relevant data sources or software tools, even when those computational resources are well articulated. This creates another major barrier to the meaningful use of a Geospatial PSE.

3.2.2.3 The Problem of Solution Construction

The problem of solution construction is related to the third level semantics, which focuses on how to construct meaningful applications (or fragments of them) from a set of individual resources. Solution construction in an open problem solving system can be very costly even with the support of GUI and visual programming, because although users do not need to conduct syntactical programming, they still have to examine the structural relationships between selected components in order to integrate them. In

addition to recognizing heterogeneous resources and selecting relevant tools, users also need to evaluate different possible ways of resource combination and configuration based on high-level applications goals to create the desired applications accordingly. This kind of work is essentially "conceptual programming", which obviously is not always the central interest of geographical problem solvers.

For example, in the data mining application depicted in figure 3.1, the most relevant application tasks are data clustering and visualization, and the components that users are most interested in are GeoMap, PCP, and SOM. However, in order to make use of those three components, a user must also examine other "auxiliary" components such as those loading and converting data, and spend significant time connecting them step by step. It is impossible for the user to quickly focus on what they are most interested in, and the correctness of the manually constructed application cannot be verified until all components have been wired in place. Such a process can easily become very tedious and time consuming when creating a large or even moderate-sized application, and it will be a significant drawback if the user's primary goal is geographical analysis rather than such kind of "conceptual programming".

It is noticeable that the problem of solution construction is not a problem caused by the visual programming paradigm itself. In fact any PSE with a front-end operational control interface is vulnerable to this problem. The bottom line is, if you want to support the creation of new tasks, then the operational structure of the tasks must be exposed to the creator, and unfortunately, in this case, the creators are geographers who are not always interested in such details. Having to be familiar with this level of details is somewhat akin to having to know how to connect together all the major mechanical components of a car before driving it to the supermarket. Such details will certainly detract from the real task at hand, which in the example above is the task of coming to understand the structure of cancer incidence and prevalence on the Appalachian landscape.

3.2.3 The Goals of a Semantic Geospatial PSE

To summarize, due to the lack of appropriate resolutions to those problems, an unfamiliar user of a Geospatial PSE has to constantly browse the entire collection of software components (which is open and therefore may keep changing), thoroughly read software documentations (if there are any) about different components, carefully select relevant tools according to her own understandings (which could be incorrect or incomplete), and manually connect the selected components to build a desirable application (which is tedious and error-prone). This inconvenient and unproductive process significantly precludes domain users, especially those with less technical background, from fully exploiting the power of the system in a timely manner.

A Semantic Geospatial PSE is a software system that provides explicit and automated solutions to the semantic problems described above. More specifically, a Semantic Geospatial PSE captures and represents semantics of all three levels in a system database or knowledgebase, and uses such information to ease resource recognition, facilitate resource discovery, and automate the development of meaningful applications. It provides users with:

1. Resource catalogue services based on carefully designed high-level descriptions, where the meanings of heterogeneous resources are accurately articulated and clearly conveyed to human users;
2. Resource discovery services based on search and query tools, with which users can easily find relevant resources according to their needs;
3. Solution building services based on AI techniques, which automatically construct high-level applications (or fragments of them) according to users' specific goals and requirements.

To achieve these goals, an appropriate knowledge model that captures the semantics of heterogeneous resources at all three levels must be developed.

3.3 The Nature of Semantics

Understanding the nature of semantics is critical to the development of the desired semantic knowledge model. Semantics, or meanings, is perhaps one of the most studied subjects in the history of human inquiry, which has drawn broad interests from philosophy, linguistics, mathematics, logic, psychology, computer science, and many other fields. This study exploits the results from two major strands of studies, including the sign-oriented view from semiotics and the formal approaches originated from philosophical and mathematical logic. As the study of sign and sign systems (Chandler 2001), semiotics examines semantics mainly from the relationships between a representation (the signifier) and the represented (the signified) object. On the other hand, formal methods are more concerned with the internal structures of abstraction and emphasize on the role of well-defined structures in the development of precise, consistent, and verifiable meaning. Semiotics and formal methods, therefore, provide sound methodologies for analyzing the first and second level semantics, where the meanings of individual computational resources can be studied as signs, and semantic relations between resources can be modeled as rigorous formal structures. Finally, a solid framework for capturing the third level semantics can be obtained by combining the semiotic and formal approaches under the context of geographical problem solving.

3.3.1 The Semiosis of Software Systems

The foundation of semiotic studies is sign and sign systems. Generally speaking, anything can be regarded as a sign once it is interpreted by someone as certain kind of representation, i.e. referring to or standing for something other than itself. The development of meaning, therefore, can be regarded as a "signification" process that links a sign with what the sign represents (Chandler 2001).

3.3.1.1 Dyadic and Triadic Sign Models

There are two classical models of sign: the dyadic model offered by Swiss linguist Ferdinand de Saussure (Saussure 1974) and the triadic model proposed by American philosopher Charles S. Peirce (Peirce 1931-58). Figure 3.2 depicts these two different but inter-connected models of sign.

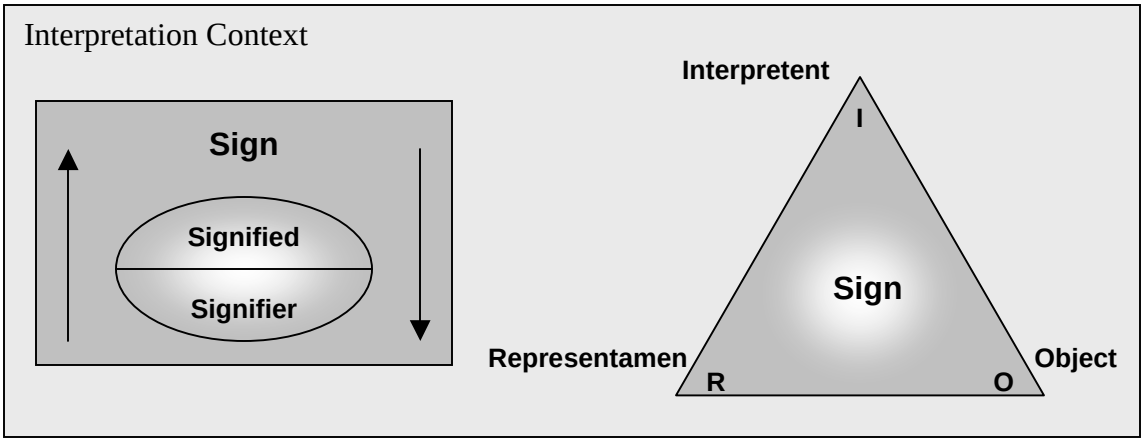


Figure 3.2 The dyadic and triadic views of a sign. Both of them believe that a sign is always situated in certain interpretation context.

In the dyadic model, a sign consists of a signifier and a signified concept. The signifier is the representational form of the sign, and the signified is the mental concept conveyed by the signifier. For example, if we consider the term "classification" as a sign, then the English word "classification" is the signifier, and the concept of classification in our mind is the signified. As shown by the left picture in figure 3.2, the relationships between the signifier and signified are two-way, where the upward arrow denotes the process of signification and the downward arrows refers to the process of representation. It should be noted that the dyadic model does not provide any direct linkage between a sign and a material "thing". In other words, the signified is always a concept, and the concept itself can stand for another concept (in another sign). How the concept is connected with the material reality is not addressed explicitly in the dyadic model,

because in Saussure's approach, signification only happens when reality is perceived and conceptualized (Saussure 1974).

The triadic model of sign resembles the dyadic model in many aspects, except the addition of "Object" (or referent), which links the sign with the thing it stands for. In the triadic model, Representmen (or sign vehicle in many literatures) corresponds to the signifier in the dyadic model, and Interpretent (or sense) can be associated with the concept signified by the sign. However, Peirce believes that what a sign stands for should be distinguished from the "sense" it makes in the interpreter's mind, therefore, he uses Interpretent and Object to separately denote the signified concept and the represented and conceptualized thing. In other words, the dyadic model views the signification of a sign as a pure psychological construct, while the triadic model admits the possible connection between a sign and the material reality it represents. Nevertheless, it should be noted that Object in the triadic model is not always a material thing, although it could be. In fact both Object and Representamen in a sign can be anything, including concepts, material realities, or even another sign. Based on this view, Peirce argues that the idea of semiotics can be used to analyze the relationships of any physical or mental constructs at multiple scales (Peirce 1931-58).

3.3.1.2 The Analysis of Sign Systems

Saussure and Peirce share some common principles about sign and sign systems, some of which are particularly important to this study. First, a sign could be made up from smaller signs, and the sign itself might be a constituent of a larger sign system. For example, in the English language, letters can be combined into words, words make up sentences, sentences constitute paragraphs, and paragraphs can be organized into an article. Similarly, if we treat Representamen, Interpretent, or Object from the triadic model as signs themselves, we can identify their own Representamen, Intepretent, and Object, and by doing so recursively we can analyze a sign system to an arbitrary detail

level. Such an idea of hierarchical sign assembly has been grasped by geoscientists and geographers to analyze geospatial knowledge at multiple spatial, temporal, and conceptual scales (Brodaric and Gahegan 2001). From the perspective of a Geospatial PSE, this recursive, multi-level view is compatible with the three-level analysis of meanings described before, where individual computational resources can be regarded as signs composed from objects, concepts, and data structures at finer detail levels. For example, the software component GeoMap can be semantically defined by a triple sign relation, where Representamen is the JavaBeans interface, Object is the (intended) runtime behaviors of the component, and Interpretent is the concept of the "mapping" functionality understood by users. In addition, those interfaces, runtime behaviors, and user concepts can be further decomposed into more detailed sign systems, e.g. the component interface is essentially an aggregation of methods and data parameters, the meanings of which are defined by its own sign relations.

Second, both the dyadic and triadic models recognize the importance of context. A sign is only a sign when it is interpreted (Peirce 1931-58), and what the sign stands for is subject to the context of sign interpretation, including the larger sign system where the sign is currently situated in, and the social conventions that govern signification. When a representation (signifier) is interpreted in different contexts, it develops into different signs because it may signify different concepts and / or stand for different things. For example, the word "football" usually signifies different sports when appearing in British and American newspapers. Similarly, computational resources such as software components signify different things when interpreted by human users and software systems, if we regard software execution as one kind of legitimate interpretation. From the user's perspective, GeoMap stands for a tool that produces geographical maps. However, for a Geospatial PSE that actually executes the component at the runtime, it means a software entity that receives, processes and outputs data blocks conforming to the JavaBeans component standard. Both perspectives are important to the meaningful

application of the component. But it is evident that users and runtime systems reside in different contexts, and consequently they are interpreting the same representation but different signs. It would be inappropriate to treat the component as a single sign with a static interpretation, and it would be unnecessary for users to understand the input / output data structures required by execution.

3.3.1.3 Computational Systems as Signs

The influence of semiotics on computational systems is substantial. For example, semiotic methods have been employed to model human computer interaction (HCI) (Souza 2005), guide user interface design (Goguen 1999), and study the meaning of computer programs (Andersen 1997). In geography and geosciences, semiotics has been employed to study map understandings (MacEachren 1995), model geoscientific knowledge development (Brodaric and Gahegan 2001), and examine the general epistemology of geosciences (Baker 1999). From a semiotic perspective, a Geospatial PSE is essentially one kind of computational representation, which is in digital form and signifies intended computational operations to both their users and the machine running them. Figure 3.3 depicts the general signfication processes of software systems under the contexts of system execution and user interpretations, which regards the system resources as signs interpreted by both the runtime machine and the user. System resources such as software components are the representational forms of information sources and analytical functionalities. The lower-right triangle in the picture describes a sign interpreted by system users, which signifies the users' understandings about the machine operations. The upper-left triangle depicts the sign interpreted by the runtime machine, which signifies a set of implementation objects recognized by the underlying platform, e.g. the JavaBeans runtime objects instantiated by GeoVISTA *Studio*. Both of them are linked with a set of intended machine operations or desired system behaviors, which are Objects or referents of both signs.

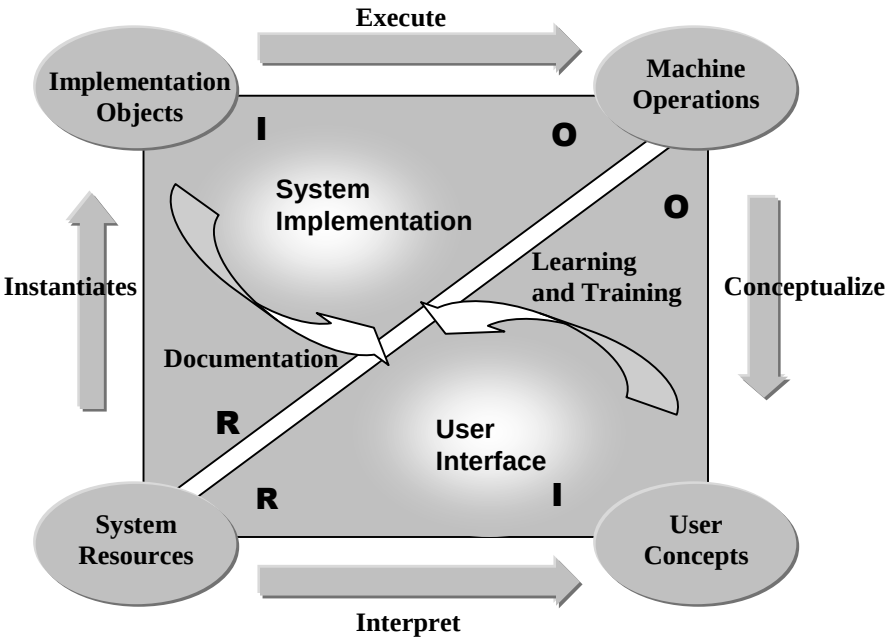


Figure 3.3 The semiotic processes of software systems, which are inspired by Anderson (1990). The capitalized letters R, I, and O denote Representamen, Interpretent, and Object respectively.

3.3.1.4 The Proxy Representation of Computational Resource

The semantic difficulties of user interpretations arise in a Geospatial PSE mainly due to the fact that users must interpret the representational forms created by resource providers (e.g. labeled icons, names of methods), which are seldom designed to convey complex meanings in an open, heterogeneous environment. Furthermore, since such representational forms are usually decided at the development time and remain fixed afterwards, they cannot signify different roles that a computational resource may play under different application contexts. For instance, in the aforementioned data-mining application the component GeoMap is regarded as a data exploration tool that can help users discover cancer distribution patterns interactively. But in other application scenarios, the same component may be only used to produce maps of the targeted geospatial phenomena, i.e. as a pure communication or presentation tool. Despite the different roles that the component may play, users have to interpret the same

representational forms (the labeled icon), memorize the intended meanings of the GeoMap component, and manually connect such knowledge with the specific problem contexts according to their own understandings. This might be less a problem for a standalone system where resources are centrally managed and their functionalities are limited and fixed. However, as shown in figure 3.4, since an open Geospatial PSE usually provides a potentially unlimited number of heterogeneous resources with unforeseeable meanings (contrasting to the closed toolboxes in traditional systems), memorizing the semantics of computational resources is exponentially complicated and as a result, the PSE system itself becomes extremely difficult to understand and use.

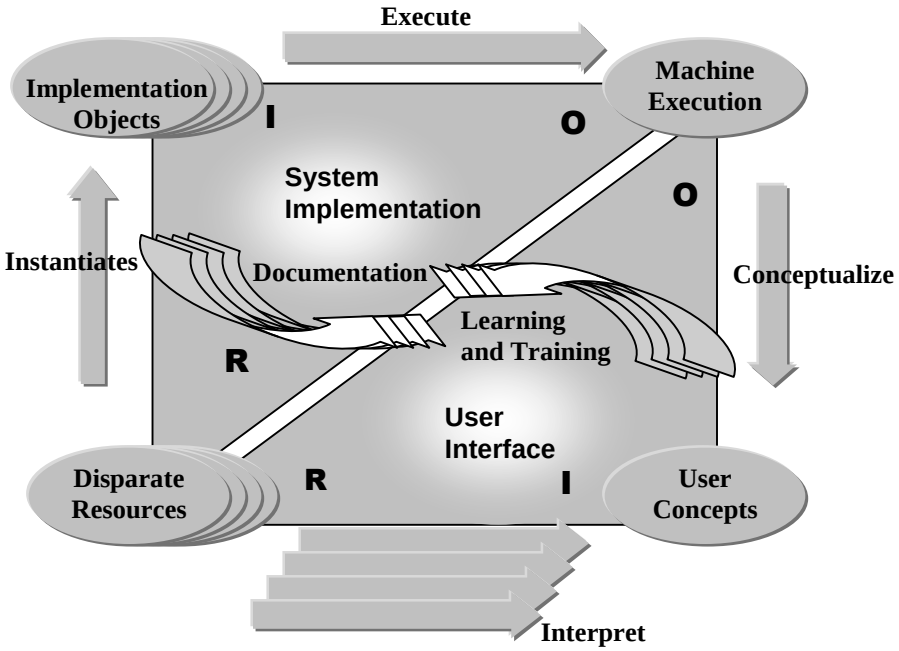


Figure 3.4 The semiotic processes of an open Geospatial PSE. Due to the potentially unlimited number of unforeseeable heterogeneous resources, complexity of learning is multiplied.

To overcome this problem, the idea of "proxy" representations is proposed to combine different representational forms of computational resources. More specifically, a proxy representation is an abstract notion of a computational resource, which contains three elements:

- **Implementation proxy** is a representation of the computational resource' implementation symbols or identifiers under a given software platform (e.g. Java class names)
- **Runtime proxy** is a representation of the runtime role the resource may play under a given runtime model (e.g. a data processing node in the dataflow model).
- **User proxy** is a representation of the high-level concept the resource should convey to users (e.g. the high-level task a component performs).

Table 3.1 The basic elements of a proxy representation.

	Purposes	Examples
Proxy Representation	An abstract unit that aggregates different representational forms for a computational resource.	The GeoMap component
Implementation Proxy	A representational form that targets on system implementation.	The name of the Java class that implements the component.
Runtime Proxy	A representational form that targets on a specific runtime model.	The nodes and pipelines in a dataflow.
User Proxy	A representational form that targets on a specific kind of application contexts.	The icon  ; The concepts of "data exploration" or "mapping".

One can think of a proxy representation as a complex symbol with three "facets", which integrates signifiers for three different interpretation contexts (table 3.1). Figure 3.5 depicts the general semiotic processes of a proxy representation. With the implementation and runtime proxies, the system can instantiate the concrete software module associated with the proxy representation, and execute it based on the resource's runtime role. On the other hand, users can comprehend the resource's meaning by examining its user proxy. It is noticeable that the runtime proxy is interpreted by both the system and the user, which means that the representation of runtime roles must be both executable and human readable. Fortunately, under the dataflow-based visual

programming model this is seldom a problem, since executable dataflows can be easily understood by human users.

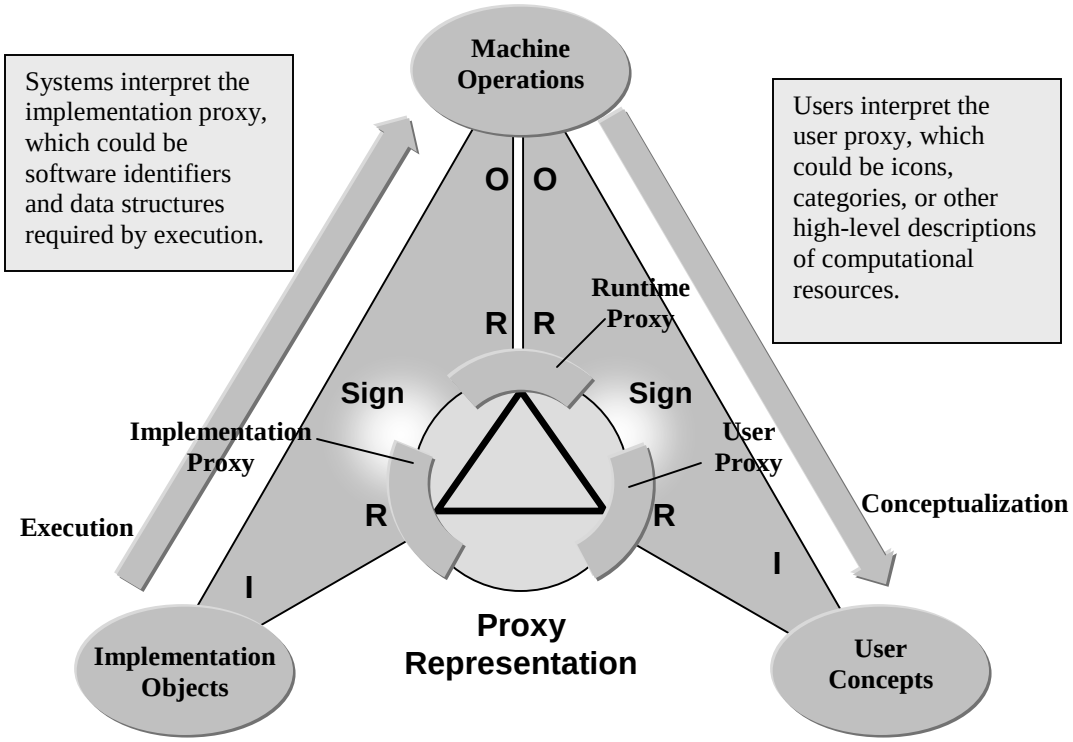


Figure 3.5 The semiotic processes of a proxy representation, where different interpreters, including users and systems, can focus on their correspondent representational forms. The linkages of different representational forms are aggregated and maintained by the proxy representation.

It is noticeable that the central purpose of a proxy representation is *not* to guide the development of a particular kind of representation (e.g. icon design or component implementation), but to provide a way to combine different kinds of representations flexibly. The main advantage of the proxy representation lies in the fact that the implementation, runtime and user proxies can be designed and constructed independently, and the system can choose the most suitable proxies according to the given problem solving context. Figure 3.6 demonstrates such flexibility, where the abstract notion of a GeoMap component can be implemented on different software platforms, executed under different runtime models, and presented to users via different forms. In other words, in the most general sense, proxy representations do not depend on

a particular software implementation platform, visual programming model, or high-level descriptions, and they are portable between different PSEs.

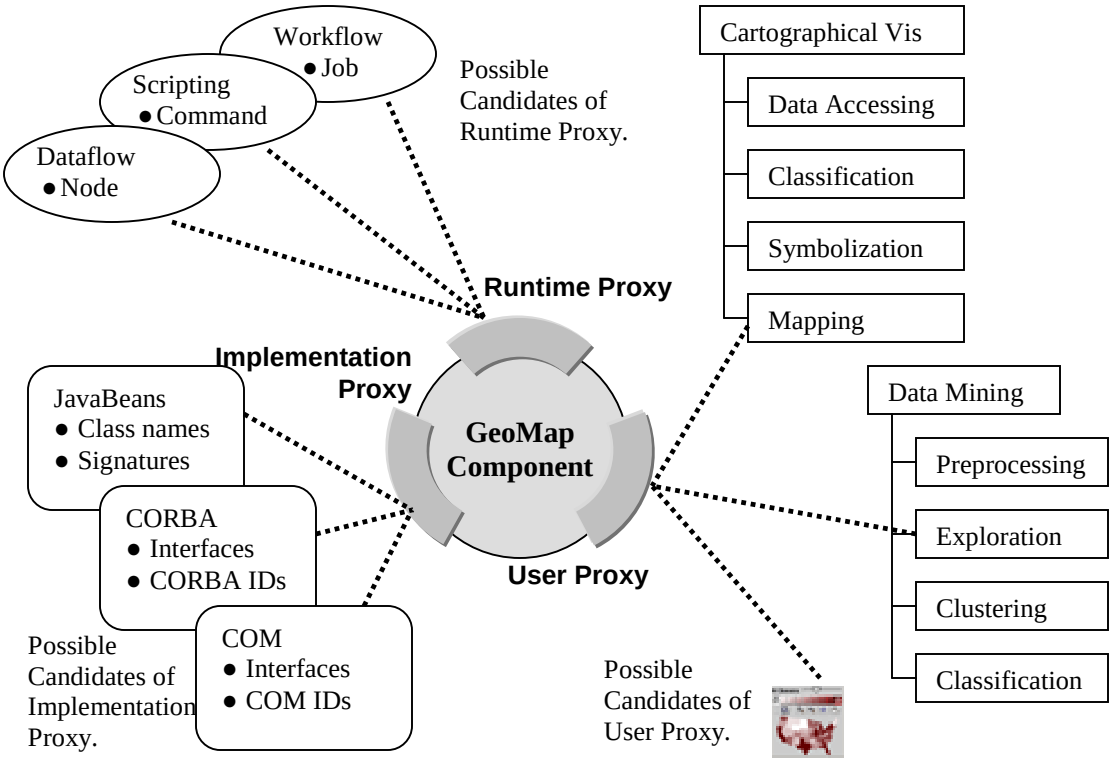


Figure 3.6 An example proxy representation for the GeoMap component. Different candidate proxies can be used according to different implementation platform, front-end programming model, and user application contexts.

For a given Geospatial PSE where the software implementation platform and runtime model are already defined by the system, the user proxy can vary in different user interpretation contexts. For example, as shown in figure 3.6, we can use the labeled icon as the user proxy for visual dataflow editing, while adopting a concept from an ontological category to facilitate user understanding and resource discovery. Furthermore, we can choose different categories according to different application contexts, e.g. using a data-mining related ontology for geographical knowledge discovery and employing a cartographical ontology for "pure" map-making tasks.

In summary, while a proxy representation itself does not create any new representational forms for a computational resource, it allows one to flexibly select and

integrate the most suitable or preferred existing representations to signify meanings to both computers (in terms of machine execution) and human users (in terms of human interpretation). The resource articulation problem can then be resolved by associating selected high-level descriptions (icons or ontologies) with proxy representations of computational resources. Based on the example in figure 3.6, one can imagine a resource catalogue service back-driven by shared ontologies of high-level tasks (e.g. those for data mining and cartographical visualization), where the concepts in the ontologies are connected with component implementations (which could be CORBA, COM, or JavaBeans, depending on the implementing platform) and runtime roles (e.g. a command string, a dataflow node, or a workflow job) through proxy representations.

3.3.2 Formal Methods of Semantic Analysis

Formal methods were originally devised in symbolic logic to study the structural relationships between logical arguments. Logic studies the existence of truth, and the term "formal" comes from the philosophical concept "form" (in contrast to "content"), which is used here to indicate that the truth values of logical statements are decided by their symbolic structures rather than their own meanings. At the first glance, formal logic seems to be opposed to semantics. However, the formal approach focuses on the structural (or syntactic) interpretation or derivation of meanings, which reveals an important aspect of semantics that is less addressed by semiotic studies. We can use a general linguistic example to illustrate the importance of structure or syntax to semantics. Consider the semantics of a natural language such as English. At the individual word level, the meaning of the English language is defined by its vocabulary. Nevertheless, when different words from the original English vocabulary are composed into a sentence, the meaning of the sentence is not equivalent to the simple summation of all its constituent words' meanings. Instead the development of sentence meanings is dependent on the grammatical structures that govern the combination of individual

words. In other words, grammars not only define the rules to compose words into syntactically correct sentences, but also specify how to construct and derive new meanings based on the structural combination of individual words. Although semioticians such as Saussure and Peirce recognize the importance of sign assembly and sign systems, the structural construction and derivation of meaning were not systemized in a rigorous way until the advent of formal theory in the first half of the 20th century and its subsequent introduction to semantic analysis (Chomsky 1986).

3.3.2.1 Formal Systems

The central concept behind any formal analysis is the notion of a formal system (or formal language), which is a symbolic system that contains the following elements (Jeffrey and Burgess 2006):

1. A finite set of symbols that can be used to construct formulae.
2. A formal syntax or grammar that strictly defines how to construct formulae. A formula conforming to the given formal grammar is called a well-formed formula (WFF).
3. A set of axioms, which are WFFs regarded to be unconditionally true in the formal system. The number of axioms can be infinite, as we can use certain axiom schemata to define an unlimited number of related axioms. The axioms of a formal system are computationally enumerable if they can be listed by an algorithm.
4. A set of inference rules, which defines the relations between WFFs, especially how to derive new WFFs from existing ones. An inference rule is also a WWF.
5. A set of theorems, which include all axioms plus all WFFs that can be derived from existing theorems. This definition recursively encompasses all WFFs

that can be deduced to be true given the axioms and inference rules of the formal system.

Symbolic logical languages such as the first order predicate calculus are formal systems. In the first order predicate calculus, for example, legal symbols include a finite set of logic connectives and a finite set of symbols that can be used to denote constants, variables and predicates. It also has a formal grammar that restricts the construction of first order formulae and a set of inference rules that define logical (deductive) implications. In this study we will mainly use first order predicate calculus as the basic formal system. *Appendix A* to this thesis contains a detailed explanation of the syntax and notations of the first order predicate calculus.

Formal systems can be used to structurally define the second level semantics of a Geospatial PSE. The relations between different computational resources can be strictly encoded as logic statements, which can be validated and queried based on logical deduction. For example, "finding tools that perform pattern classification" can be transformed into the problem of making the following first order predicate true:

$\exists ?x$ accomplish *(?x, Clustering)*

In this predicate, "*?x*" is a variable representing an unknown resource, the symbol "$\exists$" is an existential quantifier that means "there exists an *?x*", and "*Clustering*" is a constant denoting the concept of pattern classification, which for example may be defined by a data mining ontology. Finally "accomplish" is a predicate symbol stating that "*?x*" performs the task "*Clustering*". The desired resource can be discovered by logically proving the above predicate based on known facts specified in the same way. For example, if we know the component SOM can support the pattern classification task, we can represent this fact as the following predicate:

accomplish*(SOM, Clustering)*

Proving the formula "accomplish *(?x, Clustering)*" with the above fact will generate the following result:

accomplish*(?x/SOM, Clustering)*

The construct "*?x / SOM*" denotes a variable assignment (or substitution), which means substituting the variable *"?x"* with the value "*SOM*".

3.3.2.2 Semantics of Formal Systems

One important issue for formal semantic analysis is how to connect the structural system defined by formal logics with real world meanings. For example, how can meaningful individuals, such as the concept of clustering and the component of "SOM", impact the meaning of a logical statement (essentially a structural relation)? Historically, it was once believed that there could be a universal formal language L_U, which was able to consistently formalize all kinds of knowledge, including truth definitions of the language itself and axioms of all domains, and thus should solve all semantic problems (Dodig-Crnkovic 2006). However, such a universal formal system has proved to be impossible in first order logic. A famous counter-example is the "Liar's Paradox" (Barwise and Etchemendy 1987), which can be intuitively described by a man (the liar) stating "I am lying now", or more generally, exemplified by the sentence below:

"This sentence is not true."

This statement is self-contradictory because if what the sentence states is true (the man is lying) then the sentence itself must be untrue (the man is lying in saying that he is lying). The root of this paradox is that we are trying to use a statement to state the truth value about the statement itself, and generally speaking, any statement that circularly refers back to its own truth value will potentially encounter the Liar's Paradox. Godel (1992) and Tarski (1944) have separately discovered that a first order formal system with a set of computationally enumerable axioms cannot consistently define the truth values of all formal statements possibly existing in the system. In intuitive words, for any first order formal system *L*, no matter whether *L* is a ground language such as the first order predicate calculus or a language with domain axioms, if all axioms of *L* can be listed by

an algorithm, then there must be certain facts that cannot be proved to be true or false based on the axioms and inference rules provided by *L*. For instance, in the Liar's Paradox, the "liar" cannot use a statement, which can be regarded as a language, to state the truthfulness of the statement itself. Consequently, the semantics of logic statements are *conditional* and *language-dependent* in nature, or in other words, different languages may have different scope and different truth definition power over different subject matters.

Godel and Tarski's findings have profound impacts on the philosophy of science in general and the analysis of semantics in particular. If a universal formal language did exist, then all scientific disciplines would be eventually reducible to mathematics and physics, since all subject matters from different disciplines could be formalized in a self-contained manner, where external definitions of meanings and human interpretations were unnecessary (just imagine a geographical field that was entirely specified in terms of physical laws). As such a universal logic has proved to be unattainable, semantics must be defined based on interpretations outside the logical system itself. Semantics of formal systems, therefore, are always conditional, and the meanings of logic statements are essentially defined by their *truth conditions* decided by the external subject matters they specify. In other words, the meanings of scientific domains such as geography are not only dependent on their relations with physical laws, but also decided by the specific interpretations and understandings from the particular subject domains.

3.3.2.3 Model Theory

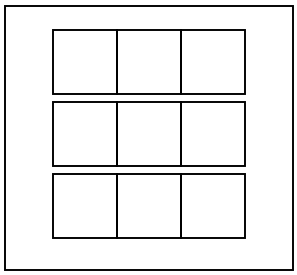
The issue of formal semantics then becomes a problem about how to specify the truth conditions that different subject domains may impose on a logical system. The semantics of formal systems have been systematically theorized under different approaches, and the most well known ones include Tarski's T-schema and semantic theory of truth (Tarski 1944, 1983), Davidson's truth-conditional semantics (Davidson

1969), Gentzen's (1934), Prawitz's (1965) and Dummett's (1996) proof-theoretic semantics, Kripke's semantics for modal logic (Kripke 1963), and his revision to Tarski's theory (Kripke 1975). In this study we adopt the model-theoretic approach based on Tarski's semantic theory of truth, a rigorously defined and widely recognized theory that distinguishes a formal language from the structures that define the truth condition of the language (Tarski 1983). The former is called an object language, which can be used to state factual propositions, and the latter constitute the semantics of the object language, which is usually defined in a meta-language other than the object language itself.

The basic idea of model theory is to define a formal structure that connects logical statements with meaningful objects coming from certain subject domains. Model theory assumes that those meaningful objects can be described by an enumerable set called a domain of discourse (DoD). While the meanings of a DoD is defined by the targeted domain outside the logic itself, the formal semantics of logical statements built upon the DoD (i.e. predicates that take elements from the DoD as parameters) can be specified as *conditional rules* that constrain their truth values. For example, suppose the GeoMap component requires a geospatial dataset as its input. Here "GeoMap", "dataset", and "geospatial" are all meaningful objects from a DoD. The concept of "input" is a relation that connects "GeoMap" with "dataset". In model theory, such a relation is considered as a formal structure, which can be represented as predicate symbol "*input/2*". The word "*input*" is the predicate name, and the number *"2"* is the predicate's arity, which means the predicate always has two parameters. Subsequently, formal semantics can be expressed as the conditional rules that decide the truthfulness of all predicates instantiated by this predicate symbol, i.e. predicates in the form "*input(X, Y)*", where "*X*" and "*Y*" are any elements from the given DoD. In this example, a straightforward rule is to require that "every input dataset of GeoMap must be geospatial". In other words, any datasets that do not satisfied this condition cannot be inputs of GeoMap. Consequently, the predicate *"input(GeoMap, DataSet)"* is true only if "*DataSet*" is also specified as a

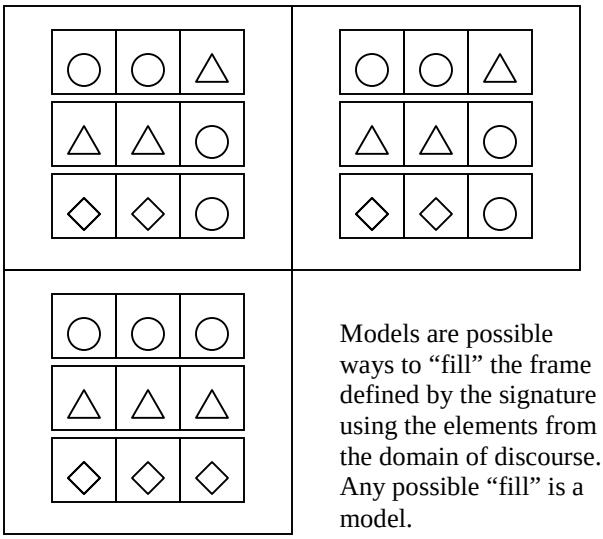
geospatial dataset somewhere in the formal system. Based on this approach, model theory eventually translates the issue of meanings into a correspondent issue of logical conditions, which is verifiable, deducible and computable under today's automated theorem proving systems.

Let L denote the ground first order language, which follows the basic syntax and inference rules of the first order predicate calculus. Now imagine that we are using L to encode the structure of a particular application domain. Knowledge about the domain can be specified as logical formulae based on a specific set of non-logical terminology, including constants, predicate symbols, and function symbols (similar to predicate symbols, a function symbol is also a name plus an arity number, but with a "return value"). The set of non-logical terminology constitutes a *signature*, denoted by K. A language L_K is a first order language that respects all syntactical and inference rules of L, but only uses the non-logical terminology from K to construct logical formulae. One can think of models as all possible ways of "interpreting" the language L_K over a *DoD*. Intuitively, a *K-model M* can be regarded as a set of fully instantiated predicates from the signature K that only take elements from the *DoD* as parameters. If a new *K-predicate* (a predicate constructed using a predicate symbol in K) p is *contained* in the model after substituting all its free variables with some elements of *DoD*, we say the model *satisfies* the predicate or is a *model of the predicate*, denoted by $M \vDash p$. The satisfaction of composite logical formulae (i.e. those constructed with logical connectives) can be defined recursively based on the satisfaction of a single predicate (see *Appendix B* for details). In addition, if a model M satisfies a set of formulae under the same group of variable substitutions, we say the model satisfies the *logical theory* Γ consisting of those formulae, and *M is a model of* Γ, denoted by $M \vDash \Gamma$. Finally, for any logical theory Γ and logical formula φ, if all models of Γ are also models of φ, we say Γ semantically entails φ.

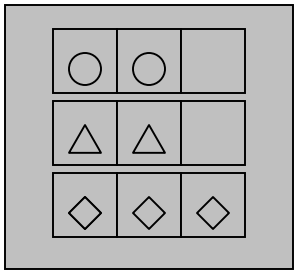


Signature is the basic "frame" of representation, which contains empty predicate and function symbols.

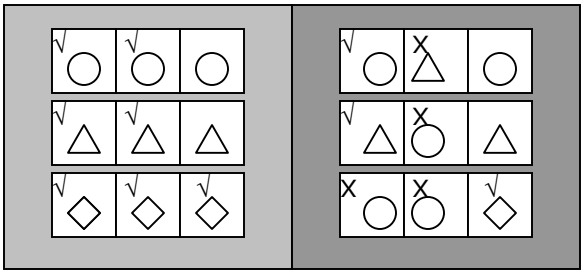
Domain of discourse provides the elements to "fill" the frame.



Models are possible ways to "fill" the frame defined by the signature using the elements from the domain of discourse. Any possible "fill" is a model.



Conditional rules are axioms that define what kind of models are allowed in the language.



Consistent Knowledge

Inconsistent Knowledge

Factual knowledge includes axioms that state facts collected from the domain, which must obey the consistency and inference rules.

User queries are partially instantiated "frames" where unknown "fills" are denoted by variables. Only models that satisfy both existing axioms (including consistent rules and factual knowledge) and the "partial frames" from user queries are valid. Those valid models are used to fill the variable positions of the user query.

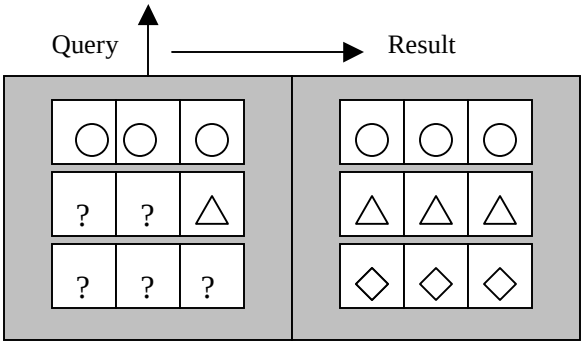


Figure 3.7 The diagrammatic explanation of model theory: how model theory can be used to support semantic knowledge representation and queries.

While a more technical explanation of model theory is included in *Appendix B* to this thesis, figure 3.7 provides an intuitive pictorial description of model theory. One can think of the shapes in the picture as different computational resources in a Geospatial PSE, and regard the block forms as the potential relations that can be used to relate or connect the shapes together. Models are concrete cases of relations, which can be obtained by "filling" all the blank forms with the given computational resources. Conditional rules defines the specific kinds of "fills" that are considered semantically true (valid), and users can use those rules to verify whether a particular case of a relation is valid, or query all valid cases for a given relation. Syntactically such queries are same as logical theorem proving, but semantically they are associated with meanings carried by the underlying DoD, as conditional rules are always defined according to understandings about the DoD. Therefore, we can say that the formal semantics of any language L_K is defined by its conditional rules.

3.3.2.4 Formal Knowledge Model

Based on model theory, we can build a formal framework of semantic knowledge modeling as depicted in figure 3.8. In Tarski's terms, L_K is the object language syntatically defined by the constants, predicate symbols, and functional symbols from the signature K. L_K's semantics is defined by a set of axioms, which constituate a logical theory that is regarded to be true in ANY case of knowledge modeling. In most cases, the semantics of L_K can be classified into two categories: *inference rules* that define how to deduce new facts according to existing ones, and *consistency rules* that constrain the validity of logical statements. Both of them can be constructed as first order logical formulae. For example, an inference rule may state that "if a component can produce a map then it is also can perform visualization", because according to our understandings, "mapping" can be regarded as one kind of "visualization". On the other hand, an consistency rule can be used to define that "if a component can produce a map, then it

must take a geospatial dataset as input". Syntactically, both rules take the form of "If A then B", where A is the premise and B the conclusion. However, the reasoning of an inference rule is forward (from A to B), as its purpose is to deduce new knowledge, and the reasoning of a consistency rule is backward (from B to A), as its goal is to check the satisfaction of the premise. Finally, axioms of factual knowledge contains facts from a particular knowledge model. For example, the logical statements that specify the properties of the GeoMap component are factual axioms. They may include predicates that describe the task that GeoMap can perform and the input / ouput data it requires. The validity of factual axioms are subject to the semantic axioms specified in the previous layer, where new facts may be deduced based on the inference rules, and the correctness of existing facts are checked by the consistency rules. It is in this sense that a specific body of factual knowledge is semantically meaningful.

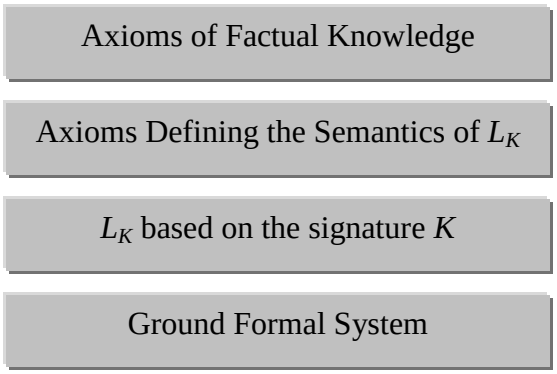


Figure 3.8 Semantic knowledge modeling based on model theory

In summary, formal systems such as the first order predicate calculus provide a means to define structural relationships among meaningful individuals (as predicates and functions), and offer the necessary tools to derive meanings from such structures (e.g. through logical deduction). In addition, model theory connects the structurally derived meanings with the meanings of individual objects by defining truth conditions of logical statements (as consistency and inference rules). A framework of formal semantic

knowledge modeling, therefore, can be constructed based on formal logic and model theory.

3.3.3 Semantics of Geographical Problem Solving

Representing the third level semantics requires us to associate a proxy representation and its formal semantics with specific application contexts. Peirce argues that thirdness is created only when means is connected with ends, or when firstness and secondness are linked with a purpose (Peirce 1931-58). In a Geospatial PSE, this means we need to select and adapt appropriate AI techniques (means) to build automated tools that reason with proxy representations (firstness), logical axioms (secondness), and produce results that satisfy the different goals of geographical problem solving (purposes). The application of AI methods to geographical problem solving is a complex topic in its own right, which will be the main subject of Chapter four. The following paragraphs, on the other hand, are focused on describing the kinds of situations in geographical problem solving, which can be used as conceptual guidelines for the development of automated problem solvers. One can regard the subsequent discussions as the *requirements* that the automation service in a Geospatial PSE should meet.

3.3.3.1 The Iterative Nature of Geographical Problem Solving

Geographical problem solving is an iterative process by nature, during which problem solvers approach the entire problem space through multiple stages, and repetitively modify or adjust their activities at each stage according to newly discovered information (Gahegan and Brodaric 2002). For instance, it would be naïve to regard the construction of the data mining application in figure 3.1 as a static linear process that can be developed from top to bottom in a single-threaded manner. As shown in figure 3.9, users often create such kind of applications incrementally and progressively, with considerable back-and-forth to re-collect datasets, re-explore the data spaces, and re-configure the clustering strategy or classification method(s). At each step a researcher

may need to evaluate the currently used tools according to the feedback from later steps, which could result in new rounds of resource identification, resource discovery and task construction. As a result, building a frond-end solution is often a highly interactive process and there is little chance that a "do-it-all" solution can be obtained immediately.

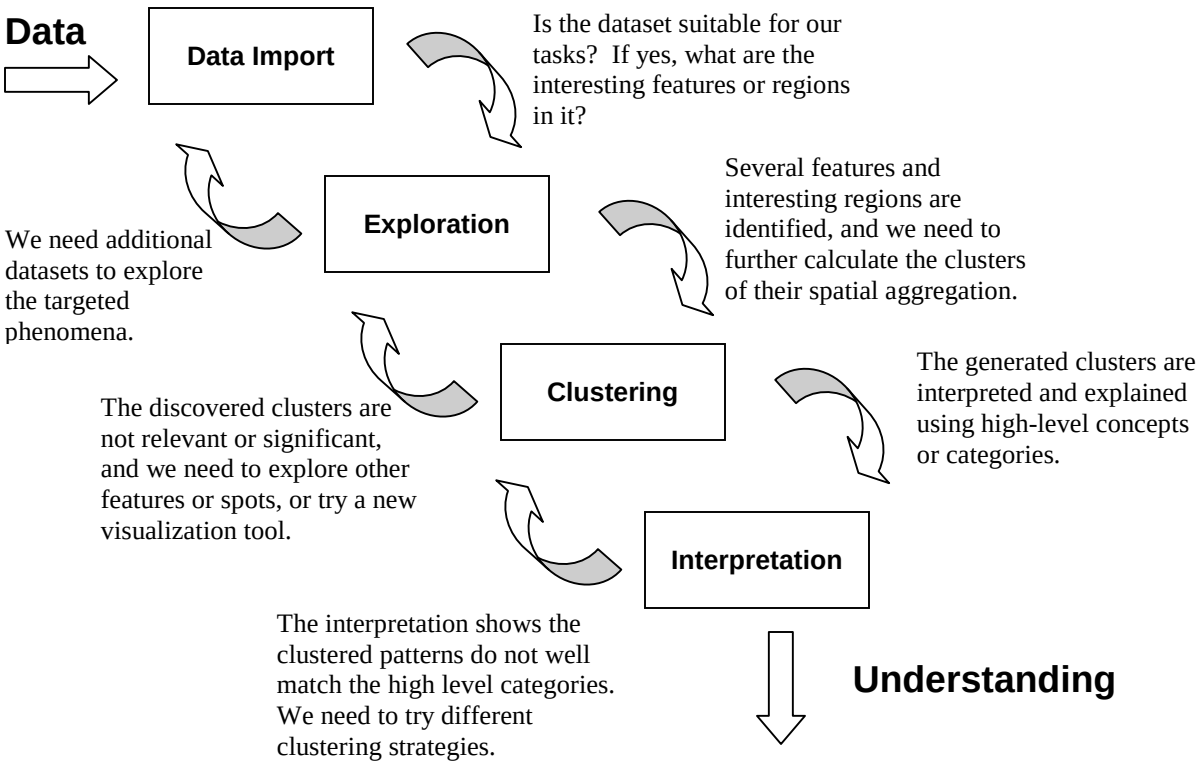


Figure 3.9 An example of iterative geographical problem solving processes, which describe the construction of the geographical knowledge discovery application depicted in Figure 3.1

3.3.3.2 Geographical Problem Solving Methods

The ultimate meanings of geographical applications should always be analyzed case by case because all real world problems are different. Nevertheless, it is possible to identify several general "work paths", problem solving skills, or conceptual strategies that are commonly used in iterative geographical problem solving. We use the term "problem solving methods" to denote such conceptual strategies. In other words, a problem solving method represents a pattern of reasoning that corresponds to a specific

type of mediation context, which can be used to guide the development of automated problem solvers. In this study, three types of problem solving methods are identified for iterative geographical problem solving:

1. Incremental problem solving based on trial-and-error. Incremental problem solving method is required when a human user does not possess any *a priori* insight about the targeted data. This is very common in applications such as ESDA (Exploratory Spatial Data Analysis) and hypothesis generation, where users aim to identify previously unknown patterns and attempt to associate them with untested explanations. Trial-and-error is a basic strategy for incremental problem solving, by which users select an arbitrary tool to perform the task, and evaluate its effectiveness based on the results. For example, in visualization-based ESDA, analysts seldom rely on a single tool to explore data patterns. Instead, different visualization techniques, including color maps, scatter plots, parallel coordinate plots, and tables, among others, are all potentially useful. Since the patterns within a specific dataset are unknown and different visualizations can reveal different aspects of data distribution, users often keep experimenting with different tools until interesting patterns are identified.

2. Solution refinement based on fast prototyping. A "prototype" is a sample application that demonstrates the most typical usage of a certain group of computational resources. In a complex geographical problem solving context, there is usually great uncertainty about the exact strategy to approach the problem, and sometimes even the precise goals of the application are not clear until we can generate a prototypical solution and run it to "see" some preliminary results in the first place. After that, the solution can be refined and adjusted by replacing or re-configuration some of its components. For instance, the application in figure 3.1 can be treated as a prototypical solution to geographical knowledge discovery. In a different geographical knowledge discovery scenario, we can use the same application to quickly obtain some immediate results, and then modify the application structure to fit the new problem settings. For example, we

can replace the PCP with a ScatterPlot component if we suspect that the latter can find patterns in the new dataset that PCP cannot find. Compared with building a new geographical discovery application completely from scratch, refining an existing prototype is much simpler and can dramatically save time and cost.

3. Task decomposition based on divide-and-conquer. The conceptual basis of task decomposition is reductionism, wherein a large problem can be reduced to many smaller sub-problems that are easier to solve. While reductionism is disputable in the general philosophy of geography, it can be used as an effective problem solving method in real world geographical analyses. Task decomposition can be achieved using the divide-and-conquer strategy. For example, the application in figure 3.1 is a complex problem that cannot be directly solved by components from any single traditional domain (or research community). But we can break it down into a "geo-visualization" and a "spatial pattern classification" sub-problem, and try to build sub-applications that perform those tasks as "fragments" of the larger task. We can test each sub-application separately to make sure they are working, and then combine them together to form the final application.

In summary, a meaningful geographical application is usually constructed by iteratively applying one or several problem solving methods listed above. To automate the construction of meaningful geographical applications and improve the productivity of geographical problem solving, a Semantic Geospatial PSE must design and develop automated reasoners to support these problem solving methods.

3.3.4 Summary

A summary of the first, second, and third level semantics in a Geospatial PSE is presented in table 3.2. While semiotics explains and models the meanings of individual computational resources as sign systems, formal methods such as the first order calculus and model theory offer the framework to define the logical structure that can deduce

structural meanings from individual meanings. Geographical problem solving methods describe the types of mediating contexts under which specific patterns of reasoning should be followed. Automated problem solvers can be built based on proxy representations, logical deductions, and geographical problem solving methods, where individual resources are articulated as appropriately designed signs, their relationships are formally defined as logical axioms, and solutions for incremental problem solving, prototype refinement, and task decomposition can be automatically constructed.

Table 3.2 Summaries of the three level semantics.

Semantics	Concepts	Targets	Theories	Purposes
First Level	Individual	Computational Resources	Semiotic Sign Structure	Articulation
Second Level	Relation	Logical Axioms	Formal Model Theory	Discovery & Query
Third Level	Mediation	Geographical Applications	Geographical Problem Solving Methods	Automation & Solution Construction

3.4 Semantic Model of Geographical Problem Solving

This section presents a semantic model of geographical problem solving based on the concepts of proxy representation, the first order predicate calculus, and model theory. The semantic model has three components: 1) The DoD model defines the basic domains of discourse for geographical problem solving. 2) The language model describes the semantic language L_K, where signature K contains a set of non-logical terminology devised for semantic modeling. The semantics of L_K is given model-theoretically as conditional rules for logical statements specified in L_K. 3) The knowledge model focuses on how to use the DoD and language models to capture factual knowledge. The remainder of this section thoroughly describes these three components in their respective order. After that, we discuss how to use the semantic model to resolve the resource articulation and discovery problem. The problem of solution construction, however, will be the topic of chapter four.

3.4.1 The Domain(s) of Geographical Problem Solving

The domain of geographical problem solving is a set of entities that participate in the development of meanings during the application of a Geospatial PSE. According to the concept of proxy representation, four top level DoDs can be identified, specifically: 1) the proxy domain (DoP), which defines the abstract notion of the proxy representation itself; 2) the system domain (DoS), which defines the implementation proxy, or elements originated from system implementation; 3) the runtime domain (DoR), which defines the runtime proxy, or elements required by a specific runtime model such as dataflow; and 4) the user domain, which defines the user proxy, or high-level descriptions. Depending on specific design and implementation choices, these four domains can be further divided into more detailed sub-domains. Figure 3.10 depicts the domain definition adopted in this study, which is based on the notions of software components, the JavaBeans component platform, the dataflow visual programming model, and the high-level abstraction of data and tasks.

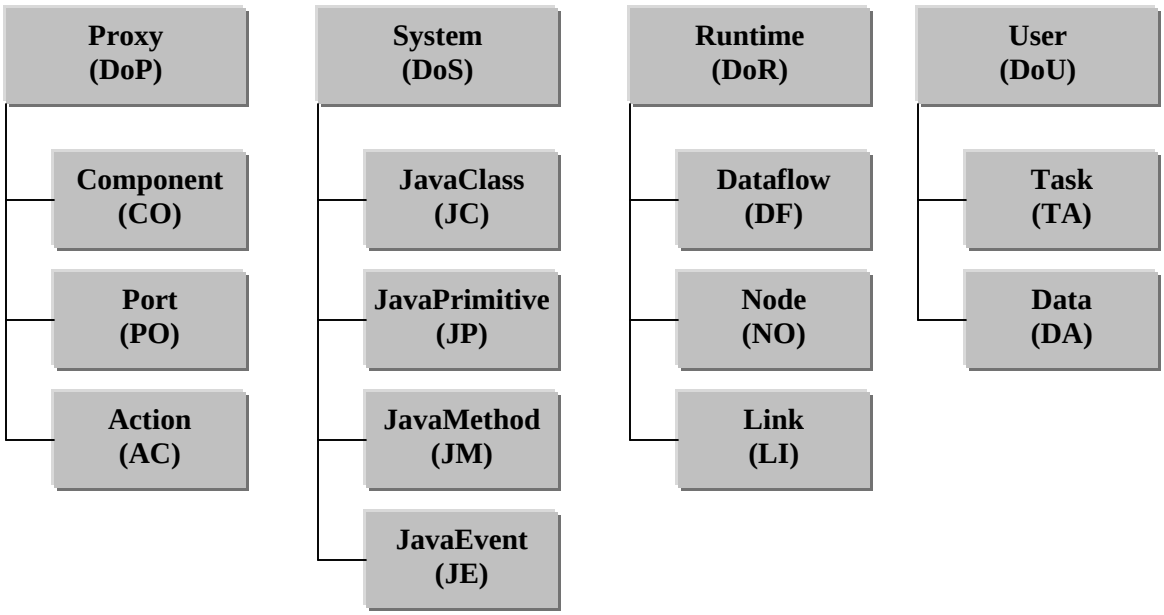


Figure 3.10 The DoDs of geographical problem solving, which is based on the idea of proxy representation, Java platform, and the dataflow runtime model.

3.4.1.1 The Proxy domain

First, the proxy domain in this study is built based on the ideas of software components. It has three sub-domains:

1. The domain of software components (CO), which includes any software units that have been engineered to meet certain component standards. For instance, a JavaBeans component such as GeoMap (and all components mentioned in the data mining example of figure 3.1) belongs to this domain.
2. The domain of data ports (PO), which defines "live" containers of data. "Port" is a term commonly used in component-based software engineering (Szyperski 1998), which is a data holder similar to a variable in conventional computer programming. The content of a port may change when the application is running, and a software component receives input data and broadcasts output data through the correspondent input / output ports.
3. The domain of actions (AC), which defines the atomic behaviors of software components. For instance, the functionality of "displaying a map" performed by the GeoMap component is an action.

3.4.1.2 The System domain

The system domain is always implementation dependent, and different component platforms may have different content or sub-domains for the system domain. As this study uses the JavaBeans component standard as the basic implementation technology, four Java-related sub-domains can be identified:

1. The sub-domain of Java classes (JC);
2. The sub-domain of Java primitives (JP), which contains the primitive data types (i.e. double, integer, float, and etc.) in Java;

3. The sub-domain of Java methods (JM), which includes the identifiers to invoke methods on runtime Java objects. They can be syntactically represented as *method signatures*.
4. The sub-domain of Java events (JE). Java is essentially a discrete event-based system, which relies on events to monitor and notify state change. For example, for the GeoMap component, the input of data, the accomplishment of the mapping functionality, and the output of results are all state changes that are notified by corresponding events.

3.4.1.3 The Runtime domain

The runtime domain contains runtime instances of computational resources, including executable copies of software components and instantiated data parameters. For example, the GeoMap component can be instantiated into multiple runtime instances. Each of them will become an individual information processing unit at runtime, which can take an instance of a geospatial dataset as input parameter and produce a map on the screen. In a dataflow-based visual programming model, the runtime domain includes the following sub-domains:

1. Nodes (NO), which are runtime instances of software components;
2. Links (LI), which defines connections between two nodes;
3. Dataflow networks (DF), which are sets of nodes and edges.

It is noticeable that under such a runtime model, an application (such as the data mining example in figure 3.1) is always a dataflow network, but a dataflow network is not always an application – only dataflow networks that conform to all the semantic constraints defined by the semantic language and bear certain high-level purposes should be regarded as applications.

3.4.1.4 The User domain

The user domain defines user proxies that contain representations built for the purpose of high-level understanding. As mentioned before, the content of the user domain is always subject to the particular user interpretation context. In front-end visual programming, for example, labeled icons can be adopted as effective user proxies. However, to stress the importance of resource articulation under a heterogeneous environment, here we present a strategy that is targeted to ontological development and knowledge sharing over the Semantic Web. It has two sub-domains:

1. The domain of tasks (TA). A task is a high-level abstraction of behavior. Compared with component actions, tasks focus on the conceptual activities rather than computational operations. In this sense the concept of the "mapping" activity is a task, while the operation performed by the GeoMap component is an action.
2. The domain of data (DA), which defines high-level abstractions of datasets. For instance, concepts such as "geospatial dataset", "point dataset", and "polygon dataset" are all belong to the domain of data. They define abstract conceptualizations of data, which do not have to rely on particular data models or data structures.

3.4.1.5 An Example

To illustrate different sub-domains of geographical problem solving, consider the FileChooser and ShapefileDataReader components used in the data mining example from figure 3.1. Basically FileChooser allows a user to select the file path of a dataset and passes the path to ShapefileDataReader, which loads the dataset. Knowledge about the components can be informally described by the following paragraph:

> *The dataflow fragment contains an instance of the FileChooser component, which is implemented by the java class named jh9gpz.io.MtFileChooser, and an*

instance of the ShapefileDataReader component, which is implemented by the java class named edu.psu.geovista.geog.data.ShapefileDataReader. FileChooser interacts with users and outputs a data element representing the selected file path, which is a java.lang.String and can be accessed by the method getAbsolutePath(). After file selection completes, FileChooser fires an action event. ShapefileDataReader takes a file path as the input through the method setFileName(java.lang.String). It produces an ESRI shapefile dataset as the output, which is encoded as a Java object array (the class name is java.lang.Object[]) and can be accessed by the method getDataObjects(). Finally the FileChooser instance is linked to the ShapefileReader instance, as the latter takes the output of the former as its input data.

The entities of different sub-domains from the examples are extracted from the informal description above, and shown in table 3.3.

Table 3.3 Entities from different sub-domains in the example.

Domains	**Sub Domains**	**Informal Descriptions of Domain Elements**
Proxy	*Component*	● The FileChooser component ● The ShapefileDataReader component.
	Port	● The output of FileChooser ● The input of ShapefileDataReader.
	Action	● The specific action performed by FileChooser, ● The specific action performed by ShapefileReader
System	*Java Class*	● java.lang.String; java.lang.Object[]; ● jh9gpz.io.MtFileChooser; ● edu.psu.geovista.geog.data.ShapefileDataReader.
	Java Event	● The action event fired by the FileChooser.
	Java Method	● getAbsolutePath(); ● setFileName(java.lang.String); ● getDataObjects().
Runtime	*Node*	● The instance of FileChooser; ● The instance of ShapefileDataReader.
	Link	● The link between the FileChooser and ShapefileDataReader instances.
	Dataflow	● The application.
User	*Data*	● File Path; ● ESRI Shapefile.
	Task	● Selecting a file; ● Loading ESRI Shapefile dataset.

3.4.2 The Formal Language of Geographical Problem Solving

A formal language of geographical problem solving, denoted by L_K, is defined as a first order language that uses the non-logical terminology defined by the signature *K*. The semantics of L_K is defined in model-theoretic terms. More specifically, we describe the denotations of predicate symbols (which are relations), the denotations of predicate terms (which are domain elements), and the truth conditions of logical statements in terms of set operations.

3.4.2.1 Elements for the Proxy Domain

As described in table 3.4, there are five predicate symbols (or five kinds of structural relations) for the proxy domain. The left column of table 3.4 contains predicate symbols, which is spelled in the format of "name / arity", where "name" is the predicate's syntactical name, and arity defines how many parameters the predicate can take. The right column of table 3.4 defines the model theoretic semantics for each predicate symbol in terms of set operations. Basically the semantics of an *n*-ary predicate symbol is regarded as an *n*-relation between different DoDs, which can be specified as a Cartesian product between sets. One can refer to figure 3.10 for the notations of different DoDs.

Table 3.4 Model theoretic semantics for proxy predicate symbols.

Predicate Symbols	Model Theoretic Semantics
input/2	d(input/2) $\subseteq CO \times PO$
output/2	d(output/2) $\subseteq CO \times PO$
performs/2	d(performs/2) $\subseteq CO \times AC$
consumes/2	d(consumes/2) $\subseteq AC \times PO$
produces/2	d(produces/2) $\subseteq AC \times PO$

According to table 3.4, *input/2* and *output/2* define possible input and output ports for software components. The set of all input and output ports of a component constitute its input and output interfaces respectively. The predicate symbol *performs/2* associates

actions to components that perform them. Finally, *consumes/2* and *produces/2* define the input and output ports used by a particular action. They must be subsets of the input and output interfaces of the component performing the action. The model theoretic semantics of the predicates produced by those five predicate symbols is presented in table 3.5, which specifies conditional constraints for these predicates. In the remainder of this chapter we will continue to use the formats illustrated by table 3.4 and table 3.5 to define the semantics of predicate symbols (as relations) and predicate terms (as conditional constraints).

Table 3.5 Model theoretic semantics for proxy predicates.

Predicates	Model Theoretic Semantics	Informal Descriptions
input(co, po)	d(co) ∈ CO and d(po) ∈ PO	co must be a component and po must be a data port.
output(co, po)	d(co) ∈ CO and d(po) ∈ PO	co must be a component and po must be a data port.
performs(co, ac)	d(co) ∈ CO, d(ac) ∈ AC, and 1. For all po where <d(ac), d(po)> ∈ d(consumes/2), we have <d(co), d(po)> ∈ d(input/2); 2. For all po where <d(ac), d(po)> ∈ d(produces/2), we have <d(co), d(po)> ∈ d(output/2).	co must be a component and ac must be an action. Moreover, all ports consumed by ac must be input ports of co and all ports produced by ac must be output ports of co.
consumes(ac, po)	d(ac) ∈ AC, d(po) ∈ PO, and there exists an component co such that <d(co), d(ac)> ∈ d(performs/2) and <d(co), d(po)> ∈ d(input/2).	ac must be an action and po must be a data port. Moreover, po must be an input port of the action that performs ac.
produces(ac, po)	d(ac) ∈ CA, d(po) ∈ PO, and there exists an component co such that <d(co), d(ac)> ∈ d(performs/2) and <d(co), d(po)> ∈ d(output/2).	ac must be an action and po must be a data port. Moreover, po must be an output port of the action that performs ac.

3.4.2.2 Elements for the System Domain

A JavaBeans-based system domain has two kinds of predicate symbols. The first kind includes *extends/2*, *convertible/2*, and *assignable/2*, which define relationships between Java data types such as classes and primitives. More specifically, *extends/2* specifies sub-classing between Java classes and *convertible/2* declares data conversion between Java primitives (e.g. whether a float data element should be converted into an integer data element). The predicate symbol *assignable/2* defines whether the data from one data type can be assigned to another data type. The truth condition of *assignable/2* is completely decidable by the sub-classing and primitive conversion rules of the Java platform. The second kind predicate symbols are used to specify the relationships between Java elements and proxy representations. The predicate symbol *javatype/2* defines the Java type of a data port, which could be either Java classes or primitives. In addition, *implementedBy/2* defines the Java class that implements a component, *notifiedBy/2* defines the Java event that notifies an action, and finally, *setter/2* and *getter/2* declare Java methods to access input and output data ports. The model theoretic semantics of these predicate symbols are described in table 3.6.

Table 3.6 Model theoretic semantics for system predicate symbols.

Predicate Symbols	Model Theoretic Semantics
extends/2	d(extends/2) $\subseteq JC \times JC$
convertible/2	d(convertible/2) $\subseteq JP \times JP$
assignable/2	d(assignable/2) $\subseteq PO \times PO$
javatype/2	d(javatype/2) $\subseteq PO \times (JC \cup JP)$
implementedBy/2	d(implementedBy/2) $\subseteq CO \times JC$
notifiedBy/2	d(notifiedBy/2) $\subseteq AC \times JE$
setter/2	d(setter/2) $\subseteq PO \times JM$
getter/2	d(getter/2) $\subseteq PO \times JM$

Furthermore, the truth conditions of predicate terms produced by these predicate symbols are listed and explained in table 3.7 based on the requirements of Java. Notice that table 3.7 only includes a minimal set of semantics that is useful in terms of logical reasoning. There are a number of low-level consistency and inference rules maintained automatically by the underlying Java platform, e.g. the definition of sub-classing and the validity of a Java method signature. As enforcing the correctness for such kind of low-level knowledge is not the main interest of this study, we assume all Java classes, primitives, methods, and events declared in the semantic language are valid Java identifiers.

Table 3.7 Model theoretic semantics for system predicates.

Predicates	Model Theoretic Semantics	Informal Descriptions
extends(jc1, jc2)	d(jc1) ∈ JC and d(jc2) ∈ JC	Both jc1 and jc2 must be Java classes.
convertible(jp1, jp2)	d(jp1) ∈ JP and d(jp2) ∈ JP	Both jc1 and jc2 must be Java primitives.
assignable(dt1, dt2)	dt1 = dt2; or d(dt1) ∈ JC and d(dt2) ∈ JC and <d(dt1), d(dt2)> ∈ d(extends/2); or d(dt1) ∈ JP and d(dt2) ∈ JP and <d(dt1), d(dt2)> ∈ d(convertible/2).	dt1 equals to dt2; or both dt1 and dt2 are Java classes and dt1 is a sub-class of dt2; or both dt1 and dt2 are Java primitives and dt1 can be converted into dt2.
javatype(po, jt)	d(po) ∈ PO and d(jt) ∈ $(JC \cup JP)$.	po must be a data port and jt must be either a Java class or a Java primitive.
implementedBy(co, jc)	d(co) ∈ CO and d(jc) ∈ JC.	co must be component and jc must be a Java class.
notifiedBy(ac, je)	d(ac) ∈ CA, d(je) ∈ JE.	ac must be a action and je must be a Java event.
setter(po, jm)	d(po) ∈ PO, d(jm) ∈ JM.	po must be a data port and jm must be a Java method.
getter(po, jm)	d(po) ∈ DE, d(jm) ∈ JM.	po must be a data port and jm must be a Java method.

3.4.2.3 Elements for the Runtime Domain

Table 3.8 Model theoretic semantics for runtime predicate symbols.

Predicate Symbols	Model Theoretic Semantics
hasNode/2	d(hasNode/2) ⊆ $DF \times NO$
hasLink/2	d(hasLink/2) ⊆ $DF \times LI$
instanceOf/2	d(instanceOf/2) ⊆ $NO \times CO$
connects/3	d(connects/3) ⊆ $LI \times NO \times NO$
channel/3	d(channel/3) ⊆ $LI \times PO \times PO$

Table 3.9 Model theoretic semantics for runtime predicates

Predicates	Model Theoretic Semantics	Informal Descriptions
hasNode(df, no)	d(df) ∈ DF and d(no) ∈ NO.	df must be a dataflow network and no must be a node.
hasLink(df, li)	d(df) ∈ DF and d(li) ∈ LI.	df must be a dataflow network and li must be a link.
instanceOf(no, co)	d(no) ∈ NO and d(co) ∈ CO.	no must be a node and co must be a component.
links(li, no1, no2)	d(li) ∈ LI, d(no1) ∈ NO, d(no2) ∈ NO.	li must be a link, and both no1 and no2 must be nodes.
channel(li, po1, po2)	d(li) ∈ LI, d(po1) ∈ PO,, d(po2) ∈ PO and there are dt1, dt2, no1, no2, co1, co2 such that 1. <d(po1), d(dt1)> ∈ d(javatype/2), <d(po2), d(dt2)> ∈ d(javatype/2), and <d(dt1), d(dt2)> ∈ d(assignable/2); 2. <li, no1, no2> ∈ d(links/3), <no1, co1> ∈ d(instanceOf/2), <no2, co2> ∈ d(instanceOf/2), <co1, po1> ∈ d(input/2), <co2, po2> ∈ d(output/2),	li must be a link and po1, po2 must be data ports that satisfy the following conditions: 1. the java data type of po1 must be assignable to the java data type of po2; 2. po1 must be an output port of the component that instantiates the source node and po2 must be an input port of the component that instantiates the target node.

The runtime sub-language is comprised of five predicate symbols. The predicate symbol *hasNode/2* and *hasLink/2* define the nodes and links belonging to a dataflow network. Furthermore, *instanceOf/2* declares the component that instantiates a node, and *link/3* defines an individual link by specifying the node it departs from and points to. Finally *channel/3* declares a data communication channel for a link, which binds an output data port from the source node component with an input data port to the target node component. During system execution, the data coming out of the output port will be sent to the input port. The model theoretical semantics of the predicate symbols and terms are listed in table 3.8 and table 3.9 respectively.

3.4.2.4 Elements for the User Domain

As mentioned before, the intention of this study is not to develop a specific form of user representation, but to build the connections between proxy representations and other high-level knowledge representations selected by the user, for instance, ontologies of data models and tasks. Consequently, we only define the necessary elements to connect proxy representations with concepts of data and tasks. Such concepts and tasks may be developed by third-party sources other than the creator of the proxy representations. Two predicate symbols are defined for the user domain, including *role/2,* which associates a data port with a data concept, and *accomplishes/2,* which associates an action with a task concept. The model theoretic semantics for the predicate symbols and terms are given in table 3.10 and table 3.11 respectively.

Table 3.10 Model theoretic semantics for user predicate symbols.

Predicate Symbols	Model Theoretic Semantics
role/2	d(role/2) $\subseteq PO \times DA$
accomplishes/2	d(accomplishes/2) $\subseteq AC \times TA$

Table 3.11 Model theoretic semantics for user predicates.

Predicates	Model Theoretical Semantics	Informal Descriptions
role(po, da)	d(po) ∈ PO and d(da) ∈ DA.	po must be a data port and da must be a data concept.
accomplishes(ac, ta)	d(ac) ∈ AC and d(ta) ∈ TA.	ac must be an action and ta must be a task concept.

3.4.3 Knowledge Models of Graphical Problem Solving

The semantics of the computational resources in a Geospatial PSE can now be encoded as logical axioms using the language described above. Conveniently, we refer to knowledge specified in the proxy sub-language as proxy semantics, denoted by K_P; knowledge specified in the system sub-language as system semantics, denoted by K_S; knowledge specified in the runtime sub-language as runtime semantics, denoted by K_R; and knowledge specified in the user sub-language as user semantics, denoted by K_U. In a given Geospatial PSE, K_P, K_S, K_R, and K_U must satisfy the following conditions:

1. $K_S \vDash K_P$, K_R, and K_U, which means that system semantics is semantically necessary for proxy, runtime, and user semantics, since it is impossible to change the implementation structures of computational resources on a given PSE at the time of application;
2. $K_U \vDash K_P$, which means that user semantics is semantically necessary for proxy semantics, because the semantic representation must conform to the data and task specification selected by the user.
3. $K_P \vDash K_R$ which means that proxy semantics is semantically necessary for runtime semantics, as the latter must be instantiated by the former (e.g. a node in a dataflow must be instantiated by a component).

Figure 3.11 can help readers better understand the semantic dependencies between system, user, runtime, and proxy semantics. From a knowledge engineering point of view, one can regard them as a special kind of typing system. For instance, a

node in a runtime dataflow model must be typed as a component, which should be further typed as a component implementation (e.g. a Java class) and a high-level user objects (icons or tasks).

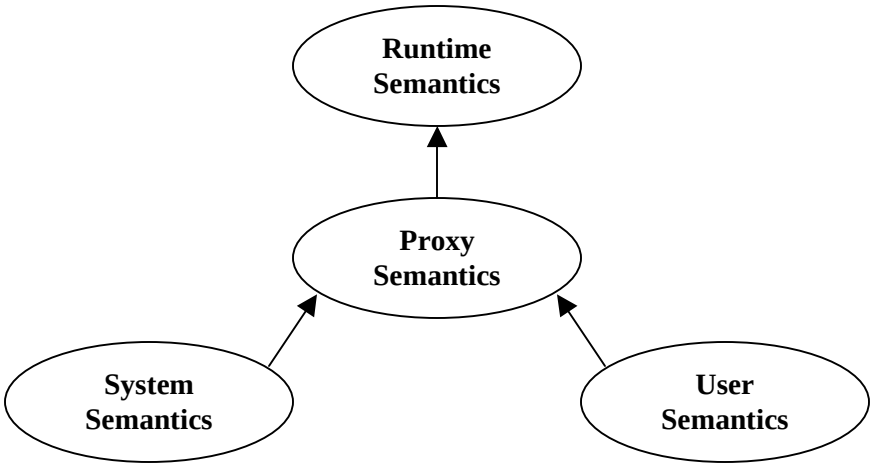


Figure 3.11 The semantic dependencies between system, user, runtime, and proxy semantics.

However, it should be noted that semantic dependencies should not be mistakenly understood as representational dependencies. A proxy representation semantically depending on a system and user proxies does not mean the proxy representation cannot be shared across different system platforms or user contexts. It only means the *validity* of the proxy semantics (i.e. the truth values of proxy predicates) is subject to the selected system and user semantics. In other words, while the same proxy semantics (e.g. the abstract notion of a "map" component) can be shared across different systems and user contexts, its truth conditions will also change accordingly. This is in fact an important rule for semantic knowledge sharing, because it can be used to detect whether a proxy representation can retain the same validity across different knowledge sharing parties, and find the necessary conditions or evidences in each party that entails such shared meanings

The subsequent paragraphs present a sample knowledge model using the simple scenario of shapefile reading described in 3.4.1.5. The order of modeling follows the semantic dependencies described above, which is system => user => proxy => runtime.

Table 3.12 lists the predicates representing the system semantics for the components of ShapefileDataReader and FileChooser. Here fc_action denotes the action of FileChooser, which selects a data file, while fc_out and sr_in denoting the output data port of FileChooser and input data port of ShapefileDataReader respectively.

Table 3.12 Predicates representing the system semantics.

Predicates	Explanations
javatype(fc_out, java.lang.String)	The data port fc_in's Java data type.
javatype(sr_in, java.lang.String)	The data port sr_in"s Java data type.
getter(fc_out, getAbsolutePath())	The data port fc_out's getter method.
setter(sr_in, setFileName(java.lang.String))	The data port sr_in's setter method.
notifiedBy(fc_action, action)	The action fc_action's notification event.
implementedBy(ShapefileDataReader, edu.psu.geovista.geog.data.ShapefileDataReader)	The component ShapefileDataReader's implementing class.
implementedBy(FileChooser, jh9gpz.io.MtFileChooser)	The component FileChooser's implementing class.

The next step is to define the user semantics. Suppose there is a task named "SelectFile", which signifies the concept of "select the file path to the dataset", and a data concept named "FilePath", which signifies the concept of file path. The corresponding user semantics is specified in table 3.13, where the action "fc_action" accomplishes the "SelectData" task and both data ports "fc_out" and "sr_in" take file paths as parameters.

Table 3.13 Predicates representing the user semantics.

Predicates	Explanations
accomplishes(fc_action, LocateData)	The action fc_action accomplishes the high-level task of LocateData.
role(fc_out, FilePath)	The role of fc_out is FilePath.
role(sr_in, FilePath)	The role of fc_in is FilePath.

Then we can define the proxy semantics for these two components in table 3.14.

Table 3.14 Predicates representing the proxy semantics.

Predicates	Explanations
output(FileChooser, fc_out)	fc_out is an output port of FileChooser;
input(ShapefileDataReader, sr_in)	sr_in is an input port of ShapefileDataReader;
performs(FileChooser, fc_action)	FileChooser performs fc_action;
produces(fc_action, fc_out)	fc_action produces new data for the output port fc_out.

Finally, the fact that an instance of FileChooser is connected to an instance of ShapefileDataReader can be specified via the following runtime semantics (table 3.15):

Table 3.15 Predicates representing the runtime semantics.

Predicates	Explanations
instanceOf(fc_node, FileChooser)	The node fc_node is instantiated by the component FileChooser.
instanceOf(sr_node, ShapefileDataReader)	The node sr_node is instantiated by the component ShapefileDataReader.
link(li, fc_node, sr_node)	The link li connects the nodes fc_node and sr_node.
channel(li, fc_out, sr_in)	A data channel of li0 binds the data port fc_out to the data port sr_in.

In table 3.15, a data channel is established to bind the data ports fc_out and sr_in, and since the fc_out's Java data type is assignable to sr_in's (because they are both

java.lang.String), and both data ports share the same role “FilePath”, the data channel is a valid one. The entire semantics of this connection are informally described as the follows: every time a user selects a new file using the instance of FileChooser, a Java event is fired to notify the completion of the action fc_aciton. Upon the event, the system execution platform retrieves the data value from fc_out using its getter method, and assigns it to sr_in via its setter method.

3.4.4 Resource Articulation and Discovery

Due to the advantages of proxy representation, the proposed semantic model provides a flexible support for resource articulation. With appropriately specified system, user, and runtime semantics, human analysts can interact with high-level descriptions of data and tasks (which have been encoded as user proxies), and the semantic knowledgebase will automatically link these descriptions back to system resources and runtime objects based on the proxy representations, the conditional rules, and logical deduction without any additional work.

For resource discovery, we can compose logical queries in the problem solving language L_P and the let the system knowledgebase find the semantic elements that satisfy the queries.

Table 3.16 Query patterns for concept-based resource discovery.

Query Patterns	Explanations
accomplishes(?action, ta) ∧ performs(?co, ?action)	Query unknown actions according to the task they accomplish, where ta is a known task concept.
role(?port, da) ∧ input(?co, ?port)	Query unknown components according to the roles of their input data ports, where da is a known data concept.
role(?port, da) ∧ output(?co, ?port)	Query unknown components according to the roles of their output data ports, where da is a known data concept.

1. Concept-based resource discovery. Table 3.16 lists the logical query patterns that may be used to support concept-based resource discovery, which can find relevant resources based on known task or data concepts. For example, suppose we have a knowledgebase that contains the semantics described in the example of section 3.4.3. The following query will retrieve all components that can accomplish the "SelectFile" task:

accomplishes(?action, SelecteFile) ∧ performs(?co, ?action)

The retrieved information will be stored in a variable binding:

[?action/fc_action, ?co/FileChooser].

This means that "FileChooser" is the desired component and "fc_action" is the correspondent action. Similar procedures can be followed to discover components using other query patterns.

2. Context-based resource discovery. Under the proposed semantic model, a context-based resource discovery problem can be transformed into a problem or several problems of concept-based resource discovery. For example, suppose we have an instance of the FileChooser component in an application context, and we want to discover all components that can be connected with the output port of this existing instance. This equates to finding all components that have an input port with a Java data type assignable from the output port of FileChooser. Both data ports also share the same data role. This is essentially a concept-based resource discovery problem that can be solved using the query pattern listed in the second row of table 3.16.

3.5 Summary

This chapter describes three levels of semantics in a Geospatial PSE, including the first level semantics about individual resources, the second level semantics regarding relations, and the third level semantics related to applications. The main semantic problems that handicap geographical problem solving in an open, heterogeneous

environment include the lack of effective articulation for individual resources, the absence of resource discovery methods, and the difficulties of manual solution construction, which correspond to the first, second, and third semantic levels respectively. A Semantic Geospatial PSE is a system that provides solutions to all three problems. Based on triadic sign model, formal model theory, and geographical problem solving methods, a semantic model of geographical problem solving is presented to guide the design and implementations of Semantic Geospatial PSEs. The semantic model regards individual computational resources as signs interpreted under different application contexts and consequently use proxy representations to aggregate different representational forms of computational resources, including system proxies that encode implementation details, user proxies that signify high-level understandings, and runtime proxies that represent runtime roles. The structural relationships among different elements of proxy representations are then formalized as logical axioms based on first order predicate calculus and model theory, which can be validated and queried using logical theorem proving. The development of automated techniques for solution construction will be the focus of the next chapter.

Chapter 4

AUTOMATED METHODS OF GEOGRAPHICAL PROBLEM SOLVING

4.1 Introduction

The chapter aims to solve the problems related to the third level semantics, i.e. how to help users convert clearly articulated and logically structured computational resources into meaningful geographical applications. As mentioned in our previous discussions, three types of problem solving methods are particularly important to an iterative geographical problem solving process, including incremental problem solving based on trial-and-error, solution refinement based on fast prototyping, and task decomposition based on divide-and-conquer. This chapter draws upon relevant theories and useful techniques from cognitive science and AI to build tools that deliver automated solutions to application contexts defined by these three problem solving methods. The automated tools should be able to search for alternative tools required by incremental problem solving, refine partial solutions provided by existing prototypes, and build complex applications based on task decomposition.

Six sections are included in this chapter. After the introduction, section 4.2 discusses the cognitive model of problem solving, analyzes the mental structure of problem space, and examines how to computationally implement (or simulate) the reasoning procedure of human problem solving. Section 4.3 describes the methods of partial order planning, which can be used to automate incremental problem solving and prototype refinement. Following that, section 4.4 discusses the ideas of hierarchical task networks, which provide natural supports for task decomposition. Section 4.5 then combines the techniques of partial order planning and hierarchical task network planning

to create a framework of automation for the semantic model described in chapter three. Finally, section 4.6 provides summaries.

4.2 Cognitive Model of Human Problem Solving

Cognitive models of human problem solving provide the basic guidelines for developing automated problem solvers. Human problem solving was first systematically studied by Gestalt [1] psychologists in order to understand the relationships between environment, perception, and human reasoning (Maier 1931; Duncker 1935; Duncker 1945). According to Gestalt studies, a "problem" is a situation where "a living organism has a goal but does not know how this goal is to be reached" (Duncker 1945). Consequently, problem solving is a *mental reasoning* process performed by the problem solving agent(s) in order to find the necessary action steps to accomplish the goal (thus "solve" the problem) (Newell, Shaw et al. 1958). In the broader picture of human cognition, problem solving represents one step of human-environment interaction, and the entire psychological process can be summarized as "perception, problem solving, and execution", during which human problem solvers observe the environment, solve the targeted problem based on perceived information, and execute the solution plan to impact the environment. While Gestalt psychology was largely focused on the role of perceptual forms in human problem solving, since the 1950s, scientists have started to examine the internal mental structures of human agents and study how such structures can be utilized to model the reasoning processes of problem solving (von Eckardt 1993). Under the banner of a newly emerged field known as cognitive science, this strand of studies draws upon multiple inputs from philosophy, psychology, AI, and other related fields to explain human behaviors. Generally speaking, cognitive science is heavily influenced by the functionalist view of the mind, which believes human thinking is decided by a set of

[1] "Gestalt" is a term coined by German psychologist Christian von Ehrenfels to refer to the "whole form" or the perceptual structures that decide how objects are perceived by human agents (Ehrenfels 1890).

mental states that control the interactions between mental structures, sensory inputs, and behavioral outputs (Putnam 1967; Armstrong 1968). While such a view is not widely accepted in philosophy, it provides a conceptual ground to mechanically model cognitive processes and computationally implement automated AI problem solvers. Cognitive models of human problem solving, therefore, constitute the theoretical foundation for automated geographical problem solving tools.

4.2.1 Information Processing Theory of Human Problem Solving

One of the most influential cognitive models of human problem solving was proposed by Newell and Simon (1972), which represents the mental structure of problem solving in terms of states and state transitions (Newell, Shaw et al. 1958; Newell and Simon 1972). Known as the information processing theory of human problem solving, Newell and Simon's model identifies two basic elements of human reasoning: 1) a world of mental states, which models the state of affairs residing in the cognitive agent's mind; 2) and a mechanism of state change, which defines how the state of affairs described by the mental states may evolve through time. A problem then can be formulated as a situation comprised of three elements:

1. **Problem Givens:** a set of mental states that describe the *current* state of affairs in the problem solver's mind. Problem givens are essentially conditions that are regarded by the problem solver as satisfied *right now*.
2. **Problem Goals**: a set of mental states that describe the *desired* state of affairs expected by the problem solver. Problem goals are currently unsatisfied conditions that are desired to be satisfied in the future.
3. **State Transition Operators:** a set of primitive transitions between different mental states. A state transition operator is defined by a set of input states and a set of output states. It is also commonplace to call a state transition operator

an “action operator” (or operator in short), with its input states as the “precondition”, and its output states as the “effects”.

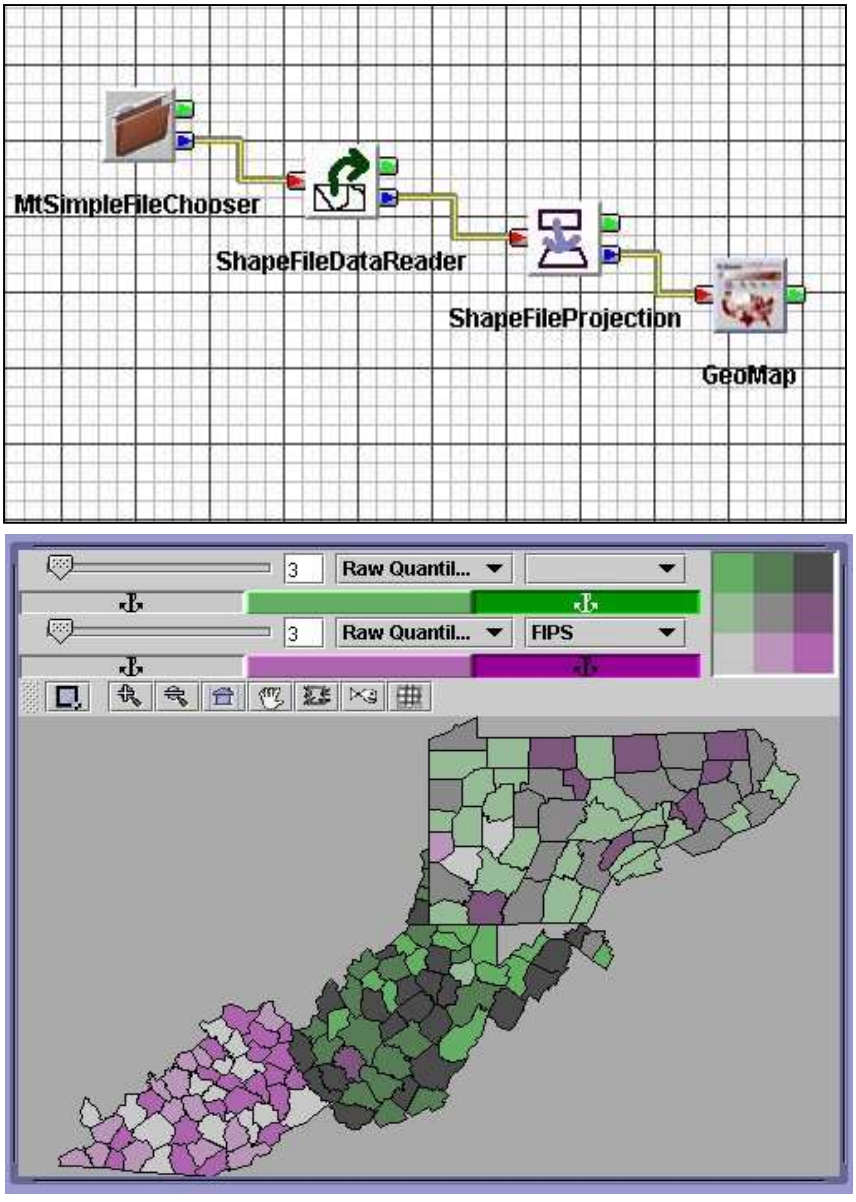


Figure 4.1 A simple map visualization example in GeoVISTA *Studio*.

A problem, therefore, only occurs when the problem givens cannot satisfy problem goals, i.e. when some desired state of affairs is *not* satisfied by the problem givens. Solving a problem is equivalent to finding an *agenda of state change* that causes

the current mental state to evolve towards meeting the unsatisfied goals (i.e. from givens to goals). Such an agenda constitutes a *solution*.

For example, suppose we are trying to visualize a geospatial map of the Appalachian region. The state of resource availability, including the presence of data sources and mapping tools, can be considered as problem givens. The state that "a geospatial map of the Appalachian region is in hand", is the problem goal, since the map is currently not available and we need to find a way to produce it. A solution then is a set of ordered activities, for instance, selecting the data file from the disk, importing the dataset into computer memory, and mapping it on the screen. Each of the activities introduces newly satisfied mental states (e.g. a file is newly selected), and applying all activities in the proper order will eventually result in the mental states that meet the problem goal, i.e. the presence of the desired map.

Figure 4.1 describes a solution to the above map visualization problem in GeoVISTA *Studio*, where four components are used: MtSimpleFileChooser that help users select a file from disk, ShapeFileDataReader that loads the dataset from the selected file, ShapeFileProjection that builds an appropriate map projection, and GeoMap that finally visualizes the data. Each component performs an action that takes some inputs and produces some outputs. If we represent the inputs and outputs as mental states, then these components can be modeled as state transition operators, which are listed in table 4.1 (note that operators are named differently from components). The problem solving agenda, or the state transition "path", is embedded in the dataflow diagrams, where the state transition performed by each component is appropriately connected and sequenced to produce a geospatial map.

The problem representation based on state transition is also known as a *problem space* in Newell and Simon's theory, and a problem solution can be found by analyzing a well-represented problem space. It is noticeable that a problem space can be much more complex than the simple example described above, as problem givens and goals can

contain more than one mental state, and a state transition operator may have multiple input and output states. However, the basic idea of state transition remains the same.

Table 4.1 State transition operators for the map visualization problem.

State / Operator	Input State	Output State
SelectFile	S0: Null state.	S1: A file path is present.
ReadShapefile	S1: A file path is present.	S2: A shapefile dataset is present.
ProjectShapes	S2: A shapefile dataset is present.	S3: Projected geometries are present.
VisualizeMap	S3: Projected Geometries are present.	S4: On-screen map is present.

Newell and Simon's model has been criticized from various perspectives, mainly for its over-simplified representation of problem space (Bhaskar and Simon 1977), its pre-assumed symbolic model of human reasoning (Searle 1980), and its inadequacy in solving complex problems (Sternberg 1995). One of the main difficulties of the model in analyzing real world problem solving is the presence of ill-defined (or ill structured) problems (Reitman 1965). Well-defined problems are problems represented by clearly identified initial states, goal states, and state transition operators. On the other hand, a problem is ill defined if some information regarding the problem space is missing. A big assumption of Newell and Simon's theory is that all relevant information about mental states and state transitions can be perceived by problem solvers and represented in problem spaces at the time the model is built. However, in real world applications this is seldom the case. In many situations only limited information about the problem space is known at modeling time, and sometimes more information can be gradually obtained during the time of problem solving. For example, a GIS novice may have little clue about how to load an ESRI shapefile dataset into ArcMap, but after interacting with the

software for a while, he / she may eventually figure out what to do. Unfortunately, Newell and Simon's theory cannot support these kinds of situations.

Despite its deficiencies, Newell and Simon's model has received wide recognition in computational sciences, First, the theoretic foundation of the model is discrete state transition, which is a natural fit for modern digital computing. The state transition model can be easily implemented on digital computers, as exemplified by GPS (General Problem Solver), an automated AI program developed by Newell and Simon. Second, the drawback of Newell and Simon's model is mainly due to the inability to create well-defined problems in real world problem domains, which is less a problem when modeling the behaviors of computational systems. Compared with real world human behaviors, computational resources are always built upon certain digital structures based on state transition (e.g. data structures and software modules). Although building an explicit problem space for computational resources is a non-trivial task, it is generally much easier than constructing a problem space for a real world domain. Therefore, this study adopts the ideas of problem space and problem space analysis as the conceptual and theoretical ground for automated geographical problem solving.

4.2.2 Automated Methods of Problem Solving

After describing how to represent a problem, the next question is how to develop automated methods to solve different problems. From previous discussions, it is evident that under a state-transition problem space, solution generation is essentially a search process that evaluates all possible combinations of state transition operators to find the right "path" that connects problem givens to goals. Generally speaking, there are two contrasting approaches to this issue:

1. The logical approach, which regards the search process as one kind of logical theorem proving;

2. The epistemological approach, which employs non-logical methods such as procedural algorithms to find solutions.

4.2.2.1 The Logical Approach

The logical approach is intellectually intriguing because by formalizing state changes as logical axioms, we can directly exploit the deductive inference system provided by logic to solve a problem. Methods of logical problem solving can be found in a number of non-classical formalisms such as linear logic (Girard 1987), situation calculus (Levesque, Pirri et al. 1998), and different versions of temporal logics (Emerson 1990). However, one critical weakness of most logical methods is the *closed world assumption* underlying them, which requires all axioms about state transition operators to be explicitly listed because they do not support logical reasoning between statements in preconditions and effects. This is an unrealistic assumption in real world knowledge modeling, as the precondition and effects of an operator may be deductively dependent on conditions defined in third-party knowledge and / or ontologies, which may vary in different application contexts. More specifically, any logical problem solving method based on the closed world assumption is vulnerable to the following two problems (Gelfond and Lifschitz 1993):

1. The problem of *qualification*, which refers to the inability to list all indirect preconditions of an operator;
2. The problem of *ramification*, which refers to the inability to list all indirect effects of an operator.

For instance, in the map visualization example described above, the state transition operator ProjectShapes requires a shapefile dataset as its input. Now if we have a polygon shapefile dataset, can it be regarded as a valid input for ProjectShapes? In a closed-world logical system, if we translate everything from table 4.1 into logical axioms, the answer will be “no”, because those axioms themselves do not state that a

"polygon shapefile dataset" is a specialized case of "shapefile dataset". In other words the logical system cannot tell this until we explicitly specify it. If we want the logical system to recognize a polygon shapefile dataset, we must explicitly list the relevant local statement in the precondition of the operator. However, in an open heterogeneous environment, relationships such as specialization are usually defined in third-party knowledge sources (e.g. in common ontologies or meta-data of a geospatial dataset), and the fact that a "polygon shapefile dataset is also a shapefile dataset" is obtained indirectly by deducing from those knowledge sources. Since it is impossible for the designer of operators to consider all third-party knowledge sources in advance, closed-world logical axioms cannot include every case where an operator is qualified (i.e. its precondition is satisfied). Even if all third-party knowledge sources are known in advance, the qualification problem still remains because we must list all premises that can deduce the precondition and premises of premises recursively until all possibilities of deduction have been considered. In our example, this means to declare "a polygon shapefile dataset is present" as a disjunction with "a shapefile dataset is present" in the precondition of the ProjectShapes operator. Such an approach is not only redundant, but also unwarranted due to the possibility of unbounded premises, as there could be an unlimited number of premises that can deduce the precondition of an operator. Similar difficulties can be observed for the effects of an action operator; in this case, it is impossible to list all indirect effects resulted from deduction. As the main goal of a Geospatial PSE is to integrate different resources with heterogeneously defined meanings, the logical approach obviously cannot support automated geographical problem solving adequately.

4.2.2.2 The Epistemological Approach

The epistemological approach refers to all methods that adopt non-logical and usually algorithmic techniques to reason with state transition operators. Most non-logical problem solving methods have been developed in the automated planning community, a

sub-field of AI that has seen rapid development since the 1960s. It is noticeable that the epistemological approach is not "logic-free": in most cases, mental states, preconditions, and effects are still defined as logical statements. However, state transition operators are usually defined not as logical axioms, but in certain formal data structures that play no role in logical deduction. An algorithm is then designed to mechanically evaluate the precondition and effects of individual operators and combine them into a solution. If we regard a problem space of state transitions as a mental world, then the logical approach attempts to specify state transitions as the laws or rules of the world (thus they are part of the mental world), while the epistemological approach seeks means outside and independent of the world to reason with state transitions (this is where the term "epistemological" comes from). Therefore, the epistemological approach in fact bypasses the problems of qualification and ramification rather than solving them.

The classical definition of a problem space in automated planning is known as the STRIPS (STanford Research Institute Problem Solver) domain, which can be described by the 3-tuple <*A*,*G*,*G'*> (Fikes and Nilsson 1971). Here *A* is a set of action operators, *G* is a set of logical statements representing the problem givens, and *G'* is a set of logical statements specifying the problem goals. In the original proposal of STRIPS planning domain, an action operator has three lists of logical statements: a list of preconditions, which includes all logical statements that must be satisfied for the operator to be applicable; an "add" list, which contains logical statements that becomes true as a result of the operator; and a "delete" list, which defines logical statements that turn false after the execution of the operator. Today it is a common practice to merge the "add" and "delete" lists into one list of effects, where the statements in the delete list are represented as negations (Ghallab, Nau et al. 2004). Consequently, an action operator is a 3-tuple *(name, precond, effects)*, where *name* denotes the syntactical identifier of the action operator, *precond* is a set of logical statements representing the preconditions of the action operator, and *effects* is a set of logical statements denoting the effects of the action

operator. On the other hand, while theoretically one can include any logical formulae in a STRIPS action operator, most existing automated planning methods only allow literals (predicates or negation of predicates) in order to improve the performance of planning algorithms (Ghallab, Nau et al. 2004).

Table 4.2 STRIPS operators for the map visualization problem.

State / Operator	Precondition	Effects
SelectFile	Empty	role(?s1, FilePath) javatype(?s1, java.lang.String)
ReadShapefile	role(?r0, FilePath) javatype(?r0, java.lang.String)	role(?r1, Shapefile) javatype(?r1, java.lang.Object[])
ProjectShapes	role(?p0, Shapefile) javatype(?p0, java.lang.Object[])	role(?p1, ProjectedShapefile) javatype(?p1, java.lang.Object[])
VisualizeMap	role(?m0, Shapefile) javatype(?m0, java.lang.Object[])	role(?m1, OnScreenMap)

Table 4.2 lists the STRIPS operators for the map visualization problem mentioned before. The input and output states from table 4.1 are translated into predicates using the semantic model described in chapter three, where the predicate symbol *role/2* defines the conceptual role of data ports and *javatype/2* specifies their Java data types. Elements such as "FilePath" and "Shapefile" are high-level concepts, which are constants in the predicate. A data port is encoded as a formal variable, as its content may change during reasoning. The automated problem solver can then use variable substitutions to find potential "matches" between data ports. For instance, ReadShapefile has an input data port denoted by the variable "*?r0*", which is a file path (signified by the concept "FilePath) represented as a Java String (signified by the Java class name "java.lang.String"). Since the operator SelectFile has an output data port "*?s1*" that has the same role and Java data type, an automated planner can determine that ReadShapefile can be executed after SelectFile, because ReadShapefile's precondition will be fully satisfied after the SelectFile's effects have been asserted.

Givens and goals can all be encoded as logical statements. In this example, problem givens is empty and the problem goal can be expressed as the single predicate "*role(?x, OnScreenMap)*" to indicate the output of an on-screen map. The algorithmic procedure of finding a solution is one that evaluates all action operators and finds an structure of connected operators, where the preconditions of all selected operators are satisfied at the time point they should be executed, and their effects eventually satisfy the problem goals. Intuitively, one can easily find the following list to be a valid solution:

(SelectFile, ReadShapefile, ProjectShapes, VisualizeMap)

This equals to the component composition displayed in figure 4.1, where the functionality of each component can be executed in the linear order to produce the desired geospatial map.

Brute-force search for problem solutions in a STRIPS-like problem domain is a non-trivial and computationally expensive procedure, where the complexity is exponential with the size of the problem space (i.e. the number of operators and mental states to evaluate) (Zhang 1999). As a result, one of the major research themes of automated planning is to devise efficient algorithms that can find valid solutions at a reasonable cost. For a Geospatial PSE, such algorithms also need to support the modes of reasoning commonly used in geographical analyses, including those related to incremental problem solving, prototype refinement, and task decomposition. The next two sections describe two automated planning methods, including the *partial order planning* method, which can be easily adapted to support incremental problem solving and solution refinement; and the *hierarchical task network planning* method, which supports task decomposition.

4.3 Partial Order Planning

As described above, a problem solution is an agenda of action operators (or a plan) that defines transitions from givens to goals. A partial order plan (or POP in abbreviation)

is such an agenda where the operators in it are partially ordered. On the other hand, if the operators in an agenda are totally ordered (i.e. as a linear sequence), we call it a total order plan (or TOP in abbreviation). In the subsequent paragraphs, we describe the concepts and techniques of partial order planning based on Ghallab, Nau, and Traverso's summaries (Ghallab, Nau et al. 2004). The idea of TOP is first introduced, as TOP can be regarded as a special case of POP.

4.3.1 Total Order Plan

In mathematics, a totally ordered set is essentially a linear chain (Burris and Sankappanavar 1981). Therefore, a total order plan is a list of action operators that modify the mental state of the problem solver sequentially until the goal states are satisfied. Suppose $P=<A,G,G'>$ is a STRIPS problem domain. A total order plan to P can be denoted by $TOP=(a_1,a_2,\ldots,a_n)$. In addition, we use an ordered list $S=(S_1,S_2,\ldots,S_n)$ to denote the mental states after each action operator in the total order plan is executed. Then the total order plan *TOP* is valid if and only if it satisfies the following constraints:

1. $a_i \in A$, where $1 \leq i \leq n$;
2. $G \vdash precond(a_1)$, which means that the problem givens satisfy the precondition of the first action operator;
3. $S_i \vdash precond(a_{i+1})$, where $1 \leq i < n$. This means that the mental state after an action operator must satisfy the precondition of the next action operator.
4. $S_n \vdash G'$, which means that the mental state after the last action operator must satisfy the problem goals.

Figure 4.2 depicts the state transition in a total order plan that solves the map visualization problem desribed before. The retangular blocks in the diagram represent operators and the circles denote mental states. An arrow departing from an operator refers to the corresponding state change that produces the mental state targeted by the

arrow, which can be achieved by updating the previoius mental state with the operator's effects. On the other hand, an arrow departing from a circle means that the mental state satisfies the precondition of the operator the arrow points to. The initial mental state (S0 in the diagram) is defined by the problem givens, which are empty in this example, and the final mental state (S4 in the diagram) should satisfy the problem goal, which is to visualize a geospatial map.

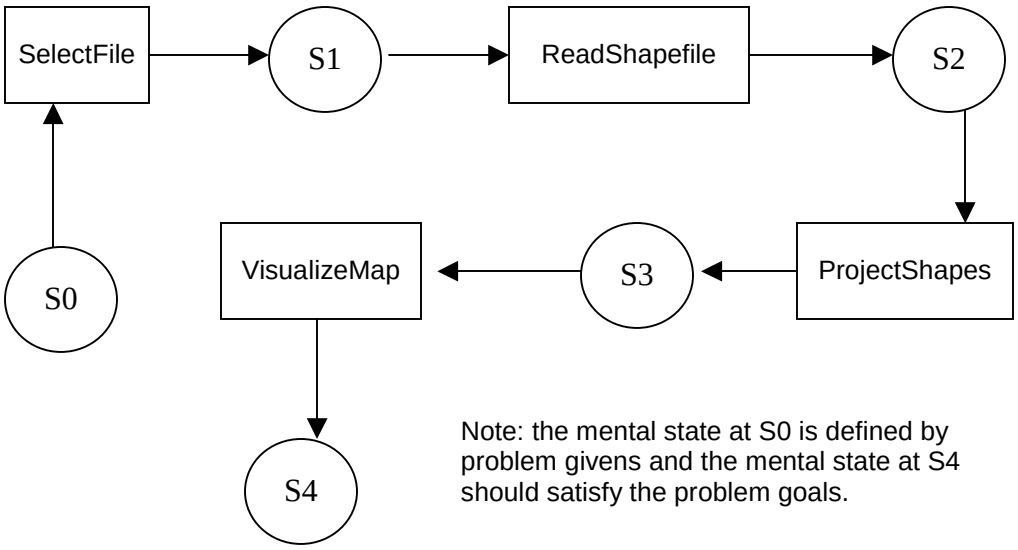


Figure 4.2 State transition diagram for the map visualization example, where the rectangular blocks denote operators and the circles represent mental states.

The concept of a total order plan had great impact on early automated planning methods, and is still widely used today due to its simplicity. Generally speaking a total order plan can be obtained by progression, i.e. searching operators either forwardly from goals to givens, or through regression, i.e. searching operators backwardly from goals to givens (Ghallab, Nau et al. 2004). Different heuristics can be used to guide progression and regression (Zhang 1999), and efficient methods such as planning graph analysis have been developed (Blum and Furst 1997). One tricky issue introduced by a total order plan is how to update a mental state with new effects. This is essentially a belief revision problem that can become very complicated in some cases (Eiter and Gottlob 1992). In this study, we assume that state update is performed by a revision function $Rev(S_{i-1}, a_i)$ to

preserve the generality of our discussion, and details of the revision function will be delegated to implementation.

4.3.2 Partial Order Plan

However, in many cases a total order plan is too restrictive a requirement for real world problem solving. Although some problems, e.g. the map visualization example describe above, can be easily solved by a totally ordered list of operators, many other problems, e.g. the knowledge discovery example used in chapter 3, do not necessarily have a list-based solution.

Table 4.3 The STRIPS operator for the PCP Component.

State / Operator	Precondition	Effects
VisualizePCP	role(?pcp0, ShapeFile) javatype(?pcp0, java.lang.Object[])	role(?pcp1, OnScreenPCP)

For instance, consider the example depicted in figure 4.3, which extends the application in figure 4.1 by connecting a PCP component after the ShapeFileDataReader component to visualize the attributes of the dataset in a parallel coordinate plot. The STRIPS operator for the PCP component is described in table 4.3, which is similar to that of GeoMap. Although VisualizePCP is only connected with the ReadShapefile operator, a valid total order plan requires all operators to be completely ordered in a linear list, which could be one of the following:

(SelectFile, ReadShapefile, ProjectShapes, VisualizeMap, VisualizePCP)

(SelectFile, ReadShapefile, VisualizePCP, ProjectShapes, VisualizeMap)

(SelectFile, ReadShapefile, ProjectShapes, VisualizePCP, VisualizeMap)

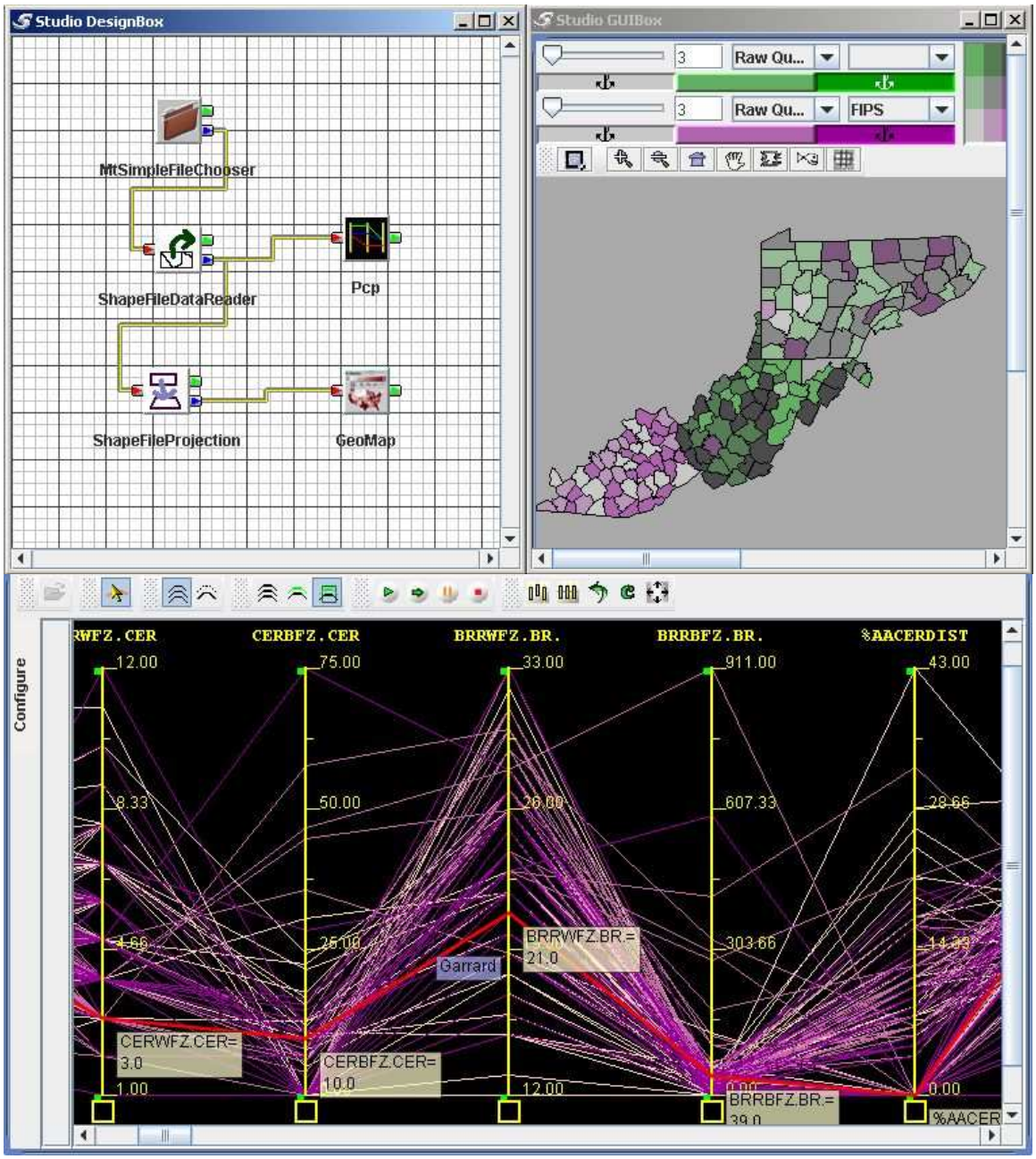


Figure 4.3 Visualizing a geospatial dataset using a map and a PCP. The application is obtained by extending the example in figure 4.1. A new PCP component is created and connected with ShapeFileDataReader, because since the component only needs attribute data from the dataset, map projection is unnecessary.

In other words, the VisualizePCP operator must be ordered with SelectFile, ProjectShapes, and VisualizeMap due to the way a total order plan is defined. This is not only unnecessary and inconvenient, but also hampers the flexibility of solutions, as any incremental addition or refinement of a plan would require a complete rebuilding of the linear ordering.

Such inconveniencies can be overcome by the approach known as Partial Order Plans (POP), which are plans that order two operators only when an explicit ordering between them is *required*. More specifically, a partial order plan for a STRIPS planning domain $P=<A,G,G'>$ is a 4-tuple $POP = <A_{pop}, E, B, L>$, where $A_{pop} \subseteq A$ is a set of action operators, E is a set of pairwise partial ordering constraints for action operators in A_{pop}, B is a set of variable substitutions, and L is a set of causal links. A partial ordering constraint in E takes the form (a_i, a_j), which means the action operator a_i must be applied before the action operator a_j. A causal link can be denoted by $a_i \xrightarrow{p} a_j$, which means a_i's effects are parts of the premise that conclude some precondition statements of a_j, and p contains satisfied precondition statements. Finally, B contains all variable substitutions under which the causal linkages stand.

A partial order plan can be viewed as a directed graph, where operators are nodes and partial ordering constraints define directional edges. A causal link $a_i \xrightarrow{p} a_j$ must be accompanied with a partial ordering constraint (a_i, a_j) in E, while a partial ordering constraint does not always have a correspondent causal link. Consequently, causal links can be regarded as annotations to a subset of edges. Figure 4.4 describes the partial order plan that corresponds to the example depicted in figure 4.3. It is noticeable that although every edge in the picture has a causal link attached to it, it is possible to define edges without any causal relationships, which specify mandatory ordering among causally unrelated operators. For instance, we can add an edge pointing from VisualizePCP to VisualizeMap, if necessary, to indicate that the PCP must always be visualized before the map.

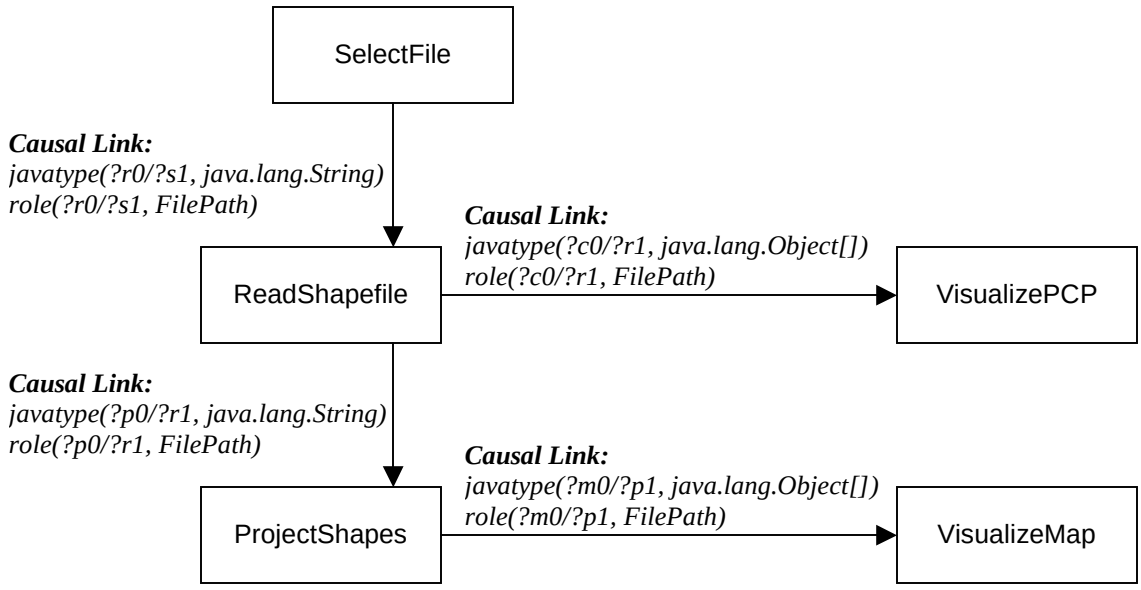


Figure 4.4 The partial order plan for the example depicted in figure 4.3, which is a directed graph with edges annotated by causal links.

A partial order plan $POP = <A_{pop}, E, B, L>$ is valid if and only if it has no *flaws*. There are two kinds of potential flaws in a partial order plan:

1. *Open sub-goals*, which could be either unsatisfied logical statements in the problem goals, or precondition statements with no causal links.
2. *Threats*, which are actions that interfere with a causal link, i.e. that the effects of one action contradicts with the precondition statements satisfied by the causal link.

We can use the following rules to eliminate flaws from a partial order plan. If a flaw is an open sub-goal, then search the operator set *A* and select an operator a_i whose effect statements logically satisfy the sub-goal (i.e. the effects deduce the sub-goal under certain variable substitutions). Append a_i to *M* and add the correspondent variable substitutions to *B*. If the sub-goal is a precondition of another action a_j, we also need to add a partial ordering constraint (a_i, a_j) and a causal link $a_i \xrightarrow{p} a_j$ to *L*, where *p* is the

satisfied sub-goal. Finally, all unsolved precondition statements of a_i become new open sub-goals.

If a flaw is a situation in which action a_k interferes with the causal link $a_i \xrightarrow{p} a_j$, we can have several options (called resolvers) to eliminate the flaw:

1. Promotion, which adds the constraint *(a_k, a_i)* to order a_k before a_i;
2. Demotion, which adds the constraint *(a_j, a_k)* to order a_k after a_j;
3. Separation, which adds additional variable bindings to make the negation of *p* not reachable from the effects of a_k;
4. Back-tracking in cases that all other resolvers fail. It replaces a_i with a new action and repeats the flaw elimination process.

A general partial order planning method is described by Penberthy and Weld in their proposal of UCPOP (Universal Conditional Partial Order Planner), one of the most representative partial order planners (Penberthy and Weld 1992). The algorithm uses a data structure called an *agenda*. Each element from an agenda is an ordered pair consisting of an action and one of its preconditions, denoted by (a, p) (Pednault 1986). In this study, the UCPOP procedure is modified to support domain semantics. Below is the pseudo-code of the partial order planning algorithm:

```
1    POP (<V, E, B, L>, Agenda)
2        If Agenda=NULL Then Return <V, E, B, L>
3        Select and pair (a_j, p) and remove it from Agenda
4        Select an action a_i satisfying p from the entire action set
5        If a_i = NULL Then Return FAILURE
6        If a_i ∉ V Then V = V ∪ {a_i} and update Agenda
7        If (a_i, a_j) ∉ B Then B = B ∪ {(a_i, a_j)}
8        If (a_i --p--> a_j) ∉ L Then L = L ∪ {a_i --p--> a_j}
9        Update B with the new variable bindings
10       For each threat on (a_i --p--> a_j) Do
11           Select a resolver to the threat
12           If the resolver is a Promotion or Demotion
13               Then Update B with the resolver
```

14 Else If the resolver is a separation

15 Then Update B with the resolver

16 Else if the resolver is NULL // which means we need to backtrack

17 Then Restore all changes since Line 4 and Goto 4

19 Return POP(<V, E, B, L>, Agenda) // This is a recursive call.

Compared with TOP, the definition of POP is more flexible, as ordering constraints are specified only when necessary. In fact a total order plan can be regarded as a special case of partial order plan, where the directed graph defined by POP happens to be a sequence. Moreover, as partial order plans are not always strictly sequential, they introduce the possibility of parallel processing, which may help analysts utilize the massive power of parallel computing to execute complex tasks. Most importantly, unlike most TOP methods that try to rebuild an operator list for every new problem, POP algorithms usually approach a problem by gradually adding / removing operators, partial ordering constraints, causal links and variable substitutions to / from an existing plan. Such a step-by-step "flaw-elimination" search strategy provides a better automation framework, which supports incremental problem solving and prototype refinement.

For instance, suppose we are exploring various visualization tools to incrementally build an application that best helps us identify cancer distribution patterns in the Appalachian region. With a POP automated solver, we can start with an empty partial order plan (i.e. A_{pop}, E, B L of the plan are all empty sets) plus an open goal "to show a geospatial map on the screen", which is the first visualization technique we are trying. As this goal is not currently satisfied, the POP solver will regard it as a flaw, and try to find an operator whose effects satisfy the goal. Consequently, the VisualizeMap operator will be selected, and its precondition will then become a new flaw. Repeating flaw elimination, a partial order plan similar to the simple map example in figure 4.1 will be generated. Now suppose we want to try another kind of visualization, e.g. PCP. Instead of prompting the automated solver to regenerate a new plan, we can submit a new open goal as a new flaw to the existing plan, e.g. "to show a PCP", and the automated

solver can eliminate it by adding the PCP operator to the plan, and appending an edge and causal link to connect it with the ReadShapefile operator.

Similarly, if we regard the example in figure 4.1 as a prototype, the same POP procedure can be directly used to refine it into the application depicted in figure 4.3. To summarize, both the definition of POP and its flaw-elimination algorithm are incremental in nature. As a result, geographical problem solving methods such as incremental problem solving and prototype refinement can be implemented as a POP procedure that gradually generates a partial order plan (which could be empty at the very beginning) by progressively eliminating its flaws.

4.4 Hierarchical Task Network Planning

Hierarchical Task Network (HTN) planning focuses on the compositional relationships between action operators. Different from the TOP and POP methods that reason with the precondition and effects of operators, HTN planning algorithms usually rely on the stored structure of task decomposition to generate valid plans. This section discusses the basic concepts of HTN and examines the application of HTN methods to geographical problem solving.

4.4.1 HTN Planning

The central concept of HTN planning is a *task network*. As one can see in the previous discussion, a partial order plan is essentially a graph-based structure comprised of connected operators. A task network is a structure similar to a partial order plan, but with two critical differences:

1. A task network contains not only operators, but also abstract "tasks" that can be accomplished by other task networks. However, a task network is considered atomic or "executable" only when it contains no abstract tasks (i.e. only has operators). Consequently, task decomposition can be achieved by

replacing all abstract tasks with relevant task networks repetitively until there are no abstract tasks in the network.

2. A task network is not generated by any planning algorithm. On the contrary, task networks are regarded as stored *a priori* knowledge generalized from past experience or directly obtained from the knowledge domain. This view implies an approach of automated planning dramatically different from the POP methods, because the planning procedure spends most effort examining the task decomposition hierarchies defined by the stored task networks rather than evaluating individual operators.

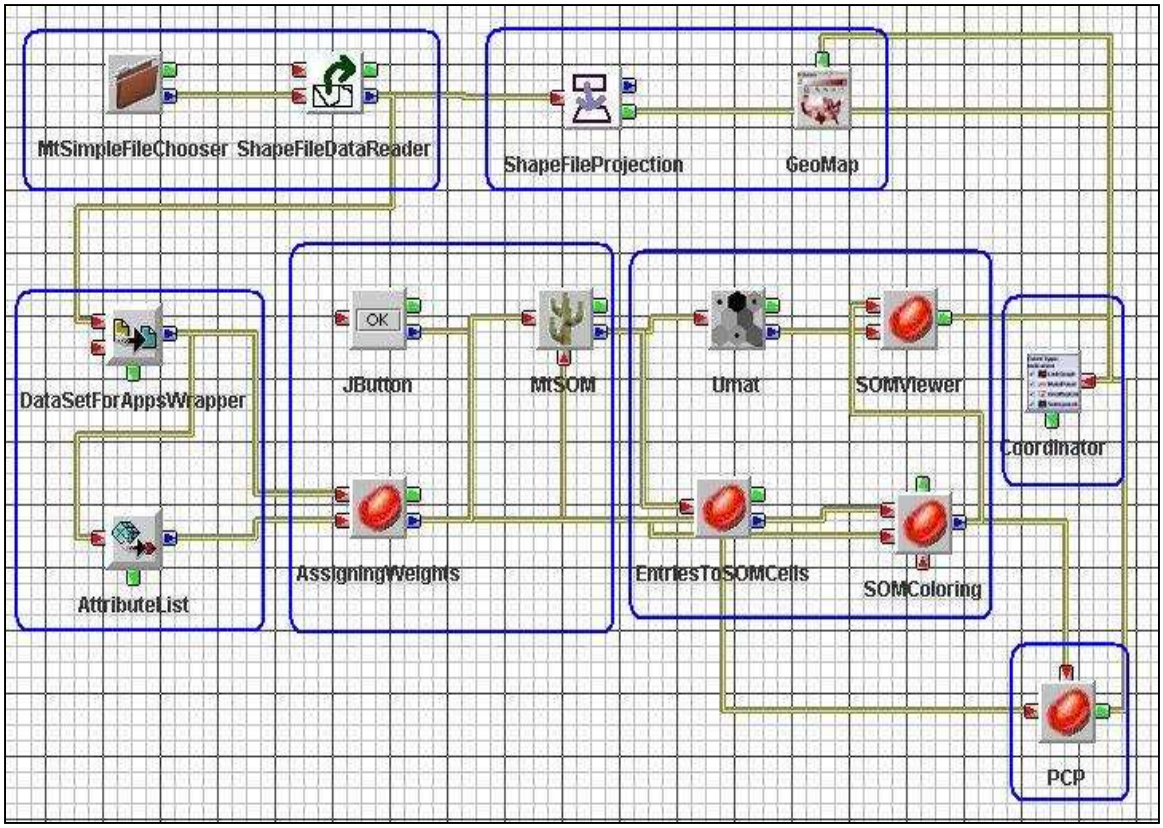


Figure 4.5 Dividing a geographical knowledge discovery application into smaller fragments. Each blue box delineates a smaller group of components composition that performs a sub-task, which can be created in a relatively independent manner.

Figure 4.5 shows the dataflow diagram of the geographical knowledge discovery example used in chapter three, which can be divided into several smaller tasks. Solutions

to those smaller tasks can be found separately and then combined into the final solution. As each sub-task is much simpler than the overall problem, e.g. with less components and connections, and can be solved in a relatively independent manner, the overall complexity of the original problem is reduced significantly.

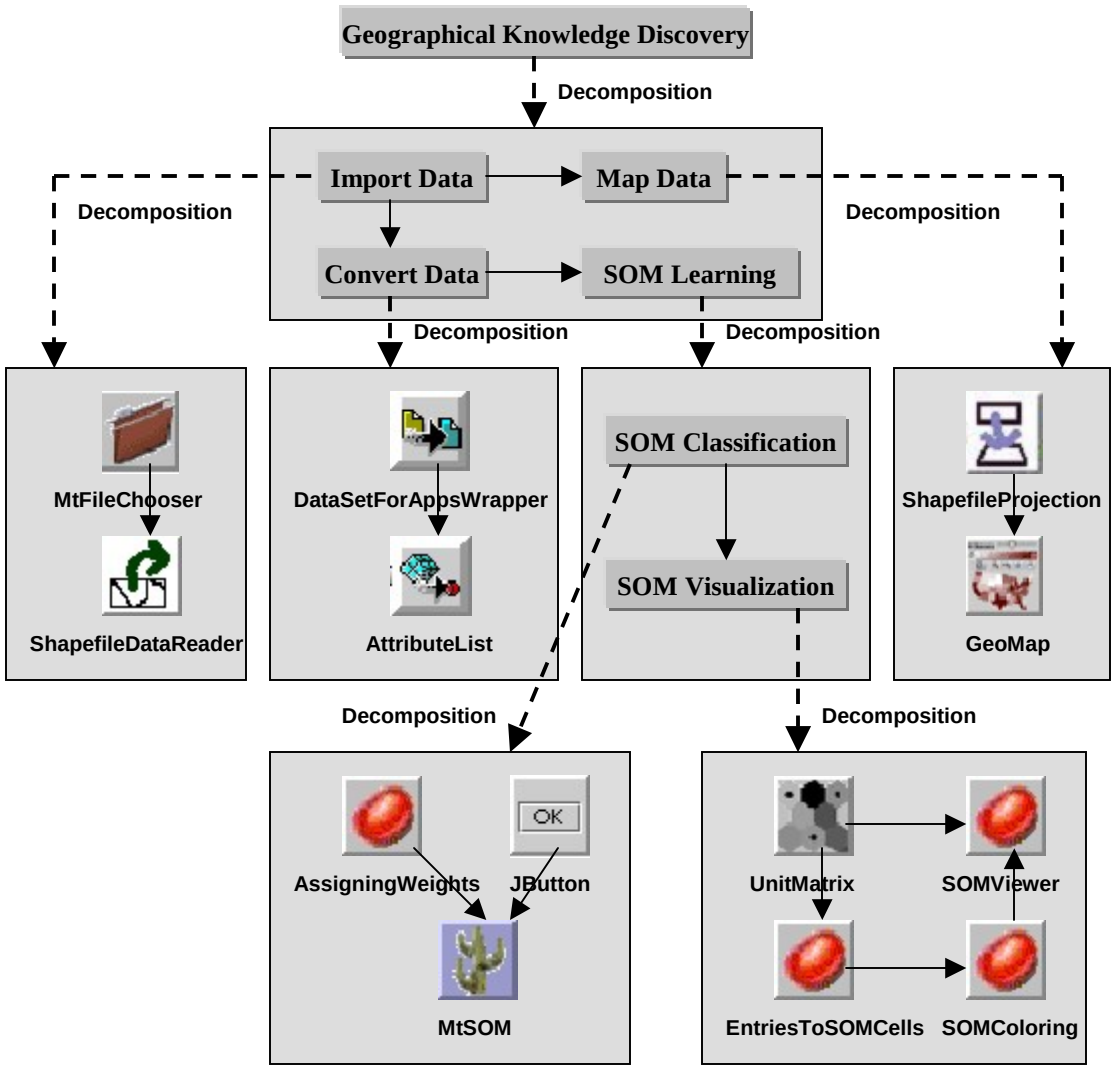


Figure 4.6 One possible task decomposition hierarchy for the geographical knowledge discovery application described in figure 4.5. To simplify displaying, the components "Coordinator" and "PCP" are not included. Such omissions do not hamper the generality of task decomposition, as both of them can be easily included as additional tasks in the hierarchy.

Figure 4.6 depicts one possible task decomposition hierarchy for the geographical knowledge discovery application depicted in figure 4.5. The task of "geographical

knowledge discovery" describes the overall application, which can be regarded as a top-level task network that only contains itself. This task can be decomposed by a task network with four tasks, including "import data" that parses the shapefile dataset, "convert data" that converts the imported dataset into the format used by the visualization tools, "map data" that displays the geospatial map, and "SOM Learning" that clusters the dataset using the self-organizing map algorithm. The relationships between these tasks are specified by partial ordering constraints, i.e. directional edges among them. Finally, as shown in the picture, all abstract tasks can be further decomposed level-by-level into concrete operators (i.e. those directly supported by individual components).

It is argued by some researchers that HTN methods are not truly "automated planners", because the structure of task decomposition is not generated by the planning method and must be defined by the user (Erol, Hendler et al. 1994). Nevertheless, HTN methods provide useful insights about the hierarchical nature of task decomposition, which is present in many problem scenarios. Therefore, they usually lead to more effective and efficient solutions than pure POP planners (Erol, Nau et al. 1994). Since task decomposition is a very useful geographical problem solving method, a HTN planner that supports task decomposition is an ideal complement to the POP algorithm introduced before.

4.4.2 Simple Task Network

The detailed design of a HTN method is dependent on the structural definition of task networks. This study adopts the approach known as *Simple Task Network* (STN), which defines a task network as a directed graph (Ghallab, Nau et al. 2004). While more generic, non-graph-based HTN methods are also thoroughly studied in AI literatures, STN is selected mainly because a graph-based task network can be easily and accurately associated with a partial order plan, and eventually converted into a dataflow network.

The specific definitions of STN concepts used in this study are given below (Ghallab, Nau et al. 2004).

4.4.2.1 Task

A task is defined by the 2-tuple *t = (name, para),* where *name* is the syntactic identifier of the task and *para* is a list of task parameters. A task is *primitive* if it shares the name with an action operator; otherwise, we say the task is *non-primitive* or *abstract*. For example, the following 2-tuple denotes an abstract task of geographical knowledge discovery:

(gkd, ["appalachian.shp"]).

In this task, "gkd" (geographical knowledge discovery) is the task name, and "applalachian.shp" is the file name for the cancer dataset of the Appalachian region. Given the STRIPS operators list in table 4.2, this task is non-primitive since no operator is named as "gkd". On the contrary, the following task is a primitive one:

(ReadShapefile, ["appalachian.shp"]).

This basically defines an atomic task that can be performed by the ReadShapefile operator. In other words, a non-primitive task cannot be directly accomplished and thus must be further decomposed, while a primitive task represent a smallest unit of action supported by a given operator. Referring back to figure 4.6, all gray boxes in the picture are non-primitive tasks, and all component icons can be regarded as potential primitive tasks.

4.4.2.2 Simple Task Network

A simple task network is an acyclic directed graph denoted by *STN = (V,E),* where *V* is a set of nodes and *E* is a set of edges. Every node $v \in V$ contains a task t_v and every edge $e \in E$ represents a partial ordering constraint between two nodes in *V*, denoted by $e = (v_i, v_j)$. As we have mentioned before, a STN is very similar to a partial order plan,

with the exception that the elements of a STN could be abstract tasks that are subject to further decomposition. If a STN only contains primitive tasks, we call it a *primitive task network*; otherwise it is a *non-primitive task network*. For example, figure 4.7 displays some task networks from the task decomposition hierarchy described in figure 4.6. The top two networks are non-primitive task networks, since they contain non-primitive tasks, and the bottom three are primitive ones.

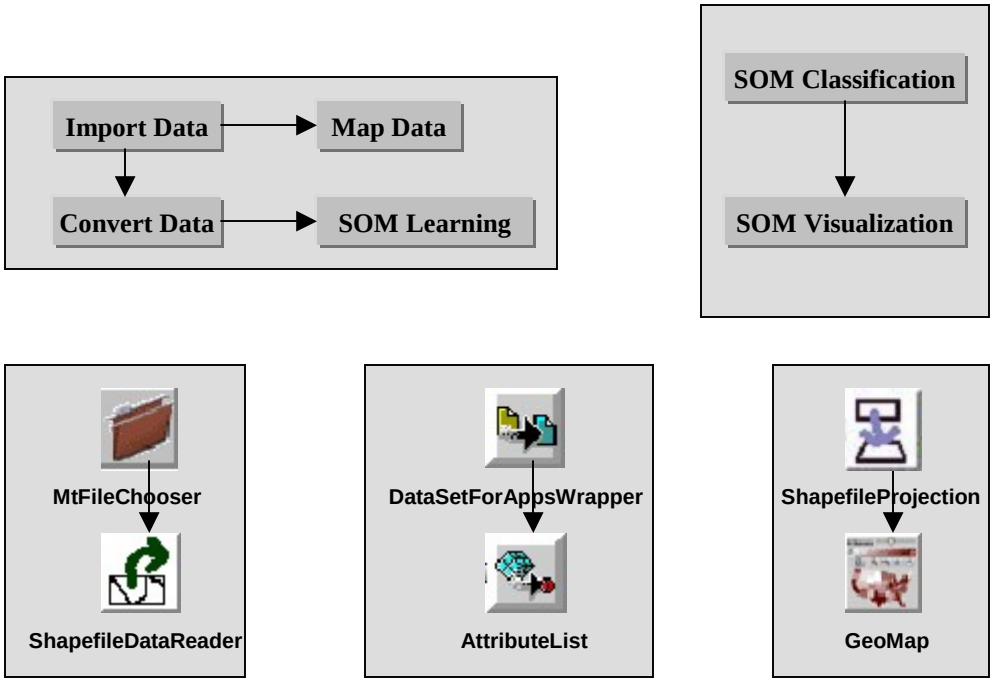


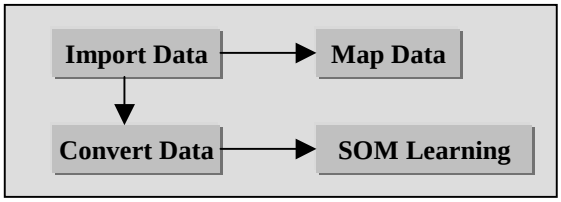
Figure 4.7 Some simple task networks from the task decomposition hierarchy described in figure 4.6.

4.4.2.3 Simple Task Method

A task method is defined by the 4-tuple *m=(name, t, precond, net)*, where *name* is the syntactical identifier of the method, *t* is a non-primitive task decomposed by the method, *precond* is a set of logical statements representing the precondition of the method, and *net* denotes a simple task network that defines the decomposition structure. A task method associates a task with a task network, and thus defines the basic structure of task decomposition. For example, table 4.4 describes an example task method that decomposes the geographical knowledge discovery task using the task network displayed

in figures 4.6 and 4.7. Similarly, each of the four sub-tasks in the resulted network can be further decomposed using an appropriate task method (e.g. "Import-Data" can be decomposed by a task method whose task network connects "MtSimpleFileChooser" and "ShapefileDataReader"), and eventually the entire geographical knowledge discovery application can be generated by task decomposition.

Table 4.4 A task method for the geographical knowledge discovery task.

Name	gkd-method
Task	(gkd, [?input])
Precondition	role(?input, FilePath) javatype(?input, java.lang.String)
Task Network	

4.4.2.4 Simple Task Decomposition

Given a task *t* and a method *m*, if there is a variable binding *B* that makes *task(m) = t*, then we say *m* is *relevant* to *t* under *B*, denoted by *B(t) = task(m)*, and the task *t* is decomposed by the method *m*, denoted by *Decomp(t, m, B) = net(m)*. This means that under the variable binding *B*, *t* can be accomplished by executing the task network defined by *m*. For example, given the task method described in table 4.4, suppose we are trying to solve the following task:

(gkd, ["appalachian.shp"]).

It is obvious that this task can be obtained by replacing the variable "*?input*" inside *(gkd, [?input])* with the file name "*appalachian.shp*". Consequently, we can say that the task method *gkd-method* described in table 4.4 is relevant to the targeted task *(gkd, ["appalachian.shp"])* under the variable binding *[?input / "appalachian.shp"]*.

4.4.2.5 Simple Task Network Decomposition

Given a simple task network *net = (V, E)*. Supposing $v \in V$ is a node and t_v is the task associated with *v*, if there is a method *m* and a variable binding *B* such that *m* is relevant to t_v under *B*, then the decomposition of the task network can be defined by *Decomp(net, v, m, B) = net'*, where *net'* is obtained by replacing the node *v* with the task network in *m*.

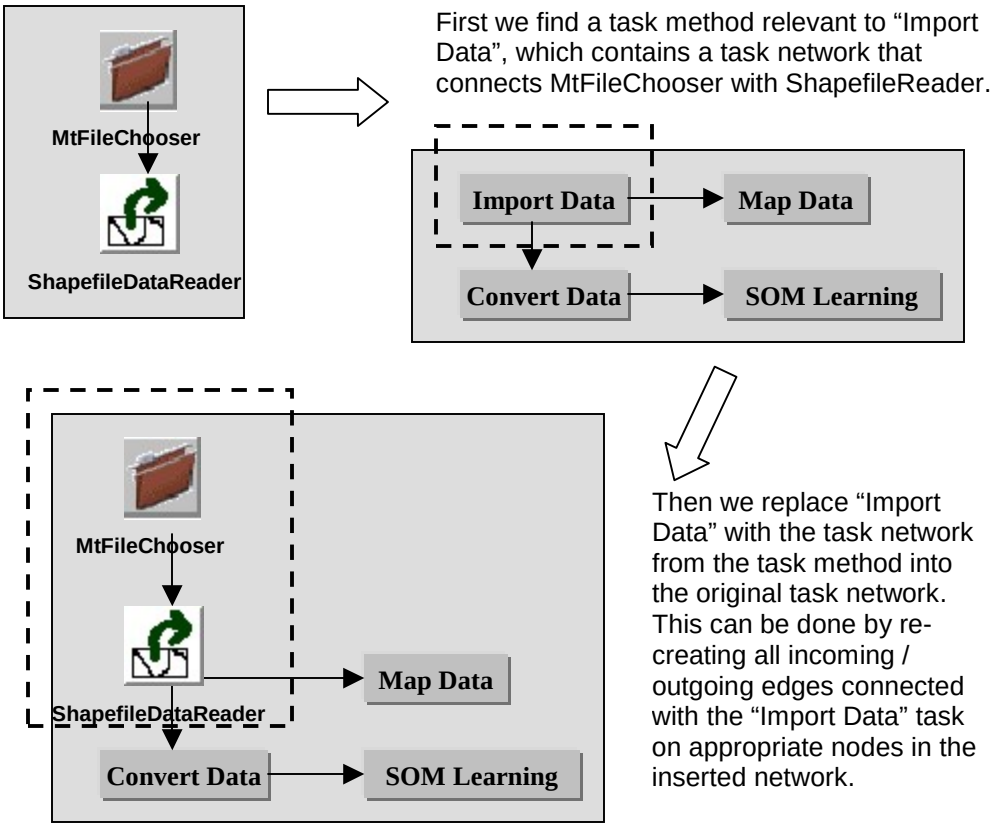


Figure 4.8 An example of simple task network decomposition, which replaces the non-primitive task "Import Data" task with a more detailed primitive task network.

The details of task network decomposition — especially the procedure of task network modification — can be quite technical, as different implementations may employ different strategies to optimize the performance of decomposition. The general steps of decomposition can be summarized as follows:

1. Remove *v* and all edges linked to it from the original task network *net*;
2. Append all nodes and edges from *net(m)* to *net*;
3. For *all* nodes in the original task network that directly precede *v*, add an edge from them to *every* node in *net(m)* that has no predecessors;
4. For *all* nodes in the original task network that directly follow *v*, add an edge to them from *every* node in *net(m)* that has no successors.

Figure 4.8 depicts an example of simple task decomposition, where the non-primitive task "Import Data" is decomposed by a method with a task network that connects MtSimpleFileChooser with ShapeFileDataReader. As shown in the picture, the node for "Import Data" is replaced by the task network defined in the decomposing method, where the edges originally connected with "Import Data" are re-created and connected with ShapeFileDataReader, the *last* node in the newly inserted network.

4.4.3 STN Planning Method

Compared with a POP planning domain that contains only action operators, a STN domain is defined by the 2-tuple *<A, M>*, where *A* is a set of action operators and *M* is a set of task methods. The knowledge of task decomposition is represented as task networks in the task methods. In addition, while the goals of a POP problem are defined as mental states, in STN planning a problem goal can be regarded as a desired task, e.g. the task of geographical knowledge discovery mentioned above. Consequently, a STN problem can be defined by a 4-tuple *<A, M, G, net>*, where *A* is a set of action operators, *M* is a set of task methods, *G* is a set of mental states representing the problem givens, and *net* denote the top-level task network that decomposes the desired task. We call net the *initial task network*, and in most cases we can regard it as a network that only contains the desired task itself. Then the STN planner can be designed as a procedure that recursively decomposes the initial task network until a primitive task network is obtained.

The abstract STN planning algorithm is described below, where *G* denotes the problem givens and net represents the task network of the initial decomposition method (Ghallab, Nau et al. 2004):

```
1     HTN(G, net)
2         If net is primitive Then
3             plan := find a primitive solution for net;
4             If plan = NULL Then Return FAILURE;
5             Else Return plan;
6         Else
7             v := select a non-primitive task node v from net;
8             m := select a method where task(m)'s name equals to v's task;
9             β := variable bindings that renames all variables of m;
10            net' = Decomp(net, v, m, β);
11            Return HTN(G, net'); // Recursive call.
```

Notice that the selection of a task method (line 8) must be subject to the condition defined by the problem givens. In other words, the precondition of the selected task methods must be satisfied by the mental states specfied by the problem givens.

We can generalize a partial order plan into a task method by giving it a task name. Given a partial order plan *POP = (V, E, B, L)* and a task name *N*, a task method denoted by *m = (name, precond, net)* can be generated based on the following rules:

1. *name(m) = N*;
2. *precond(m)* = the union of preconditions from all action operators in *V* that have no predecessors;
3. *net(m) = (V, E).*

Now we can combine the partial order planning and STN planning methods. A semantic problem space is defined as the 4-tuple (P_S,G,G',t), where $P_S = \langle A, M \rangle$ is an problem domain with task methods, *G* represents problem givens, *G'* denotes problem goals, and *t* is a task goal that suggests the initial point of task decomposition. If the result of decomposition meets the requirements of both givens and goals, we return it as a

valid solution, otherwise we start the partial order planning procedure from scratch. The combined method can be described using the pseudo-code below:

```
1       PLAN(G, G', t)   //t is only a hint;
2               Agenda := Create an Agenda according to G';
3               plan := Initial Partial Order Plan;
4               If t = NULL Then Return POP(plan, Agenda)       // No task hint;
5               m := select a method where task(m) = t;
6               If m = NULL Then Return POP(plan, Agenda)     // No task method;
7               htn = HTN(G, net(m))     // Generate a partial order plan using HTN;
8               If htn = NULL Then Return POP(plan, Agenda);
9               htn = POP(htn, Agenda)            // Refine the HTN Plan
10              If htn = NULL Return POP(plan, Agenda) // Fail to refine the HTN Plan.
11              Return htn        // Return the refined HTN Plan
```

4.5 Automated Geographical Problem Solving

This section discusses the connections between the semantic model proposed in chapter 3 and the POP and STN methods described in this chapter. Generally speaking, we can regard the semantic model as a *specification* framework that describes geographical problem solving semantics, while viewing the cognitive model of problem solving and automated planning methods as the internal implementation structures and algorithms, which provide a mechanism to efficiently reason with semantics specified in the semantic model. The main goal of this section, therefore, is to define how knowledge structures in the semantic model can be consistently compiled into computable POP and STN models, and how the results from the automated planners can be translated back into shareable knowledge defined by the semantic model.

Three types of transformation are indispensable for automated geographical problem solving:

1. The transformation from a component action into a STRIPS operator that can be evaluated by automated planners;
2. The transformation from a dataflow network into a partial order plan;

3. The transformation from a partial order plan into a dataflow network.

It is noticeable that task methods required by STN planning do not need to be translated into the semantic model, because they only contain information relevant to solution generation. Users do not need to be aware of a particular task method unless they want to examine the internal logic of the automated planner. In the future, it is possible to develop ways to present the structure of task decomposition to users if such information is believed to be useful to human analysts. However, in this study, task methods remain as internal implementation structures that only function in the backend.

4.5.1 Generating STRIPS operators

A STRIPS operator can be generated from a component action according to the following rules:

1. Every component action corresponds to one and only one STRIPS operator;
2. Any port that represents an input or output of a component is translated into a formal variable;
3. For any port consumed by the component action, the role and Java data type of the port are translated into precondition statements of the STRIPS operator;
4. For any port produced by the component action, the role and Java data type of the port are translated into effect statements of the STRIPS operator.

The operators listed in table 4.2 are in fact generated using the above rules, and therefore we will not give additional examples of the translation.

4.5.2 Translating Dataflow Networks into Partial Order Plans

Translating dataflow networks into partial order plans is necessary because in many cases an automated problem solver needs to refine an existing (fragment of) dataflow network into a new application. A dataflow network can be translated into a partial order plan according to the following rules:

1. For every link in a dataflow network, the component actions connected by the link are translated into two action operators in the resulting partial order plan according to the operator generation rules described in 4.4.1. Moreover, a partial ordering constraint (or an directional edge) is created between them.
2. If a link has a data channel, then a causal link is created for the partial ordering constraint corresponding to the link. Moreover, the data channel is converted into a set of literals and appended to the causal link's satisfaction list *p*.

4.5.3 Extracting Dataflow Networks from Partial Order Plans

Extracting a dataflow network from a partial order plan is the final step of automated geographical problem solving, where the output partial order plan generated by the automated problem solver is converted into front-end dataflow applications. A partial order plan can be converted into a dataflow network according to the following rules:

1. Every partial ordering constraint in the partial order plan should be translated into a dataflow link, where the source and target nodes of the partial ordering constraint are converted into component actions;
2. For every causal link in the partial order plan, create a data channel for each causal literal that contains a data port. Any variable "*?x*" is regarded as a data port if and only if it appears both in a role predicate *role (?x, R)* and a Java data type predicate *javatype (?x, T)*, where "*R*" is a data concept and "*T*" a Java data type.

A good example is the dataflow application depicted in figure 4.3 and the partial order plan depicted in figure 4.4, which can be translated to each other using the rules described in 4.5.2 and 4.5.3. Readers can refer to chapter seven for a more user-oriented

description about how the techniques introduced above can help analysts construct useful applications.

4.6 Summary

This chapter presents a framework of automated geographical problem solving based on cognitive models of human problem solving and automated planning techniques. From a cognitive point of view, problem solving is a human reasoning process that evaluates the problem givens to derive an agenda of action steps, the execution of which should accomplish the problem goals. Under the cognitive models originally developed by Newell and Simon, both problem givens and goals can be encoded as mental states, and action steps can be modeled as state transition operators. The reasoning process of problem solving then equals to finding the appropriate combination of state transitions that transit the given state towards the goal state. Two categories of automated planning methods can be employed to support geographical problem solving, including the partial order planning method that incrementally solves a problem and gradually refines an existing partial solution, and the hierarchical task network planning method that solves a complex problem based on task decomposition. An automated problem solver that combines both approaches is described in this chapter, which adopts the general POP procedure and the STN planning method. Finally, we discuss the relationships between the automated problem solver and the semantic model proposed in chapter three, including how to translate knowledge from the semantic model into the automated problem solver's inference model, how to convert a dataflow application into a partial order plan, and how to extract a dataflow from a partial order plan. With the semantic model as the representational framework and the automated problem solver as the inference engine, a Semantic Geospatial PSE can now provide fully automated semantic supports to different scenarios of geographical problem solving.

Chapter 5

REPRESENTING GEOGRAPHICAL PROBLEM SOLVING SEMANTICS IN OWL

5.1 Introduction

The main focus of this chapter is to translate the semantic model described in the previous chapters into a practical knowledge representation language. As the primary goal of a Geospatial PSE is to integrate computational resources from disparate origins, it is critical for the system to adopt a portable representational framework to support the sharing of semantics in an open, heterogeneous environment. Consequently, this chapter presents a knowledge representation language based on RDF (Resource Description Framework) and OWL (Web Ontology Language), which have recently been defined by W3C (The World Wide Web Consortium) as the knowledge exchange standards on the Semantic Web (Leuf 2006). A Geospatial PSE can then use the OWL-based language as the external format to create, store, and share semantics, while utilizing the formalisms defined by the semantic model to build the internal knowledge processing and automated reasoning structures.

Four sections are included in this chapter. After this introduction, section 5.2 describes the general framework of knowledge representation for geographical problem solving, and discusses why RDF and OWL are important to a Semantic Geospatial PSE. Section 5.3 presents an OWL-based knowledge representation language, which translates all elements of the semantic model into OWL terms. Finally, summaries are provided in section 5.4.

5.2 General Framework of Knowledge Representation

Generally speaking, any computational artifact can be regarded as a knowledge representation (Sowa 1999). However, knowledge representations created and used in a knowledge-based application usually need to fulfill the following roles (Davis, Shrobe et al. 1993):

1. It must define a set of *surrogates* for the represented objects;
2. It must commit to one or several *ontologies* that define the (intended) meanings of the representation;
3. It must incorporate a mechanism of *reasoning* to enable inference over the represented knowledge;
4. It should be an appropriate medium of *efficient computation* that facilitates digital knowledge processing;
5. It should serve as an effective medium of *human expression* that supports human comprehension.

The semantic model described in the previous chapters satisfies roles 1-3, as it employs the first order predicate calculus to create a system of symbolic surrogates and logical reasoning, while using model theory to strictly define the meanings of geographical problem solving. However, the semantic model has not put much emphasis on roles 4 and 5. Although the first order predicate calculus and model theory are well-established formal tools, they do not automatically translate into good knowledge engineering choices. The flat, linear style of first order formulae and the list structure of first order theory is not a convenient way to construct real world knowledge, because a moderately complex specification may result in a huge list of lengthy and unreadable first order formulae. In addition, reasoning in first order predicate calculus is seldom fully supported in real world knowledge-based systems, as most systems only implement a

subset of the first order language (e.g. Prolog) and others use their own (and sometimes proprietary) languages of knowledge representation (e.g. the CLIPS expert shell).

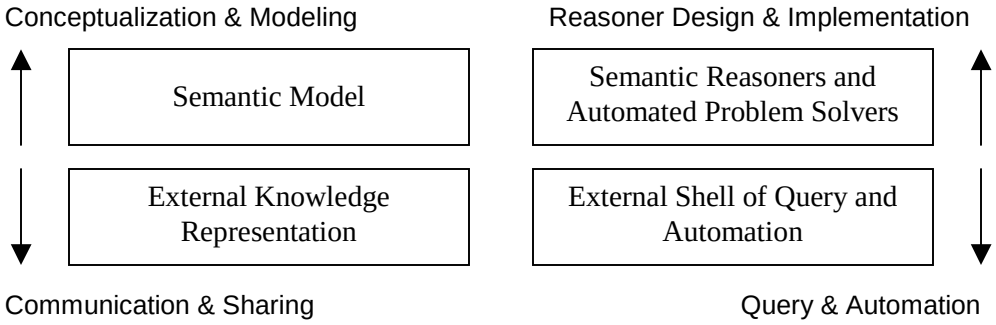


Figure 5.1 The role of knowledge representation in a Semantic Geospatial PSE.

Consequently, it is necessary for a Semantic Geospatial PSE to embed the semantic model into a more feasible knowledge representation framework. As depicted in figure 5.1, a knowledge representation is essentially a carrier of knowledge models conceptualized and formalized based on the semantic model. The inference engine implements the automated problem solvers and inference rules according to the semantic model, and at the same time provides the front-end interface for query and automation based on the external knowledge representation to utilize its portability. In other words, one can regard the semantic model as a conceptual tool and the goal of this chapter is to develop the concrete representation language.

This study selects W3C's OWL (which is defined on top of RDF) as the basic knowledge representation language. The reasons for using OWL are two fold:

1. First, OWL is W3C's standard knowledge exchange language over the Semantic Web. Therefore, encoding geographical problem solving semantics in OWL can significantly improve the accessibility and portability of a Semantic Geospatial PSE.
2. Second, OWL is based on a number of established methods in AI and knowledge representation, including semantic networks (Shastri 1988), conceptual graphs (Sowa 1976), and description logic (DL) (Baader,

Calvanese et al. 2003). An OWL-based knowledge representation can take advantage of those methods to ease the implementation of the inference engine. Furthermore, as the relations between these methods and first order predicate calculus have been well studied, it is not difficult to translate the semantic model, which is specified in first order predicate calculus, into OWL.

5.2.1 Levels of Representation

Similar to the layered knowledge modeling approach described in chapter three (figure 3.8), a knowledge representation can also be developed from three levels of abstraction (Uschold and Jasper 1999):

1. Level 0 is the layer of the ground knowledge representation language, which defines the domain-independent elements of knowledge representation, including the basic syntax and inference rules;
2. Level 1 is the layer of domain ontology, which defines common elements of the targeted knowledge domain. In other words, domain ontology is domain specific but application independent;
3. Level 2 is the layer of operational data, which defines knowledge elements collected from specific cases of application. Operational data is both domain and application dependent.

Such a layered view allows different groups of knowledge engineers and users to focus on their specific interests rather than exploring all details of knowledge representation. Most importantly, these three abstraction levels can be easily mapped to the semantic knowledge model defined in chapter three, where Level 0 is equivalent to the ground formal system, Level 1 can be associated with the object language L_K AND its semantic axioms, and Level 2 corresponds to factual axioms. This means we can concentrate on developing appropriate ontologies at Level 1 to map the semantics of the object language L_K into a ground knowledge representation language such as OWL.

Figure 5.2 shows the detailed levels of knowledge representation adopted by this study. The Level 0 language is OWL, which is also defined upon a stack of underlying representations, including low-level schema such as ASCII and Unicode that encode textual characters, the eXtensible Markup Language (XML) that provides the basic syntax to structure textual information, and the Resource Description Framework (RDF) that specifies relationships between uniquely identified resources. The Level 1 language can be developed as OWL ontologies for the semantic model of geographical problem solving, and consequently, the Level 2 operational data includes any semantics specified based on these ontologies. The remainder this section gives a detailed explanation of RDF and OWL, while the Level 1 and Level 2 representations will be the foci of the next section.

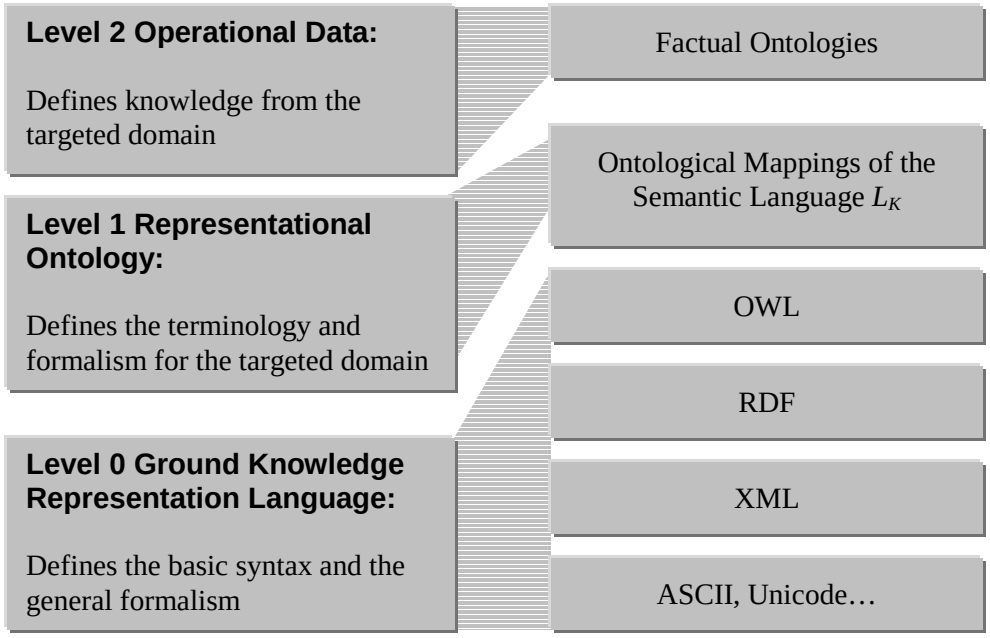


Figure 5.2 Leveled representation of knowledge. Level0: Domain-independent language; Level1: Specialized domain-dependent language; Level2: Knowledge content.

5.2.2 RDF

RDF provides the basic syntax to represent knowledge as structurally related resources. A resource is a knowledge representation entity that may denote anything in the targeted knowledge domain, including computational artifacts, web pages, and even human individuals. The design of RDF aims to resolve two major issues of knowledge representation in an open, heterogeneous environment such as the Internet.

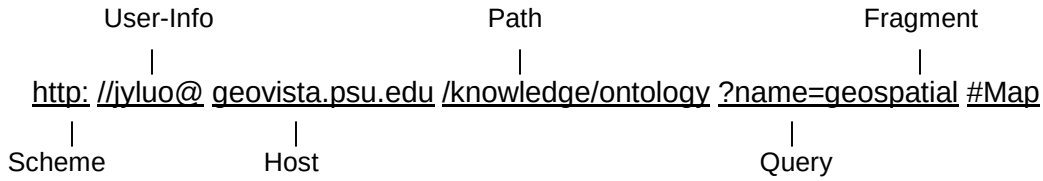
The first is how to uniquely identify resources originated from different knowledge sources. For example, consider the concept of a "map". In geography and related fields, a map is a symbolized geometric representation of geospatial phenomena. However, in mathematics the same term is commonly used as a synonym of a continuous function. Moreover, in computer science and engineering, "map" is also a frequently used data structure. Now imagine a geographical problem solving scenario that requires inputs from all three fields, such as one that estimates the 5-year county level population change of Pennsylvania using a mathematical map (function), stores the results in a map data structure, and visualizes the data in a geospatial map. Since there is no rule to enforce the use of the term "map" in distributed knowledge sharing, how to disambiguate the term in different contexts and correctly process the correspondent information becomes a potential problem. RDF resolves this problem by using Uniform Resource Identifiers (URIs), a standardized textual encoding scheme that assures a unique resource identification, to denote resources. One can refer to the W3C's specification for a detailed definition of URI's syntax (IETF 1998).

Figure 5.3 depicts an example that uses URIs to distinguish the three "map" concepts mentioned above, with the syntactical composition of the first URI explained. The URI of an RDF resource is usually divided into a *namespace* and a *local name*, where the namespace contains the scheme, user-info, host, path, and other scheme-specific parts of the URI, and local name is a short and easy-to-remember fragment of the

URI. In other words, we can regard a local name as the human readable part of a resource, while its namespace define the origin of the resource in a heterogeneous environment. Finally, a resource can also be succinctly denoted by the following format to achieve better human readability:

ns:localname

Here "*ns*" is a XML *prefix* declared in the XML-based RDF document, which is normally an abbreviation of the original namespace. For example, assuming the URIs in figure 5.3 are specified in a RDF document that declares "geospatial", "math", "data-structure" as the namespace prefixes to the three URIs in the respective order, the three resources can then be specified as "geospatial:Map", "math:Map", and "data-structure:Map" accordingly.



http: //jyluo@ geovista.psu.edu /knowledge/ontology ?name=math #Map

http: //jyluo@ geovista.psu.edu /knowledge/ontology ?name=data-structure #Map

Figure 5.3 how to use URIs to uniquely identify three "Map" concepts.

The second issue that RDF focuses on is to define a structure of knowledge representation based on binary relations. More specifically, in a RDF document a resource is related to another resource through an RDF property, which is a resource denoting a binary relation. A particular case of a binary relation is known as an RDF statement (or a triple), which has a subject (the resource related from), a predicate (the RDF property) and an object (the resource related to). Figure 5.4 shows an example of an RDF statement, where the RDF resource "geospatial:Mapping" is the subject, the RDF property "rdfs:subClassOf" is the predicate, and the resource "geospatial:Operation" is

the object. It means that the concept denoted by "geospatial:Mapping" is a specialization of the concept denoted by "geospatial:Operation", or more simply, mapping is one kind of geospatial operation.

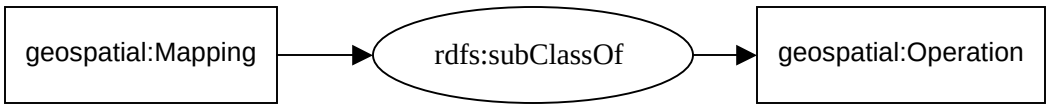


Figure 5.4 An example RDF statement, which relates "geospatial:Mapping" to another resource "geospatial:Operation" through the property "rdfs:subClassOf".

The philosophy of RDF is to build complex knowledge representations based on the relatively simple structure of binary relations, which as a result, reduce the syntactical overhead of knowledge sharing. RDF adopts a formalism called EC logic (Existential Conjunction Logic) developed by Charles. S. Perice (Roberts 1973). As a result, an RDF statement is always an existentially quantified 2-ary predicate and an RDF document is regarded as a conjunction of all statements it contains. By doing so, RDF excludes complex features of the first order predicate calculus (such as universal quantification and multi-ary predicates) that could make knowledge representations difficult to share. On the other hand, the binary structure used in RDF is associated with a popular knowledge representation scheme in AI known as Semantic Networks, which can encode knowledge into a directed graph (Shastri 1988; Sowa 1991). Consequently, an RDF knowledge representation can utilize various graph visualization techniques to create more intuitive and comprehensible presentations for human users (such as the one depicted in figure 5.4).

5.2.3 OWL

Pure RDF ontologies are flat, unordered collections of triple RDF statements, which are inconvenient when representing complex resources. This motivates the development of OWL, which provides additional formalism to define a frame-based and

object-oriented knowledge representation layer on top of RDF. The foundation of OWL is Description Logic (DL), a formalism that models the world as a two-part abstraction, including a universe of terminologies and a universe containing facts described by these terminologies (Baader, Calvanese et al. 2003). A universe of terminologies defines the vocabulary to represent the targeted knowledge domain, which is usually organized into categories or class hierarchies. An instance is a resource that takes one or several classes from the terminological universe as its type(s). Consequently, DL defines a strongly typed frame or object-oriented language for knowledge representation. This can be achieved by adding more axioms to RDF. Those additional axioms can be classified into two groups: 1) axioms about the relationships among classes (which constitute the Terminological Box, or TBox); and 2) axioms about the assertion of instances or facts (which constitute the Assertion Box or ABox).

For example, the RDF statement demonstrated in Figure 5.4 can be extended to define two classes in the terminological universe. TBox specifies relationships and rules among terminological concepts such as the specialization or sub-classing relationship. Suppose there is a third class denoted by “geospatial:BivariateMap”, which is a sub-class of “geospatial:Map”, then the resource “geospatial:BivariateMap” is also a sub-class of “geospatial:Operation”, because the TBox of OWL defines “sub-class” to be a transitive relation. On the contrary, ABox specifies relationships and rules associated with instances. Consider a new RDF resource called “geospatial:MapInstance” is declared to denote the mapping functionality provided by the GeoMap component. If we define “geospatial:MapInstance” to be an instance of “geospatial:Map”, then it also becomes an instance of “geospatial:Operation”, the super class of “geospatial:Map”. This is because the ABox axioms in OWL state that a class’s instances are also instances of the class’s super classes.

From the perspective of language specification, OWL is essentially a combination of the RDF Schema (a simple RDF-based category definition language) and the more

complicated DL-based OWL specification. Currently W3C provides three dialectics of OWL (W3C 2004):

- OWL DL defines a complete set of DL formalism, which enforces DL rules in knowledge representation. In other words, in OWL DL everything must be a class or instance of a class, and un-typed RDF resources are disallowed;
- OWL Lite is a subset of OWL DL, which removes several elements that are believed to be less important to most applications or considered more difficult to support in knowledge representation tools;
- OWL Full is a convenient language to mix OWL elements and non-OWL RDF statements. The knowledge structures defined in OWL Full are interpreted in an RDF-compatible manner, where OWL axioms are not enforced. In other words, if a resource is not specified as an OWL class or instance, it will be treated as a plain RDF resource rather than an error.

This study adopts OWL Full as the basic knowledge representation foundation, because it allows free combinations of OWL and non-OWL RDF resources, which provides more flexibility and compatibility than OWL DL and OWL Lite.

5.3 Ontologies of Geographical Problem Solving

OWL ontologies of geographical problem solving can be developed by mapping all elements of the semantic model into OWL classes, properties, and instances. Generally speaking, the mapped OWL ontologies will use the default name namespace "http://www.geovista.psu.edu/ontology/sgps.owl#". In subsequent discussions, if a resource is identified only by its local name (e.g. "Map"), then it uses the default name space. The abbreviation "sgps" is also used in places where the default name space must be explicitly specified (e.g. "sgps:Map").

The semantic model described in chapter three is comprised of a DoD model, a language model, and a knowledge model. The rest of this section maps elements from

these three models into OWL resources. It is noticeable that while the DoD and language models are mapped as Level 1 ontologies, the knowledge model, which contains factual knowledge, should be mapped to Level 2 ontologies (operational data).

5.3.1 Mapping the DoD Model to OWL

A straightforward approach is to map all the domains and sub-domains described by the semantic model into OWL classes. First, the union of all semantic domains and sub-domains constitutes the super domain of geographical problem solving, which can be represented as the root class. Below is the OWL declaration of the root class:

```
<owl:Class rdf:about="Root"/>
```

Consequently, all the domains in the semantic models can be defined as direct or indirect sub-classes of the root class. Table 5.1 lists the OWL class declarations for the four primary domains and their sub-domains. It is noticeable that the mappings of the data and task domains are different from other domains in the following way: the elements of data and task domains are mapped to *sub-classes* of the "sgps:Data" and "sgps:Task" classes respective, while the elements from other domains are mapped to *instances* of classes. This is because we want to utilize the TBox and ABox axioms of OWL to define categories of data and tasks. For example, consider the GeoMap component that accomplishes the high-level task of "mapping". The GeoMap component will be specified as an instance of the "sgps:Component" class, while the mapping task will be defined as a sub-class of "sgps:Task", as shown below.

```
<sgps:Component rdf:about="GeoMap">
        <!-- properties of the GeoMap component -->
</sgps:Component>

<owl:Class rdf:about="Mapping">
        <rdfs:subClassOf
        rdf:resource="http://www.geovista.psu.edu/ontology/sgps.owl#Task"/>
        <!-- properties of the Mapping task. -->
</owl:Class>
```

Table 5.1 OWL mappings for the semantic domains.

Domains and Sub-Domains	OWL Mappings
Proxy	<owl:Class rdf:about="Proxy"> <rdfs:subClassOf rdf:resource="Root"/> </owl:Class>
Component	<owl:Class rdf:about="Component"> <rdfs:subClassOf rdf:resource="Proxy"/> </owl:Class>
Port	<owl:Class rdf:about="Port"> <rdfs:subClassOf rdf:resource="Proxy"/> </owl:Class>
Action	<owl:Class rdf:about="Action"> <rdfs:subClassOf rdf:resource="Proxy"/> </owl:Class>
System	<owl:Class rdf:about="System"> <rdfs:subClassOf rdf:resource="Root"/> </owl:Class>
Java Class	<owl:Class rdf:about="JavaClass"> <rdfs:subClassOf rdf:resource="System"/> </owl:Class>
Java Primitive	<owl:Class rdf:about="JavaPrimitive"> <rdfs:subClassOf rdf:resource="System"/> </owl:Class>
Java Event	<owl:Class rdf:about="JavaEvent"> <rdfs:subClassOf rdf:resource="System"/> </owl:Class>
Java Method	<owl:Class rdf:about="JavaMethod"> <rdfs:subClassOf rdf:resource="System"/> </owl:Class>
Runtime	<owl:Class rdf:about="Runtime"> <rdfs:subClassOf rdf:resource="Root"/> </owl:Class>
Node	<owl:Class rdf:about="Node"> <rdfs:subClassOf rdf:resource="Runtime"/> </owl:Class>
Link	<owl:Class rdf:about="Link"> <rdfs:subClassOf rdf:resource="Runtime"/> </owl:Class>
Dataflow	<owl:Class rdf:about="Dataflow"> <rdfs:subClassOf rdf:resource="Runtime"/> </owl:Class>
User	<owl:Class rdf:about="User"> <rdfs:subClassOf rdf:resource="Root"/> </owl:Class>
Data	<owl:Class rdf:about="Data"> <rdfs:subClassOf rdf:resource="User"/> </owl:Class>
Task	<owl:Class rdf:about="Task"> <rdfs:subClassOf rdf:resource="User"/> </owl:Class>

Another noticeable issue is that instances of system classes (e.g. JavaClass) should encode the implementation identifiers required by the software platform (e.g. Java class names and method signatures). In predicate calculus those identifiers can be simply represented as string constants. In OWL, additional schemes must be developed to encode them since a string representation of an implementation identifier cannot be automatically used as a valid URI. This study creates a special URI encoding scheme to represent implementation identifiers, which takes the following form:

platform:platform-dependent-scheme://identifier

In this scheme, "platform" is the name of the implementation platform, e.g. "java" means the platform is Java; while "platform-dependent-scheme" can be used to specify whether the URI is a Java class, Java primitive, Java event, or Java method signature. Finally, "identifier" represents the concrete implementation identifier (such as a class name), which should be encoded into a valid URI string. Table 5.2 provides several examples of this encoding scheme.

Table 5.2 Sample URIs of Java implementation identifiers.

	Identifiers	**Encoded URIs**
Java Class	java.lang.String	java:class://java.lang.String
Java Primitive	int	java:primitive://int
Java Method	setFileName(java.lang.String)	java:method://setFileName(java.lang.String)
Java Event	dataSetChange	java:event://dataSetChange

5.3.2 Mapping the Semantic Language to OWL

The semantic language described in chapter three is essentially a first order language based on the specially designed signature. Therefore, mapping the semantic language to OWL equals to developing appropriate OWL properties and constraints for the predicate symbols in the signature.

5.3.2.1 Mapping Elements for the Proxy Domain

Five predicate symbols are related to the proxy domain, including *input/2, output/2, performs/2, consumes/2, and produces/2*. Readers can refer back to chapter three for their detailed descriptions. Their corresponding OWL mappings are included in table 5.3, where each predicate symbol is mapped into an OWL object property.

Table 5.3 OWL mappings for proxy predicate symbols.

Predicate Symbols	OWL Mappings
input/2	<owl:ObjectProperty rdf:about="input"> <rdfs:domain rdf:resource="Component"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty>
output/2	<owl:ObjectProperty rdf:about="output"> <rdfs:domain rdf:resource="Component"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty>
performs/2	<owl:ObjectProperty rdf:about="performs"> <rdfs:domain rdf:resource="Component"/> <rdfs:range rdf:resource="Action"/> </owl:ObjectProperty>
consumes/2	<owl:ObjectProperty rdf:about="consumes"> <rdfs:domain rdf:resource="Action"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty>
produces/2	<owl:ObjectProperty rdf:about="produces"> <rdfs:domain rdf:resource="Action"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty>

5.3.2.2 Mapping Elements for the System Domain

In chapter three, seven predicate symbols are created for the Java-based system domain, including *extends/2, assignable/2, convertible/2, javatype/2, implementedBy/2, notifiedBy/2, setter/2*, and *getter/2*. The first three of them, i.e. *extends/2, assignable/2*, and *convertible/2*, are related to type casting and data conversion that can be handled by the Java runtime platform. For instance, a Java runtime environment will maintain the information about the Java class hierarchies to decide whether instances of one Java class can be converted to instances of another Java class. In other words, during a course of logical query, the inference engine can consult the system platform to obtain such

information, and therefore, it is unnecessary to explicitly address them in the knowledge representation (Otherwise, we have to build an ontology of all available Java classes and data types). Consequently, only *javatype/2*, *notifiedBy/2*, *setter/2*, and *getter/2* are mapped into OWL. The proposed OWL mappings are described in table 5.4.

Table 5.4 OWL mappings for system predicate symbols.

Predicate Symbols	OWL Mappings
javatype/2	<owl:ObjectProperty rdf:about="javatype"> <rdfs:domain rdf:resource="Port"/> <rdfs:range> <owl:Class> <owl:unionOf> <owl:Class rdf:resource="JavaClass"/> <owl:Class rdf:resource="JavaPrimitive"/> </owl:unionOf> </owl:Class> </rdfs:range> </owl:ObjectProperty>
implementedBy/2	<owl:ObjectProperty rdf:about="implementedBy"> <rdfs:domain rdf:resource="Component"/> <rdfs:range rdf:resource="JavaClass"/> </owl:ObjectProperty
notifiedBy/2	<owl:ObjectProperty rdf:about="notifiedBy"> <rdfs:domain rdf:resource="Action"/> <rdfs:range rdf:resource="JavaEvent"/> </owl:ObjectProperty>
setter/2	<owl:ObjectProperty rdf:about="setter"> <rdfs:domain rdf:resource="Port"/> <rdfs:range rdf:resource="JavaMethod"/> </owl:ObjectProperty>
getter/2	<owl:ObjectProperty rdf:about="getter"> <rdfs:domain rdf:resource="Port"/> <rdfs:range rdf:resource="JavaMethod"/> </owl:ObjectProperty>

5.3.2.3 Mapping Elements for the Runtime Domain

As displayed in table 5.5, five predicate symbols are defined for the dataflow-based runtime domain. It is noticeable that there are two 3-ary predicate symbols, including *connects/3* and *channel/3*, which cannot be directly mapped to any single OWL classes or properties. The predicate symbol *connects/3* declares a link that connects a

source node to a target node. Therefore, it is represented as two OWL properties in table 5.5, with one for the source node and the other for the target node.

Table 5.5 OWL mappings for runtime predicate symbols.

Predicate Symbols	OWL Mappings
hasNode/2	<owl:ObjectProperty rdf:about="hasNode"> <rdfs:domain rdf:resource="Dataflow"/> <rdfs:range rdf:resource="Node"/> </owl:ObjectProperty>
hasLink/2	<owl:ObjectProperty rdf:about="hasLink"> <rdfs:domain rdf:resource="Dataflow"/> <rdfs:range rdf:resource="Link"/> </owl:Class>
instanceOf/2	<owl:ObjectProperty rdf:about="instanceOf"> <rdfs:domain rdf:resource="Node"/> <rdfs:range rdf:resource="Component"/> </owl:ObjectProperty>
connects/3	<owl:ObjectProperty rdf:about="source"> <rdfs:domain rdf:resource="Link"/> <rdfs:range rdf:resource="Node"/> </owl:ObjectProperty> <owl:ObjectProperty rdf:about="target"> <rdfs:domain rdf:resource="Link"/> <rdfs:range rdf:resource="Node"/> </owl:ObjectProperty>
channel/3	<owl:Class rdf:about="Channel"> <owl:Restriction> <owl:onProperty rdf:resource="outport"/> <owl:cardinality>1</owl:cardinality> </owl:Restriction> <owl:Restriction> <owl:onProperty rdf:resource="inport"/> <owl:cardinality>1</owl:cardinality> </owl:Restriction> </owl:Class> <owl:ObjectProperty rdf:about="outport"> <rdfs:domain rdf:resource="Channel"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty> <owl:ObjectProperty rdf:about="inport"> <rdfs:domain rdf:resource="Channel"/> <rdfs:range rdf:resource="Port"/> </owl:ObjectProperty> <owl:ObjectProperty rdf:about="hasChannel"> <rdfs:domain rdf:resource="Link"/> <rdfs:range rdf:resource="Channel"/> </owl:ObjectProperty>

On the other hand, *channel/3* declares data channels attached to a link. Since one link may have more than one data channel, a separate OWL class for data channel is

created, which binds an output port from the link's source node with an input port in the link's target node. As a result, a link can declare multiple instances of the "DataChannel" class using the "hasChannel" property to define multiple data channels.

5.3.2.4 Mapping Elements for the User Domain

The user domain contains two predicate symbols, including role/2 that defines the data role of a port, and accomplishes/2 that defines the task that a component action accomplishes. Both of them are mapped to the "rdf:type" property from the basic RDF vocabulary, which declares the type of a RDF resource. Since data and task concepts are defined as sub-classes rather than instances of "Data" and "Task", mapping both predicate symbols into "rdf:type" will make ports and actions to be parts of OWL's ABox (i.e. instances of OWL classes). The benefit of this approach is that now we can utilize the TBox and ABox axioms defined by OWL to specify and inference with categories of data and tasks, without any additional effort.

5.3.3 Building Knowledge Representations in OWL

The example described in section 3.4.3 of chapter three will be used to illustrate how to translate a semantic knowledge model into an OWL-based knowledge representation. Basically this example models the semantic knowledge that describes the connection between a MtSimpleFileChooser node and a ShapeFileDataReader node in a dataflow. Table 3.12-3.15 from chapter three list the system, user, proxy, and runtime semantics of the example, which can be translated into the following OWL ontologies.

5.3.3.1 The System Ontology

The system ontology is the OWL translation of the semantics described in table 3.12 of chapter three, which declares resources of relevant Java identifiers based on the URI encoding scheme described before:

```
<sgps:JavaClass rdf:about="java:class://java.lang.String"/>
```

```
<sgps:JavaClass rdf:about="java:class://jh9gpz.io.MtSimpleFileChooser"/>
<sgps:JavaClass
        rdf:about="java:class//edu.psu.geovista.data.ShapeFileDataReader"/>
<sgps:JavaMethod rdf:about="java:method://getAbsolutePath()"/>
<sgps:JavaMethod rdf:about="java:method://setFileName(java.lang.String)"/>
<sgps:JavaEvent rdf:about="java:event://action"/>
<sgps:JavaEvent rdf:about="java:event://dataSetChange"/>
```

5.3.3.2 The User Ontology

The user ontology is the OWL translation of the semantics described in table 3.13 of chapter three. It declares a data class named “FilePath”, which is a sub-class of “sgps:Data”, and two task classes, including “SelectFile” and “ReadShapefile”, both of which are sub-classes of “sgps:Task”. The OWL specification is described below:

```
<owl:Class rdf:about="FilePath">
        <rdfs:subClassOf
        rdf:resource="http://www.geovista.psu.edu/ontology/sgps.owl#Data"/>
</owl:Class>
<owl:Class rdf:about="SelectFile">
        <rdfs:subClassOf
        rdf:resource="http://www.geovista.psu.edu/ontology/sgps.owl#Task"/>
</owl:Class>
<owl:Class rdf:about="ReadShapefile">
        <rdfs:subClassOf
        rdf:resource="http://www.geovista.psu.edu/ontology/sgps.owl#Task"/>
</owl:Class>
```

5.3.3.3 The Proxy Ontology

The proxy ontology is the OWL translation of the semantics described in table 3.14 of chapter three. It contains two instances of the “sgps:Component” class, two instance of the“sgps:Action” class, two instances of the “sgps:Action” class, and two instances of the “sgps:Port” class. The detailed OWL specification is below:

```
<sgps:Component rdf:about="FileChooser">
        <sgps:output rdf:resource="FileChooserOutput"/>
        <sgps:performs rdf:resource="FileChooserAction"/>
        <sgps:implementedBy
                rdf:resource="java:class://jh9gpz.io.MtSimpleFileChooser"/>
</sgps:Component>
<sgps:Component rdf:about="ShapeFileDataReader">
        <sgps:input rdf:resource="ShapeFileDataReaderInput"/>
        <sgps:output rdf:resource="ShapeFileDataReaderOutput"/>
        <sgps:performs rdf:resource="ShapeFileDataReaderAction"/>
        <sgps:implementedBy
```

```
        rdf:resource="java:class//edu.psu.geovista.data.ShapeFileDataReader"/>
</sgps:Component>
<sgps:Action rdf:about="FileChooserAction">
        <rdf:type rdf:resource="SelectFile"/>
        <sgps:notifiedBy rdf:resource="java:event//action"/>
</sgps:Action>
<sgps:Action rdf:about="ShapeFileDataReaderAction">
        <rdf:type rdf:resource="ReadShapefile"/>
        <sgps:notifiedBy rdf:resource="java:event//dataSetChange"/>
</sgps:Action>
<sgps:Port rdf:about="FileChooserOutput">
        <rdf:type rdf:resource="FilePath"/>
        <sgps:javatype rdf:resource="java:class://java.lang.String"/>
        <sgps:getter rdf:resource="java:method://getAbsolutePath()"/>
</sgps:Port>
<sgps:Port rdf:about="ShapeFileDataReaderInput">
        <rdf:type rdf:resource="FilePath"/>
        <sgps:javatype rdf:resource="java:class//java.lang.String"/>
        <sgps:setter rdf:resource="java:method://setFileName(java.lang.String)"/>
</sgps:Port>
```

5.3.3.4 The Runtime Ontology

The runtime ontology is the OWL translation of the semantics described in table 3.15 of chapter three. It declares the dataflow fragment that connects a FileChooser node with a ShapeFileDataReader node. The OWL specification is described as the follows:

```
<sgps:Dataflow rdf:about="ExampleDataflow">
        <sgps:hasNode rdf:resource="FileChooserNode"/>
        <sgps:hasNode rdf:resource="ShapeFileDataReaderNode"/>
        <sgps:hasLink rdf:resource="PassSelectedFile"/>
</sgps:Dataflow>
<sgps:Node rdf:about="FileChooserNode">
        <sgps:instanceOf rdf:resource="FileChooser"/>
</sgps:Node>
<sgps:Node rdf:about="ShapeFileDataReaderNode">
        <sgps:instanceOf rdf:resource="ShapeFileDataReader"/>
</sgps:Node>
<sgps:Link rdf:about="PassSelectedFile">
        <sgps:source rdf:resource="FileChooserNode"/>
        <sgps:target rdf:resource="ShapeFileDataReader"/>
        <sgps:hasChannel rdf:resource="FilePathChannel"/>
</sgps:Link>
<sgps:Channel rdf:about="FilePathChannel">
        <sgps:outport rdf:resource="FileChooserOutput"/>
        <sgps:inport rdf:resource="ShapeFileDataReaderInput"/>
</sgps:Channel>
```

5.4 Summary

This chapter presents an OWL-based language to represent knowledge defined by the semantic model proposed in chapter three, which can be used to share semantics of geographical problem solving over the Semantic Web. The OWL-based language can be developed by mapping elements of the semantic model, including the semantic domains, the semantic language, and semantic knowledge models, into OWL classes, properties, and instances. A Semantic Geospatial PSE can then use the language as the external format of knowledge sharing to take the advantage of OWL's expressiveness and portability in an open, heterogeneous environment, while sticking to the semantic model in internal knowledgebase design and inference engine implementation.

Chapter 6

REFERENCE IMPLEMENTATION

6.1 Introduction

This chapter presents a reference implementation for the Semantic Geospatial PSE based on the concepts and ideas described in the previous chapters. The design of the reference implementation is discussed in section 6.2, which states the general design goals of the system and illustrates how semantics can be engineered as an integral part of a Geospatial PSE. After that, section 6.3 describes the main functionalities of the reference implementation and demonstrates the system's tremendous automation power in supporting different scenarios of geographical problem solving from the user's perspective. Finally, section 6.4 provides a brief summary of the chapter.

6.2 The Design of the Reference Implementation

The central design goal of the reference implementation is to hide the technical details of component selection, configuration, and composition from users. More specifically, it provides users with:

- Categorized resource directories that organize computational resources according to their high-level tasks rather than relying on the names given to methods by programmers, or other implementation identifiers;
- A resource query service that discovers useful functionalities for users according to their specific application requirements;
- Automated solution building support that constructs dataflow applications based on user-specified application goals.

Figure 6.1 depicts the design of the reference implementation. The three-layer architecture of the design is based on the general PSE framework presented in chapter two, which includes a system, a semantic, and a user layer.

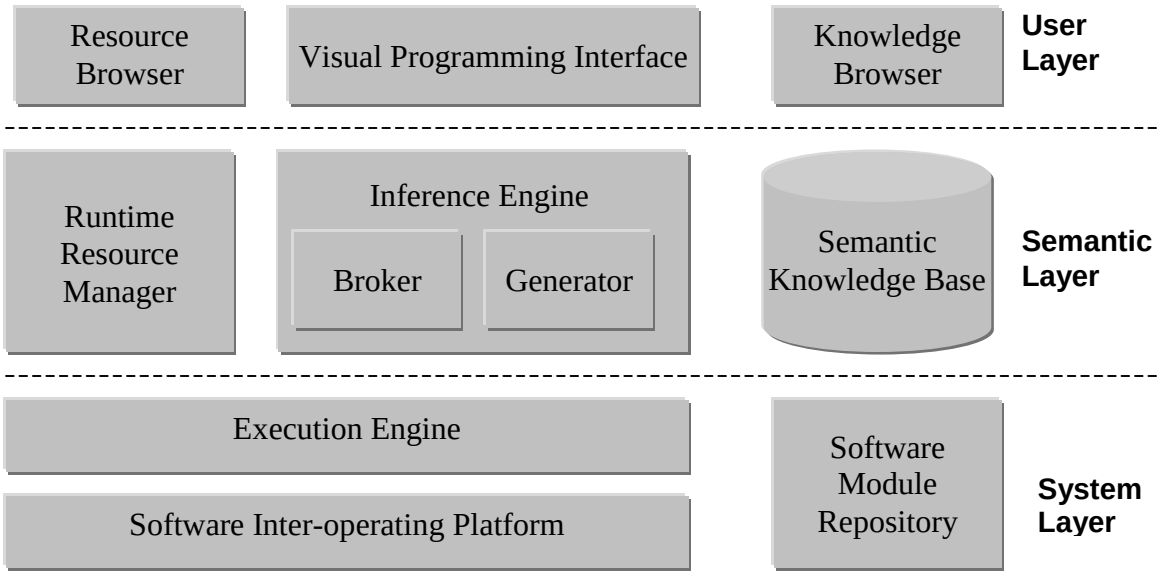


Figure 6.1 The detailed architectural design of the reference implementation.

The system layer provides the basic digital computing infrastructure for the reference implementation, including a software interoperation platform that defines the working protocols among heterogeneous software modules, an execution engine that executes dataflow applications, and a software module repository. The semantic layer provides the infrastructure to manage and process geographical problem solving semantics. It contains a knowledge base that stores the semantic descriptors of software modules, an inference engine that retrieves software modules and generates dataflow applications according to user requests, and a runtime resource manager that maintains the connections between the runtime instances of computational resources (software modules and the datasets they are handling) and their semantic descriptors in the knowledge base. The user layer provides the necessary user interface for analysts to access and manipulate computational resources integrated by the system. It includes a visual programming interface that supports both manual and (semi) automated dataflow

construction, a knowledge browser to visualize and select resources according to their semantic descriptors, and a runtime resource browser to display the UI elements of dataflow applications executed by the execution engine.

Table 6.1 The role and implementation strategy of each system module.

Module	Role	Implementation details
Software Interoperation Platform	Providing the basic runtime environment for heterogeneous computational resources.	Java platform plus the JavaBeans component standard.
Execution Engine	Executing JavaBeans dataflows specified in the runtime sub-language of the proposed semantic model.	A Java-based dataflow interpreter is implemented as the core element of the execution engine.
Software Module Repository	Storing and managing software components physically.	The default file-based strategy used by Java is adopted, which stores software components in jar files.
Semantic Knowledgebase	Storing and managing semantic descriptors of software components.	The semantic knowledgebase is implemented in Prolog, which represents semantics as prolog clauses.
Semantic Inference Engine	Resource brokerage (i.e. query and resource retrieval) and solution generation (via automated planning).	The inference engine is implemented partly in Java and partly in Prolog. The general procedure of inference is implemented in Java implements and Prolog code is embedded wherever logical deduction or knowledgebase query is necessary. .
Runtime Resource Manager	Maintaining the linkages between runtime software resources and their semantic descriptors in the knowledgebase.	The resource manager is primarily built on top of a hashed lookup table that maps between runtime resources and their semantic descriptors.
Visual Programming Interface	Viewing, editing, and creating JavaBeans dataflow applications.	An interactive GUI panel is implemented, with which users can create dataflows either manually or (semi) automatically with the help of the semantic inference engine.
Knowledge Browser	Browsing and selecting the semantics of computational resources.	A hierarchical tree panel is implemented, with which users can browse and select desired concept from hierarchical categories.
Runtime Resource Browser	Displaying the UI components and outputs of runtime dataflows.	Each UI component is displayed in a separate dialog box.

Table 6.1 lists the roles and implementation details of different layers and components that comprise the system. Figure 6.2 shows the basic GUI interface of the system, in which a simple dataflow is created in the visual programming interface (the top window) to map the demographics of the counties in the west coast of United States. The output map is displayed by the runtime resource browser in the bottom window.

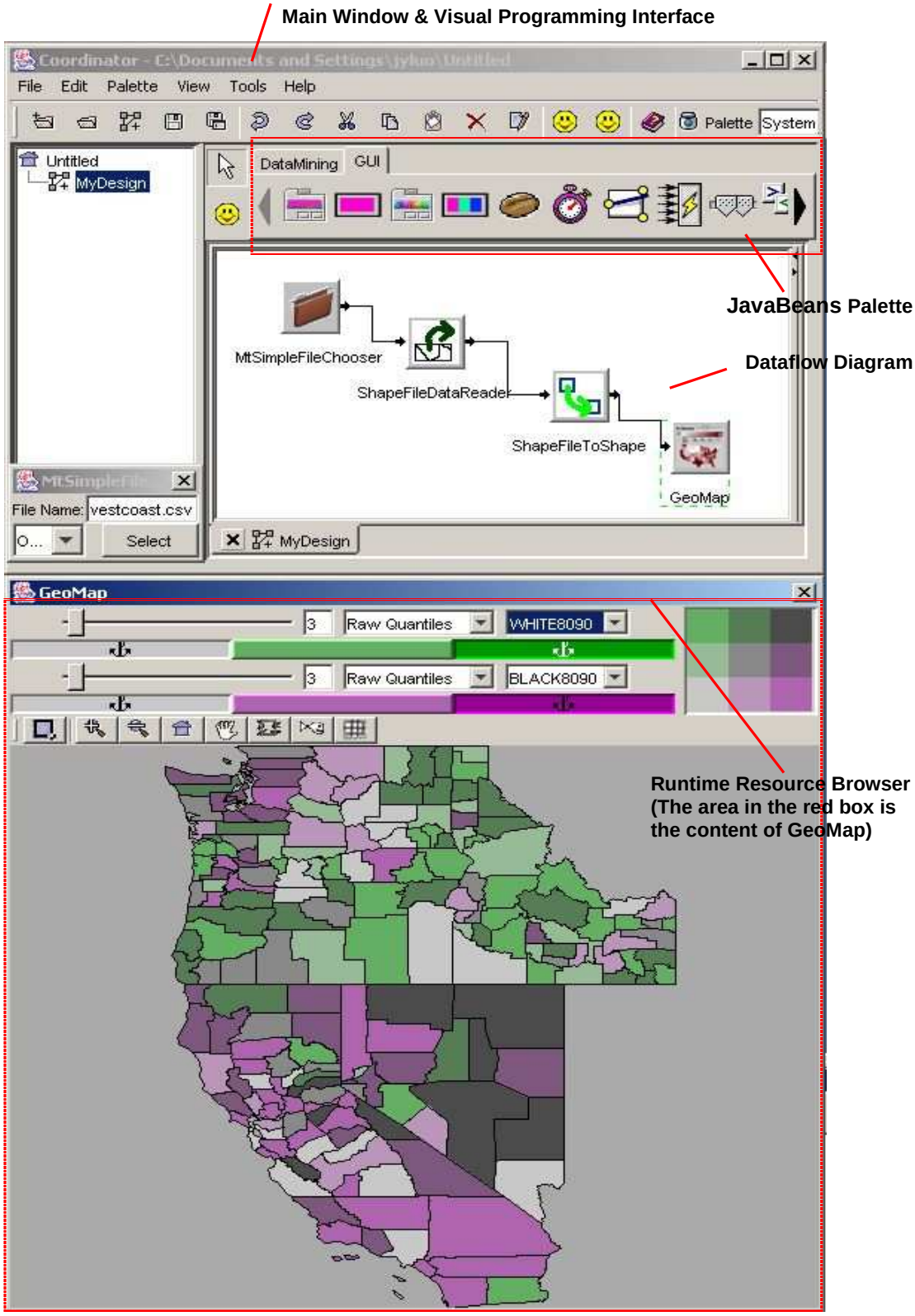


Figure 6.2 The basic user interface of the reference implementation, in which a simple mapping dataflow is created to display county-level demographics of the western U.S..

The system supports both manual and (semi-) automated construction of dataflow applications. In the manual mode, the system bypasses the semantic layer and analysts can create dataflow applications in a similar fashion to GeoVISTA *Studio*, i.e. by selecting the components from the JavaBeans palette, dropping them in the visual programming window, and wiring them together one-by-one. As shown in figure 6.3, users manually compose dataflow applications through the visual programming interface, where the created applications are displayed to users as dataflow diagrams and passed to the execution engine as runtime dataflow specifications defined in the semantic model. Then the execution engine retrieves components from the software module repository, instantiates them into the runtime formats required by the underlying software inter-operation platform, and executes the dataflow application. Finally, the output of execution is collected by the execution engine and passed back to the runtime resource browser, which presents the results of the dataflow application in GUI panels accordingly (e.g. the map panel in figure 6.2).

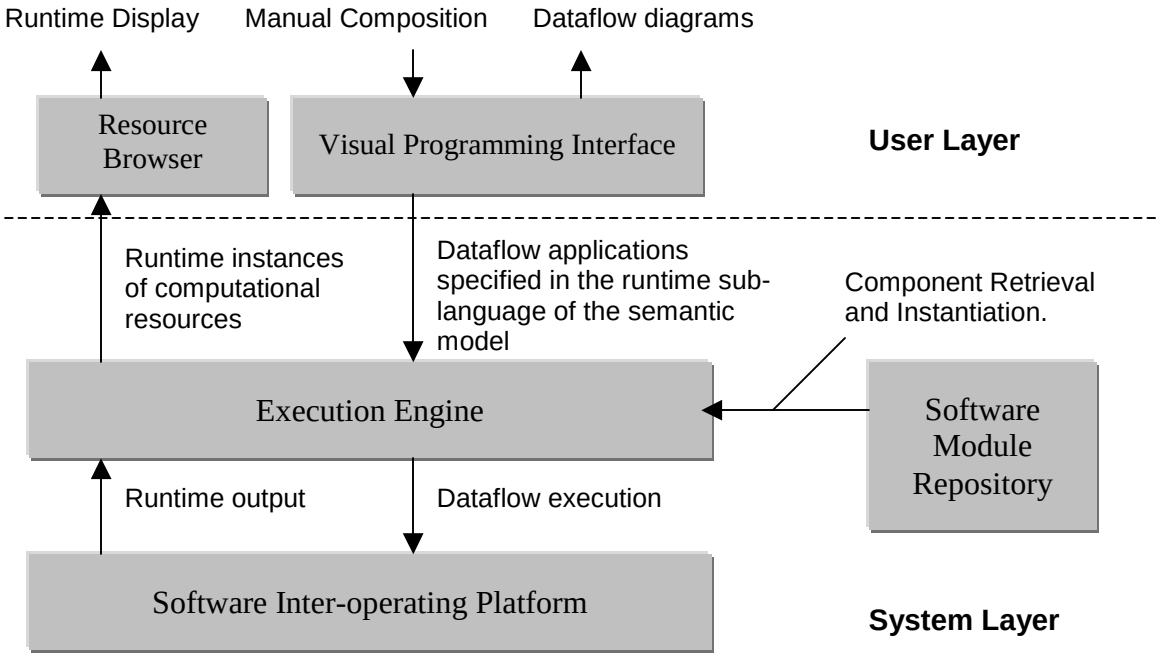


Figure 6.3 Manual problem solving under the reference implementation.

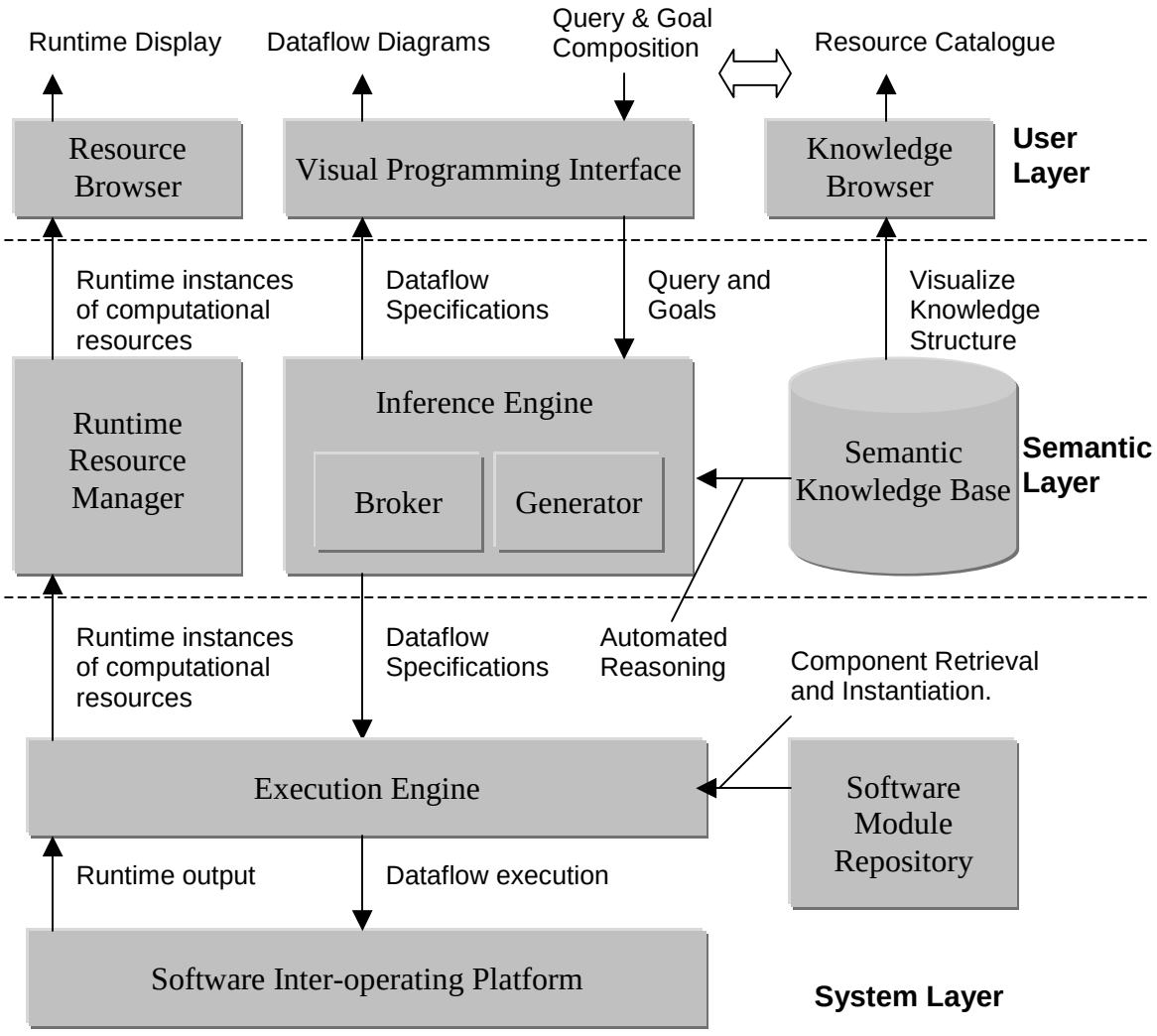


Figure 6.4 Automated problem solving under the reference implementation.

In the (semi-) automatic mode, users can interact with different wizards integrated with the visual programming interface, which guide users to explore and retrieve resources from the software module repository according to their semantic descriptors (through the knowledge browser embedded in the wizards), and activate the underlying semantic inference engine to construct the necessary dataflow automatically. Figure 6.4 shows the general process of automated problem solving under the reference implementation. So, instead of directly creating dataflow applications by hand, users compose high-level queries and goals using the semantic descriptors stored in the

knowledgebase. The query and goals are then submitted to the semantic inference engine, which reasons with the knowledgebase to generate valid dataflows and synthesize desirable applications. Specifications of the resulted dataflows, which are defined in terms of the proposed semantic model, are passed back to the visual programming interface to generate dataflow diagrams and sent to the execution engine to run. The rest of the behaviors of the system are basically the same as the manual mode, except that the runtime outputs from the execution engine are first passed to the runtime resource manager, which updates the information about the connections between runtime instances and their semantic descriptors in the knowledgebase, and then dispatches outputs to the resource browser.

As one can see from figure 6.4, the semantic inference engine contains two key components:

- *The Semantic Broker* is a query engine that retrieves resources according to their high-level semantic descriptions.
- *The Semantic Generator* is an automated planner that generates dataflow diagrams (and thus applications) according to high-level requirements.

The implementations of the semantic broker and generator are based on the contents of chapter three and chapter four.

6.3 Semantically Oriented Geographical Problem Solving

This section describes the usage of the reference implementation from the user's point of view. The main purpose is to show how the semantic inference engine can help users overcome the semantic problems that hamper geographical problem solving in an open, heterogeneous environment, including the problems of resource articulation, resource discovery, and solution generation.

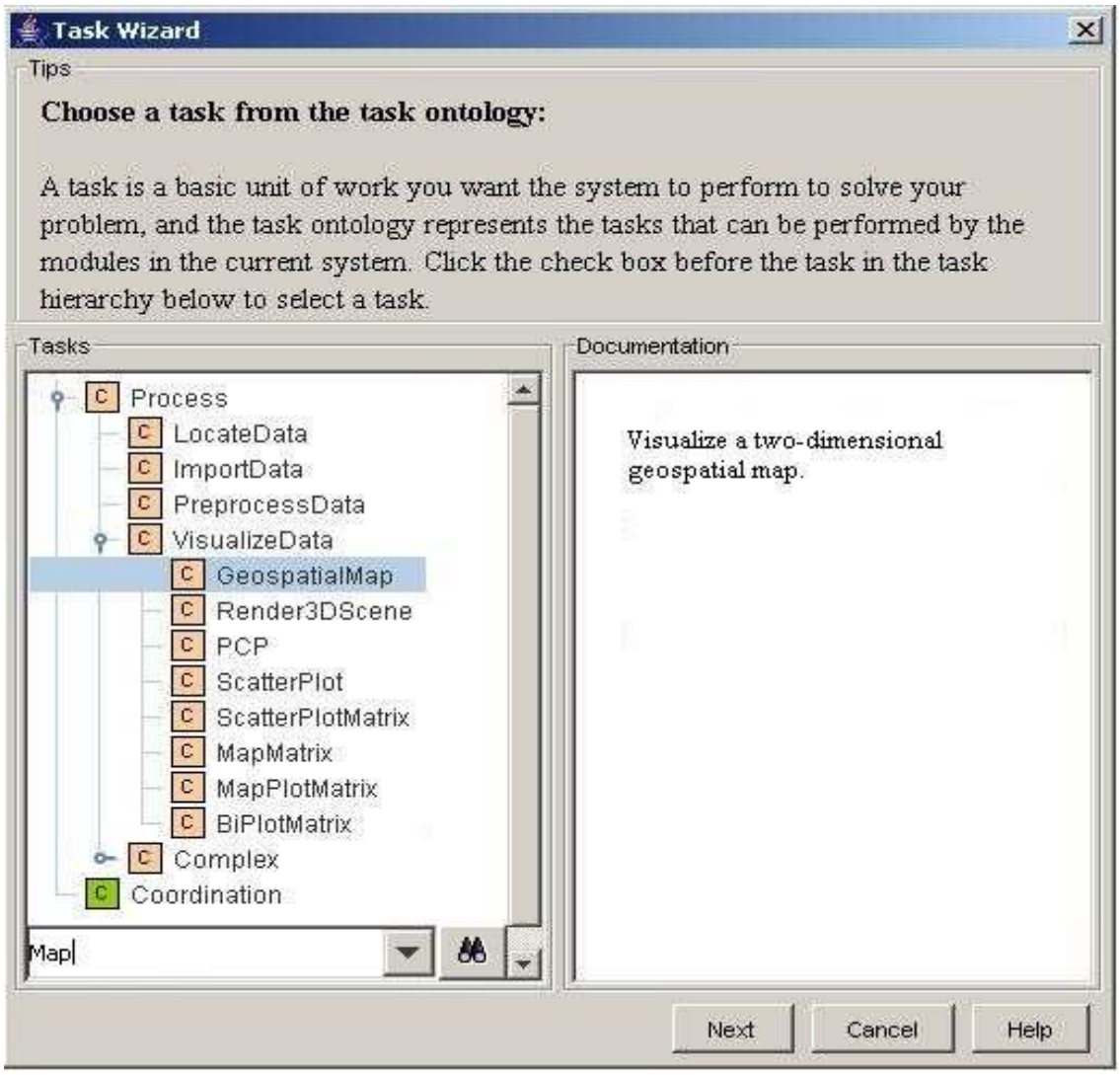


Figure 6.5 The knowledge browser in a task selection wizard, which shows the category of tasks currently in the semantic knowledge base.

6.3.1 Resource Catalogue

The reference implementation eases the articulation of heterogeneous resources with the resource catalogue service provided by the system. In software systems such as GeoVISTA *Studio*, heterogeneous resources are articulated through their built-in representations, which are usually decided by the implementation. For instance, the palette in figure 6.2 (which is also a feature in GeoVISTA *Studio*) uses the icons shipped with the JavaBeans components to represent resources. As we have discussed in chapter

three, icons are not an effective and efficient means of heterogeneous resource articulation, especially when the number of resources is large. One can simply look at figure 6.2 to imagine how hard it will be for an unfamiliar user to "guess" the exact meanings implied by those icons.

In the reference implementation, heterogeneous resources are organized according to their proxy representations rather than the native representations defined by the implementation. This allows users to browse, interpret, and access resources through shared concepts and relations, which can then be linked back to low-level system and runtime structures via proxy representations in a user-transparent way. Figure 6.5 shows a knowledge browser that can be used to navigate the tasks supported by the JavaBeans components in the PSE's software module repository. The content of the knowledge browser is completely independent from the component implementation because the system can employ any user-preferred task ontologies to build the proxy semantics. Once a user selects a task concept in the knowledge browser, the semantic broker of the system can automatically retrieve the component associated with the concept in a relevant proxy representation. In summary, with the knowledge browser as the main resource exploration and selection interface, users no longer need to interact with the fixed representations provided by software developers, but can take advantage of the well-organized high-level concepts provided by proxy representations.

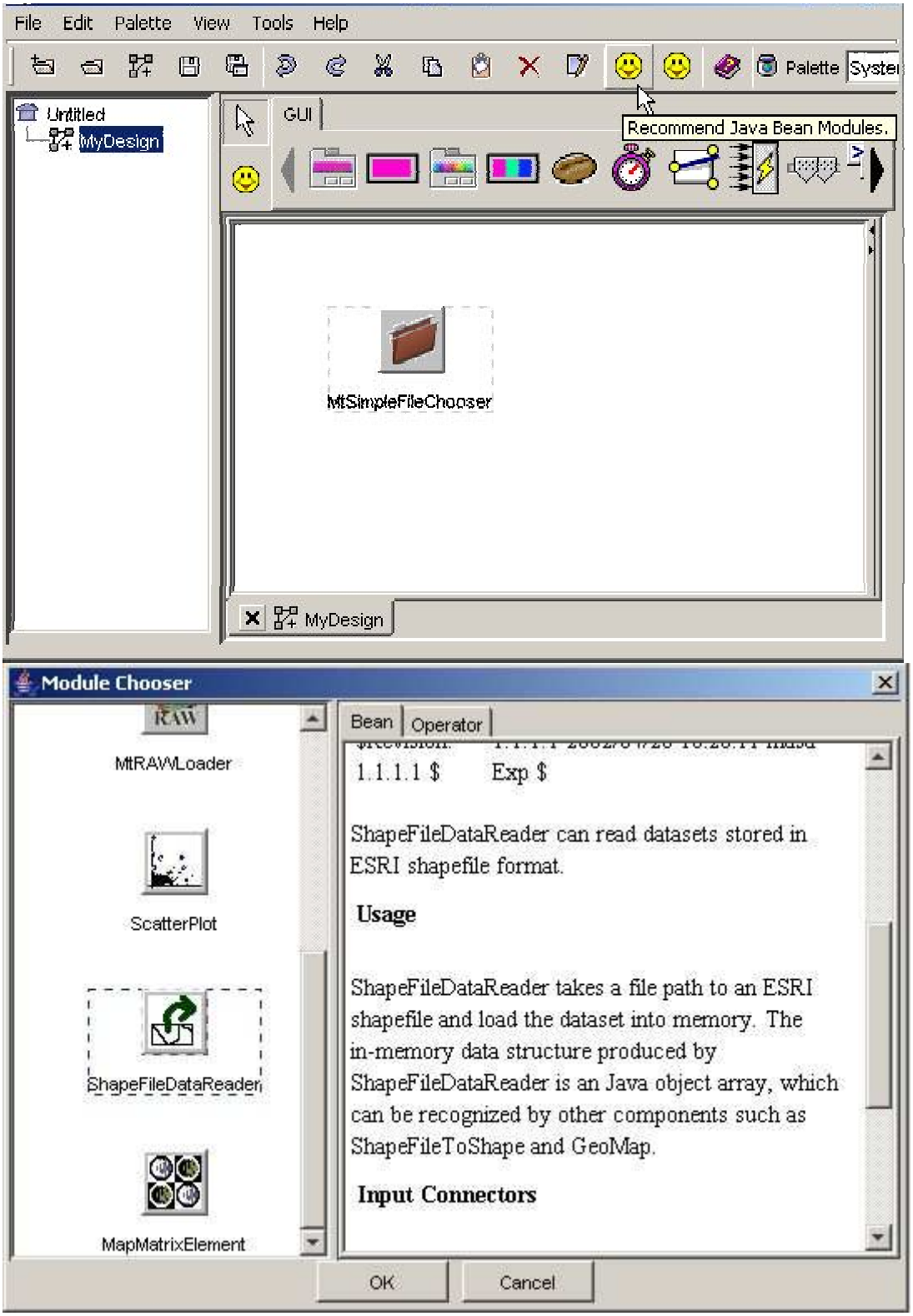


Figure 6.6 Components recommended by the system that can be connected with an MtSimpleFileChooser component.

6.3.2 Resource Query

Concept-based resource discovery can be supported by standard ontological queries offered by the knowledge browser. For instance, in figure 6.5, one can quickly find a task concept using the search box in the knowledge browser, and the semantic broker can link the result back to the corresponding component(s) in a seamless manner. On the other hand, a context-based query can be achieved using the interactive wizard provided by the system, which gathers necessary information from the application context, submits it to the semantic broker, retrieves resources from the results, and passes them back to the user.

Figure 6.6 shows an example of tool recommendation in a context-based query. Imagine that an analyst starts problem solving by selecting a data file, which, as shown in the figure, can be opened by the MtSimpleFileChooser component. Since MtSimpleFileChooser is essentially a GUI component that allows users to select arbitrary files, it is a reasonable starting point for most applications, especially when the user has a data file to process. Now assume the analyst is unsure about what to do next, because he / she is unfamiliar with the components in the system and has little idea about which component is capable of processing the data. This is not unusual considering the fact that the components in the system are provided by different sources, and may change at any time. As shown in the picture, in the reference implementation, users can click the tool recommendation button in the toolbar of the visual programming interface and ask the system to recommend the components that can be connected with MtSimpleFileChooser. The semantic broker will then query the knowledge base to retrieve all components that satisfy such a criterion, and list them (and their documentations if there are any) in the result dialog box for users to choose from. Once the analyst selects a module and clicks the “OK” button, the system will link the selected component (i.e. ShapeFileDataReader) with MtSimpleFileChooser and automatically build the input / output data connections

between both components according to the result of the query. The output of the context-based query is shown in figure 6.7.

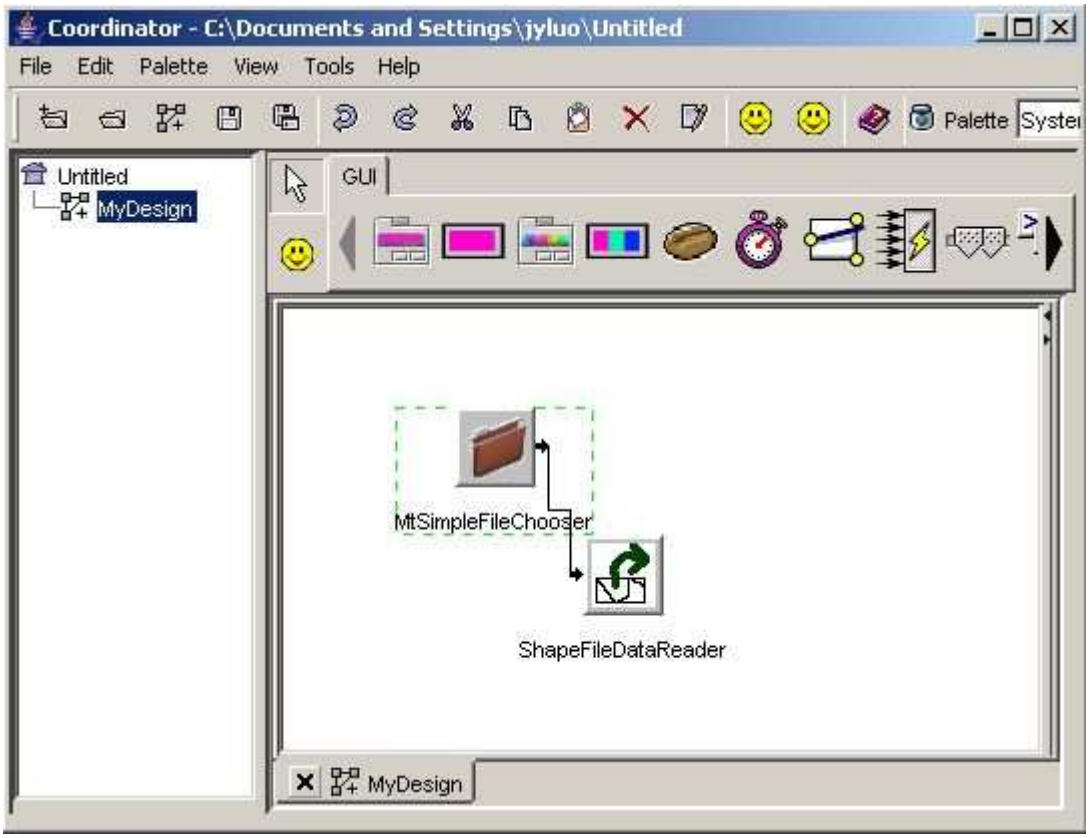


Figure 6.7 The system connects the selected module (ShapeFileDataReader) with MtSimpleFileChooser automatically.

6.3.3 Automated Problem Solving

As discussed in previous chapters, one of the main goals of a Semantic Geospatial PSE is to automate problem solving in different situations, including those requiring incremental problem solving, prototype refinement, and task decomposition. In the reference implementation, such a goal is achieved with various dialogue boxes and user wizards integrated with the visual programming interface, through which users can utilize the automated planner implemented in the semantic inference engine to generate desired dataflow applications in an automated and flexible way.

6.3.3.1 Incremental Problem Solving

The context-based resource discovery tool mentioned in previous discussions can be employed to support incremental problem solving. For instance, suppose the example depicted in figure 6.6 is the starting point for an analyst to explore the cancer distribution of the Appalachian region (using the dataset we have shown in figure 3.1). Due to the exploratory nature of the analysis, the user has little prior knowledge about what patterns to find and wants to experiment with different kinds of visualization tools. Using the tool recommendation wizard depicted in figure 6.6, the analyst can incrementally extend the dataflow application by appending newly selected visualization tools step-by-step, and gradually find the desired solution by evaluating their performance individually. Figures 6.8 and 6.9 show how the dataflow depicted in figure 6.7 can be further extended. At each step, the tool recommendation wizard is invoked to find the components that can be added next, and eventually a simple mapping application can be created.

Incremental problem solving is a good approach for exploratory analysis, as users can easily go back to earlier steps to experiment with alternative solutions. Figure 6.10 shows a scenario where the analyst is attempting to find other components to visualize the same data. In this case, the user went back to the step after ShapeFileDataReader (i.e. after the dataset is loaded) to look for alternative visualization tools. As shown in the example, suppose the user wanted to evaluate the effectiveness of the Parallel Coordinate Plot (PCP), hoping the PCP might reveal interesting patterns between multiple data attributes, which could be difficult to identify in a 2D map. After the user had selected the PCP and clicked the OK button, the PCP component was added to the application (figure 6.12). The resulting visualization is displayed in figure 6.11, where the GeoMap component shows the spatial distribution of selected attributes and the PCP component plots the inter-linkages among different attributes.

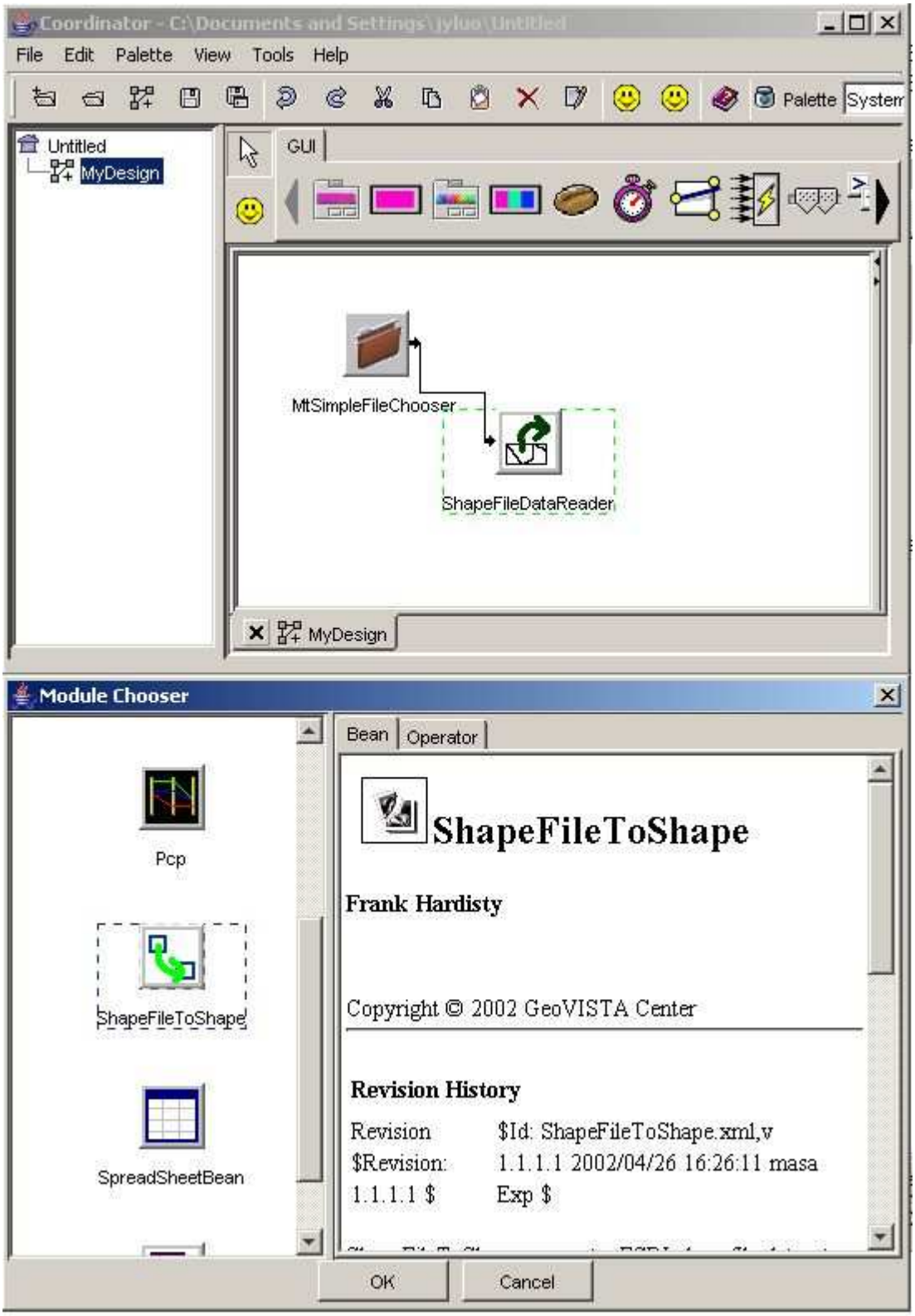


Figure 6.8 Components that can be connected after ShapeFileDataReader, with the component ShapeFileToShape being currently selected.

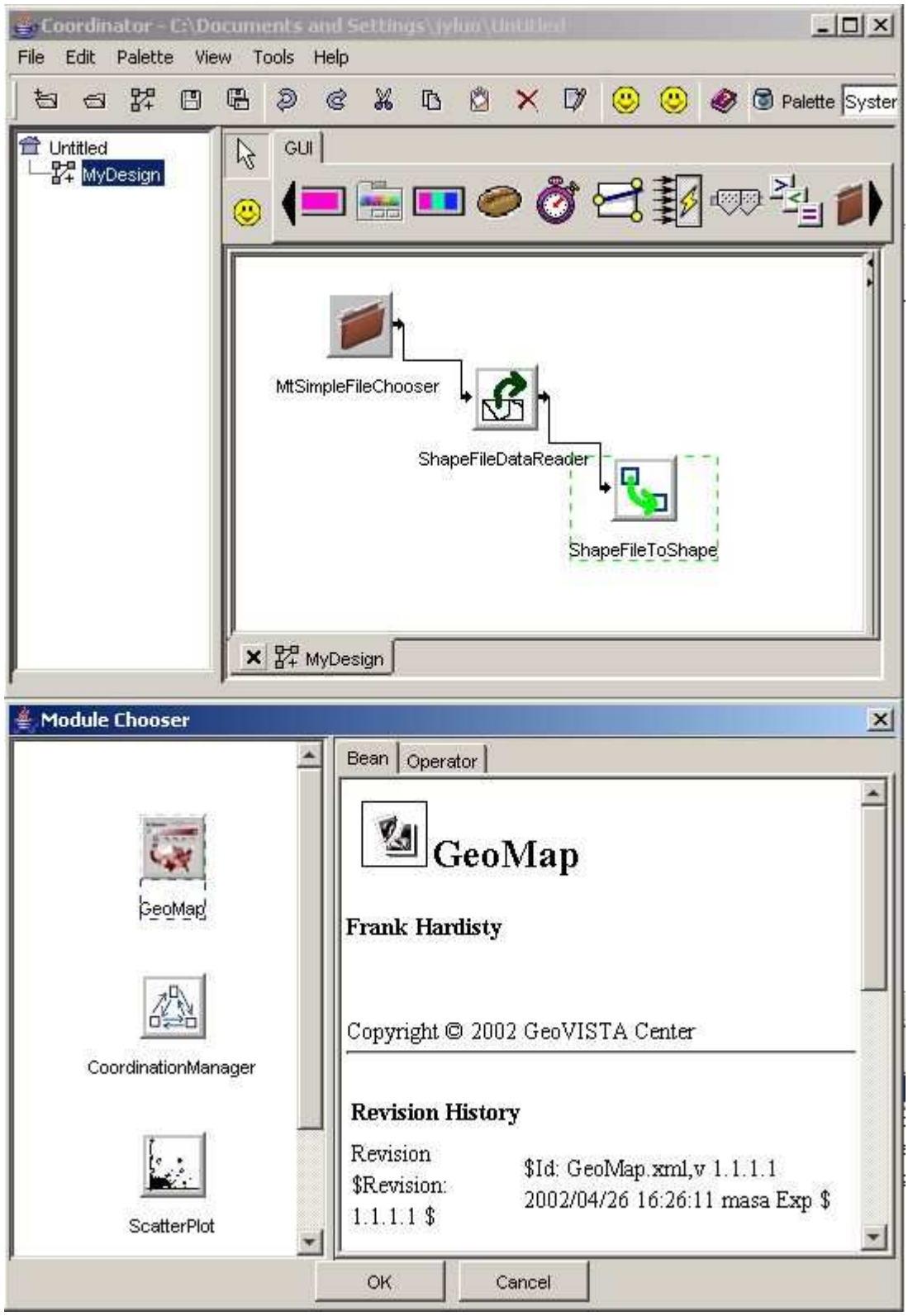


Figure 6.9 Components that can be connected after ShapeFileToShape, with the component GeoMap being currently selected.

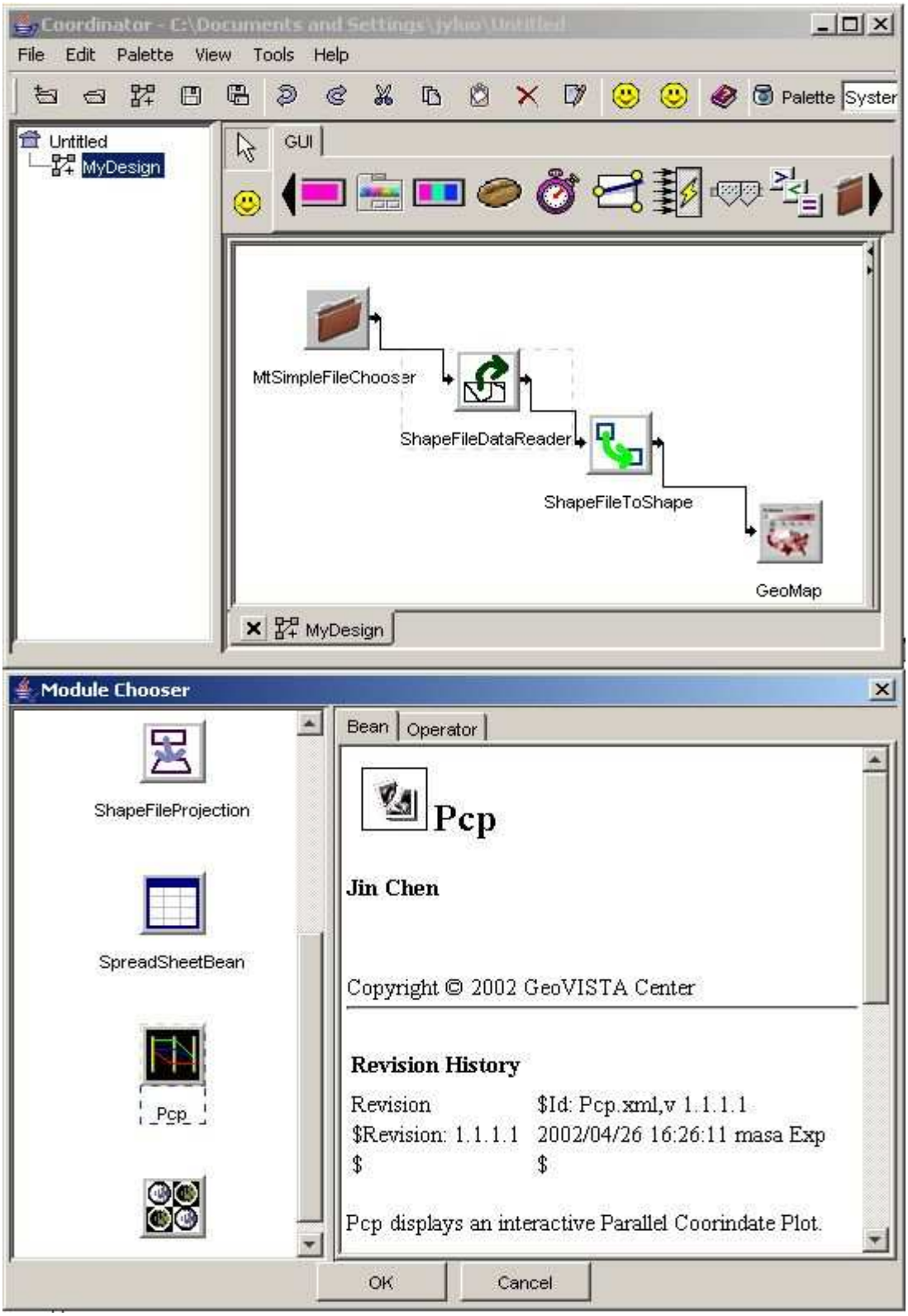


Figure 6.10 Components that can be connected after ShapeFileDataReader, with the component Pcp being currently selected.

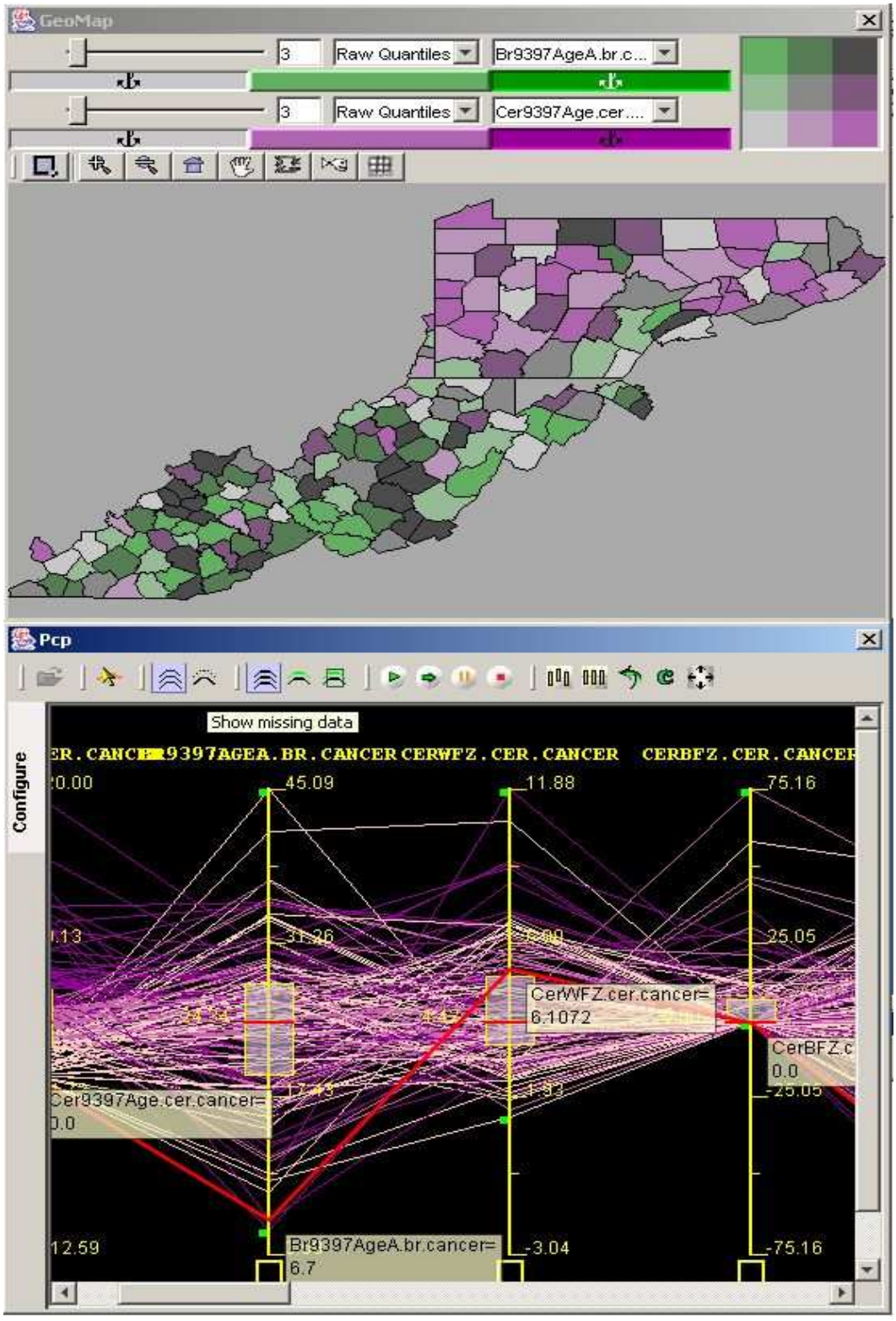


Figure 6.11 Visualizing the county demographics of the Appalachian region using GeoMap and Pcp.

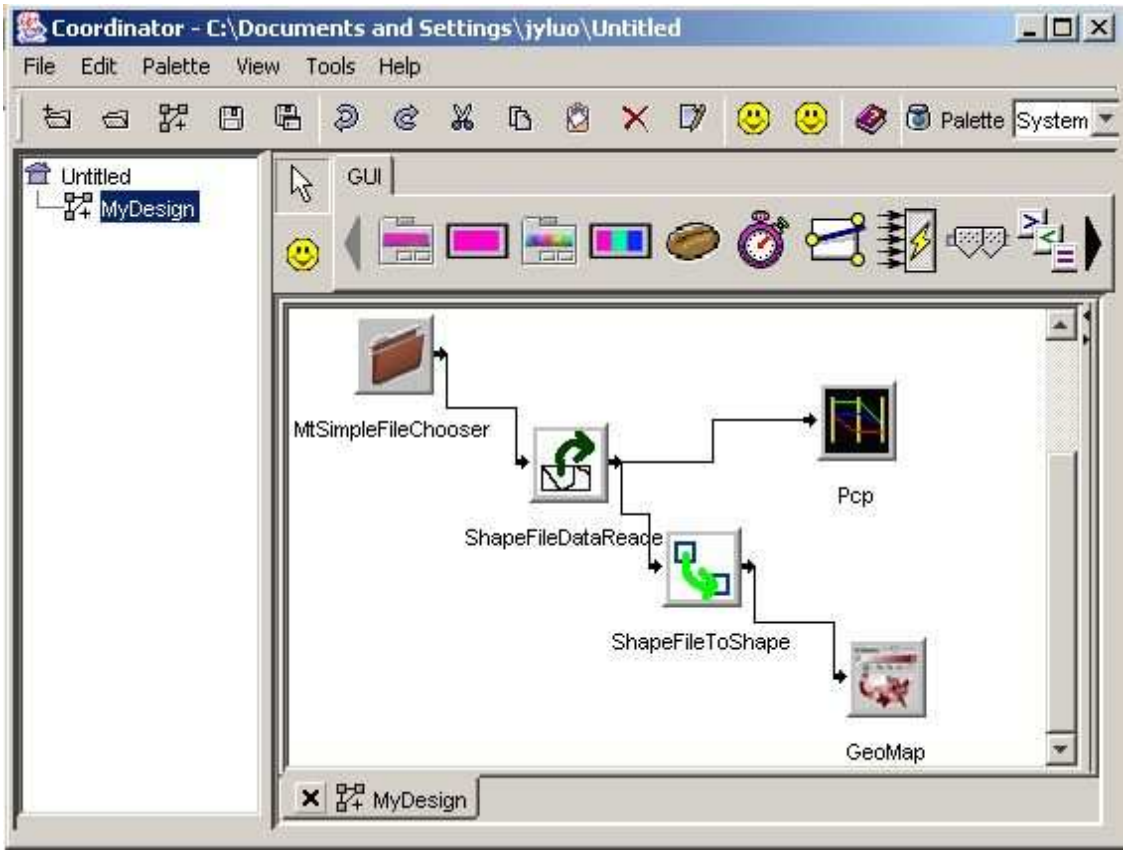


Figure 6.12 Dataflow diagram for the visualization application displayed in figure 6.11.

While one can build a fairly large application step-by-step using the tool recommendation wizard, it is important to understand the exploratory nature of incremental problem solving. Incremental problem solving is usually a more suitable problem solving method for users with little prior knowledge about what the final application should be like, and therefore users are provided with all possible options, with which they can select or experiment based on their own opinions. In situations where users have deeper understandings about the nature of the targeted problem, incremental problem solving may not be the best approach.

6.3.3.2 Solution Refinement

Solution refinement seeks to refine an incomplete or partially specified solution into a desired working application. Users normally hold partial knowledge about the final solution, perhaps knowing some (but not all) of the key components that should be used, and the inference engine can generate the "missing parts" of the application automatically. Generally speaking, there are two different types of solution refinement. The first type aims to extend a partially specified dataflow into a fully functioning application. This is commonplace in real world analysis because in many cases an analyst may have decided to use a particular set of components and functionalities, but does not understand all the details to connect them. For example, figure 6.13 describes a scenario where the user has created an MtSimpleFileChooser node and a GeoMap node in an incomplete dataflow. As these two components are the most important ones for file selection and geospatial mapping, the user may have already learned what they can do from the components' documentations, the system's resource catalogue, or past experience. However, it is unlikely that the user can easily and clearly remember the "less critical" components such as ShapeFileDataReader or ShapeFileToShape, since those data importing and data conversion tools are normally peripheral (although indispensable) to an analysit's major interest. Fortunately, solution refinement provides a user-transparent support for such a situation. As shown in the picture, the user can drag the output connector (the departing arrow) of MtSimpleFileChooser and drop it onto the input connector (the incoming arrow) of GeoMap, which activates the automated planner to find viable connections between those two components using the partial-order planning algorithm. Once the connected path is identified, the system automatically generates all the missing nodes and linkages to create a functioning application. The entire procedure is implemented in an automated and user transparent manner, and analysts do not need to click any button to explicitly activate the planner.

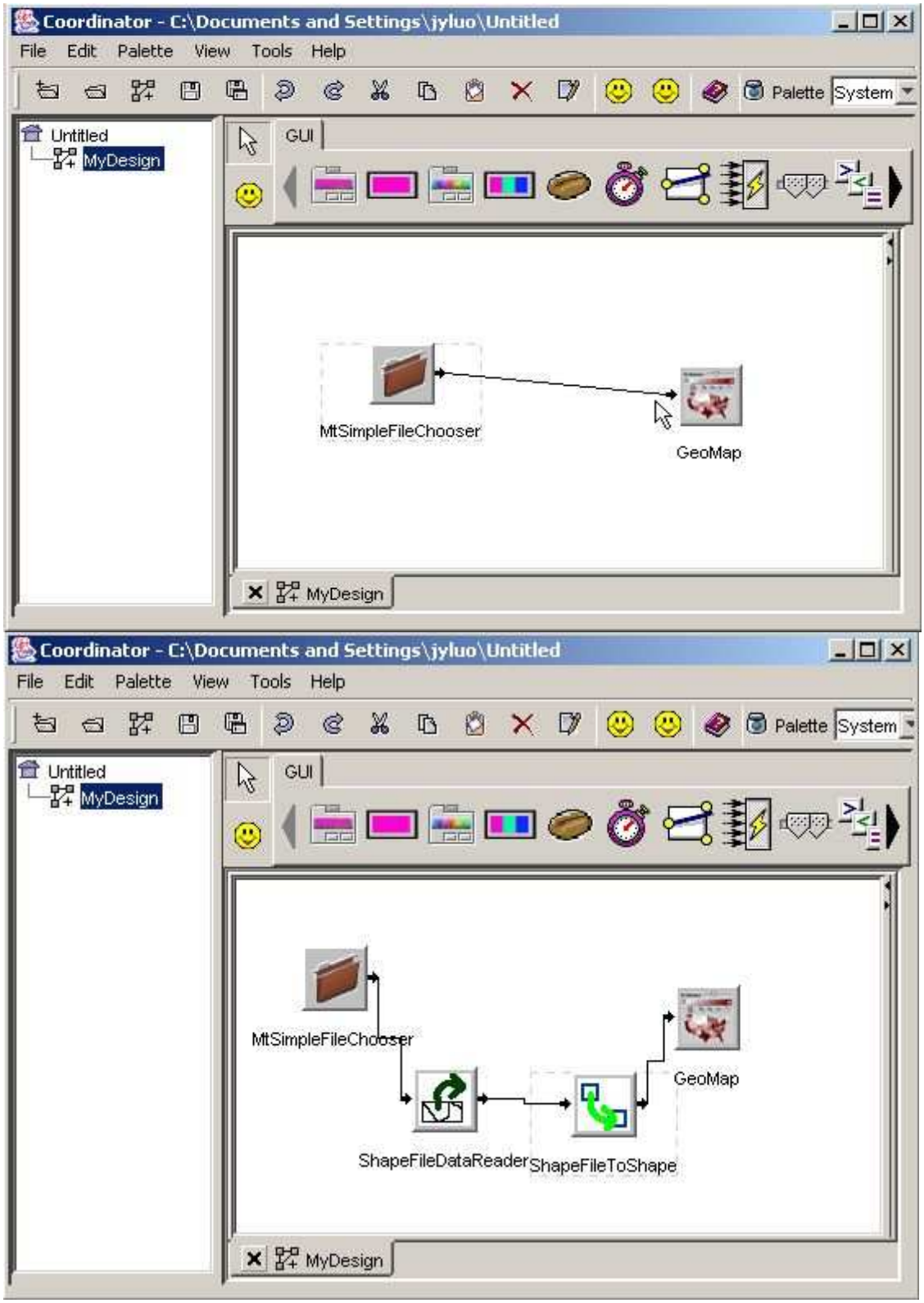


Figure 6.13 The system automatically builds all the necessary connections between MtSimpleFileChooser and GeoMap.

The second type of solution refinement aims to generate a dataflow application according to the user's high-level expectations. In this case, the user may know nothing about which components to use, but have a general idea about what task the final application should perform. For example, suppose a user wants to display a 3D scene stored in a USGS DEM file. In the reference implementation, this task can be accomplished using the Java3D beans stored in the system module repository. However, as Java3D is a highly specialized technology, it is unlikely (and maybe unnecessary) for a geospatial analyst to master the details of Java3D models and understand how to connect Java3D beans together. Under this system, fortunately, users can simply use the knowledge browser, select the "Visualize3DScene" task (Figure 6.14), and let the automated planner generate the dataflow application (Figure 6.15).

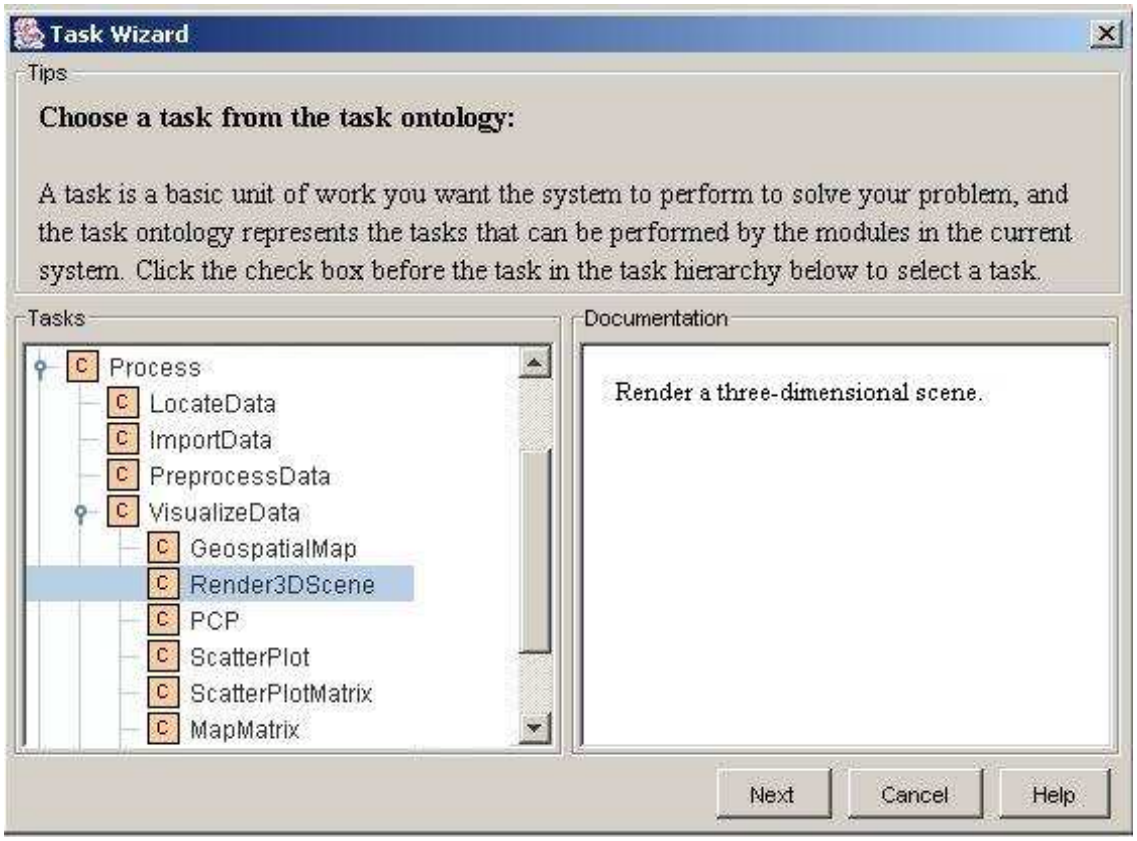


Figure 6.14 Select the task of "Render3Dscene" and the system will automatically build the Java3D dataflow. Users do not need to understand the details of Java3D model.

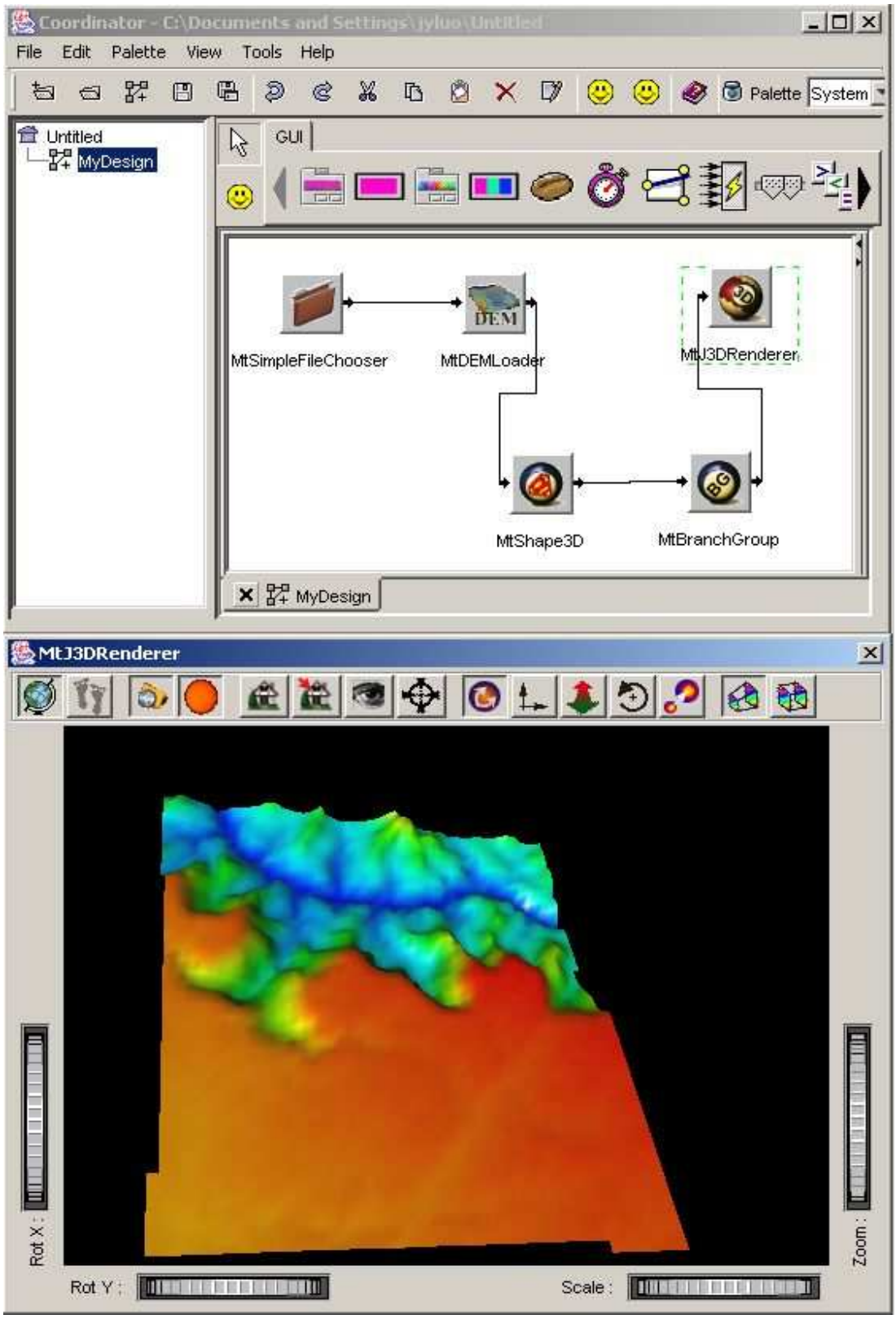


Figure 6.15 Visualize the 3D scene stored in a USGS DEM file, where the dataset displayed is for the Grand Canyon region.

In summary, the solution refinement tool facilitates geographical problem solving by converting users' partial knowledge about a problem into a complete dataflow application. For users with some prior knowledge about what components to use, the system can find all the missing parts to complete the final application transparently,; and for users whose knowledge only extends as far as what task they wish to perform, the system can generate the entire application they need from scratch.

6.3.3.3 Task Decomposition

While incremental problem solving and solution refinement methods are quite useful for a wide range of problems, they are not adequate in supporting more complex real world applications. For instance, in many cases a real world problem may require a large number of components, which makes incremental problem solving extremely inefficient and thus impractical. Furthermore, a complex problem can possibly contain multiple stages of problem solving with different requirements, which make it impossible to specify the user task as a single goal. Figure 6.16 recalls the geographical knowledge discovery application described in chapter four, which is an example of such a complex geographical problem-solving scenario. As we can see from the picture, it is laborious to construct the entire application step-by-step, and moreover, it is not easy to clearly specify the goal of the application because it contains diverse functional parts (e.g. those for data importing, mapping, PCP, coordination, and machine learning) with different goals. As discussed in chapter four, such a complex problem can be solved by a hierarchical task planning method that decomposes the entire data-mining task into smaller subtasks recursively until each subtask can be accomplished by an atomic software component. The structures of task decomposition are stored as reusable task methods. When users select a complex task as the problem goal, the automated planner in the inference engine can use the stored task methods to decompose the task and generate complex solutions.

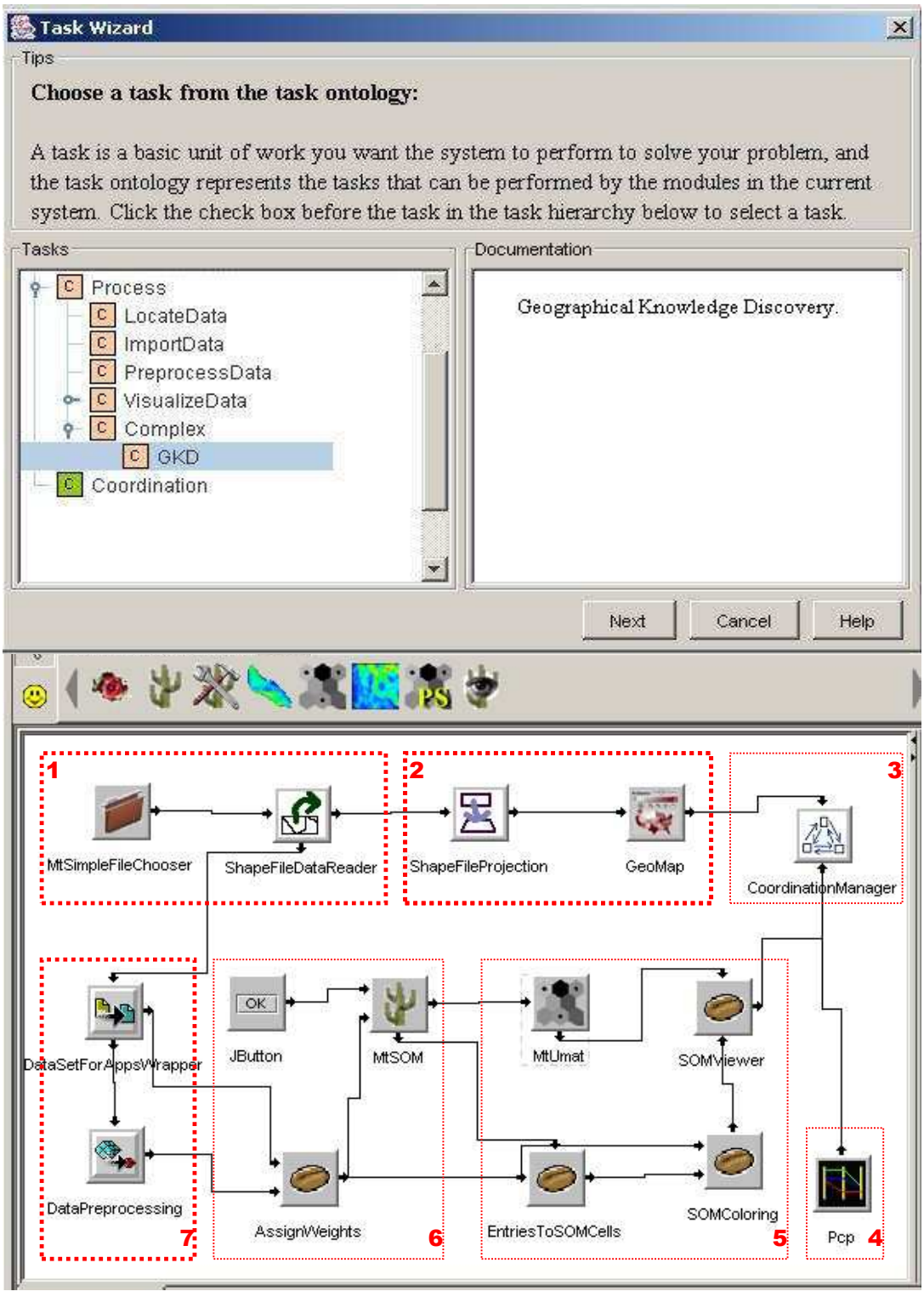


Figure 6.16 Generating a complex geographical knowledge discovery application by hierarchical task planning. The red boxes and numbers are not parts of the GUI. Each red box corresponds to a sub-task, which contains a fragment of component composition stored as a task method in the knowledgebase.

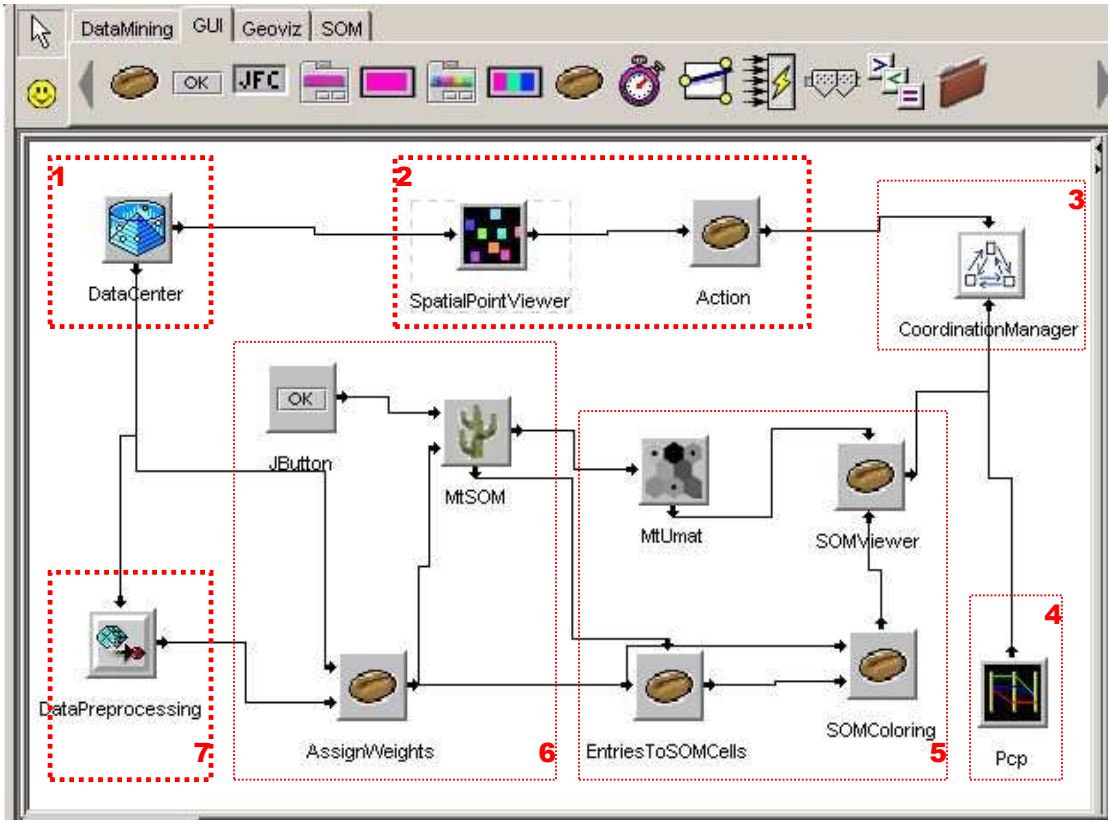


Figure 6.17 A geographical knowledge discovery application that handles point datasets. Boxes with thicker boundaries represent sub-tasks that have changed.

Another important feature of hierarchical task planning is the reusability of task methods. One can regard a task method as a script of abstract sub-tasks, which can be eventually decomposed into different concrete dataflows according to different problem settings. For example, a known weakness of the application in figure 6.16 is that it cannot handle a point dataset, because the GeoMap component only supports polygon data. This limitation is not unusual in a heterogeneous environment because users normally have no control over the capabilities of components developed by other parties. Consequently, if we need to analyze a point dataset (e.g. data of cities or towns instead of states and counties), we must replace GeoMap and perhaps other related components (for example, ShapeFileProjection and ShapeFileDataReader). Without reusable task

methods, we have to regenerate the entire application from scratch. Using a hierarchical task planner, fortunately, this problem is simply equivalent to replacing the corresponding sub-tasks with components that can handle point data (displayed as thicker boxes in figure 6.16). Figure 6.17 displays a geographical knowledge discovery application that handles point datasets, which reuses the general task decomposition structure of figure 6.16. It replaces the tasks numbered as 1, 2, and 7 with new software components:

- For task 1, a DataCenter component replaces the MtSimpleFileChooser and ShapeFileDataReader components to provide the necessary data importing support for point data;
- For task 2, a SpatialPointViewer component specially engineered to visualize geospatial points replaces GeoMap. Unlike GeoMap, SpatialPointViewer itself cannot be connected with CoordinationManager, and therefore, a wrapper component (Action) is introduced to bridge them.
- For task 3, the DataSetForAppWrapper is no longer necessary because now the outputs of DataCenter can be directly used by the DataPreprocessing component (this is because they are actually developed for the same project).

From a task decomposition point of view, all these changes show that the same tasks (e.g. data importing, mapping, data preprocessing) can be performed in different ways, and the hierarchical task planner can reuse the task structure defined by the same tasks, but instantiate them differently according to different problem settings.

6.4 Summary

A Semantic Geospatial PSE can be developed based on the three-layer architecture that includes a software-interoperation layer, a semantic layer, and a user layer. While the software-interoperation and user layers can be implemented based on existing software engineering technologies, the semantic layer can be built according to

the semantic model described in this thesis. The main enhancement that the semantic layer has made to the system is the structured knowledgebase and the automated inference engine, which equips users with a semantically organized resource catalogue, a flexible resource discovery tool, and powerful automated problem solving support. With the Semantic Geospatial PSE, analysts can easily identify relevant resources, quickly locate useful tools, and automatically or semi-automatically build desired applications. This allows them to remain focused on the nature of their geographical problems rather than the details of how to perform the necessary computation. Consequently, the time and effort involved in geographical problem solving are significantly reduced.

Chapter 7

CONCLUSIONS AND FUTURE RESEARCH

7.1 Introduction

Geographical research prior to the digital computing era has often been noted as cumbersome and data poor, where most researchers' time was spent on collecting, cleaning, and recording data. When modern computers and sensing technologies were introduced to the geospatial communities in 1960s, they quickly transformed geography into a data rich field, where unprecedented amounts of geospatial data were made available to analysts. One of the main concerns since then has been to develop adequate analytical power to fully analyze geospatial data in digital computers (Openshaw 1992). During the past decade, however, the technical landscape of GISci and geographical analysis has been further transformed into an environment featuring an abundance of both digital data and analytical functionalities, thanks to the rapid development of geospatial data handling and spatial analysis technologies. While the collection of data and the development of new analytical tools are still critical parts of geographical research, the problem of how to locate, combine, and utilize existing computational resources becomes equivalently important to the success of geographical problem solving. Furthermore, recent standardization initiatives in the geospatial information community have created increasing demands for sharing, repurposing, and integrating heterogeneous GI technologies, which requires analysts to be able to efficiently combine different data and tools into meaningful applications. The Semantic Geospatial PSE presented in this thesis is a technology responding to such a need.

Compared with many other studies in GISci and geographical analysis, this thesis has been focused on computational infrastructure rather than specific data or analytical

methods. In other words, instead of studying certain geographical phenomena or examining a particular kind of analysis, this study tackles the semantic problems that geographers may encounter during the process of computer-assisted problem solving, and aims to develop the necessary enabling technology to solve these problems. Built on top of the prevailing data sharing and software inter-operation standards in the geospatial information community, the Semantic Geospatial PSE introduces a framework of semantic modeling, representation, and inference, which bridges high-level geographical understanding, low level software implementation details, and runtime logics. Based on that, the Semantic Geospatial PSE enables geographical analysts to better identify useful resources, discover relevant tools, and build desired applications. To conclude this study, the remainder of this chapter first summarizes the main results of the thesis. After that, some interesting issues elicited from the results of this study are discussed, and possible directions of future research are outlined.

7.2 Thesis Summaries and Major Results

As described in the first chapter, this thesis's contributions reside in four areas, specifically, at the system level, the conceptual level, the automation level, and the representation level.

In chapter two, the concept of a Geospatial PSE is proposed by combining the general ideas of scientific PSEs and the specific requirements of geographical problem solving. A Geospatial PSE contains three key elements: an open software framework that integrates heterogeneous data and software functionalities, a visual programming interface that allows users to visually construct high-level geographical applications, and a knowledge-based system that enables users to retrieve, configure, and utilize computational resources in an automated and transparent manner. A Semantic Geospatial PSE is a Geospatial PSE that uses an explicit semantic model of geographical problem solving to design and implement its knowledge-based system. Consequently, the

Semantic Geospatial PSE provides the necessary knowledge-based support and automation to bridge the semantic gap between heterogeneous resources and geographical problem solvers.

Chapter three proposes a semantic model of geographical problem solving according to Peirce's theory of pragmatism. More specifically, the meaning of geographical problem solving is analyzed at three levels: first level semantics address properties of individual resources, second level semantics cover relations, and third level semantics deal with geographical applications. Based on the triadic sign model from semiotic studies, first level semantics is modeled by proxy representations, which are essentially abstract notions of computational resources that flexibly aggregate multiple representations required by different contexts. Second level semantics is modeled by formal logic and model theory, which provide a rigorous framework to represent and manipulate structural relations among resources and offer a computational means to automate resource queries. Third level semantics is analyzed by examining three types of geographical problem solving methods: incremental problem solving, prototype (or solution) refinement, and task decomposition. While every geographical application has its own unique meaning, these methods address three modes of reasoning during the iterative process of geographical problem solving, and thus provide guidelines for the design and implementation of automated problem solvers for geographical analysis.

Chapter four then draws upon theories and techniques from cognitive science and AI to build automated problem solvers that meet the requirements of geographical problem solving. Based on Newell and Simon's cognitive model of human problem solving, two AI planning techniques are utilized and integrated: partial order planning that supports incremental problem solving and solution refinement, and simple task network planning (which is a variant of hierarchical task network planning) that supports task decomposition. The technical details concerning how to convert between the results of the automated planner and the dataflow visual programming model are also discussed.

Chapter five examines how to embody the proposed semantic model (which has been mostly specified in the first order predicate calculus at the conceptual level) into a real world knowledge representation. An knowledge representation language based on W3C (World Wide Web Consortium)'s OWL (Web Ontology Language) is developed to take the advantage of the expressiveness and portability of ontological computing. The correspondence and translation among the semantic model, the automated planner, and the OWL-based language are strictly defined so that a Semantic Geospatial PSE can use the OWL-based language for external knowledge exchange, the first order semantic model for internal knowledgebase implementation, and the cognitive model for automated reasoning.

Finally, chapter six presents a reference implementation to demonstrate the idea of a Semantic Geospatial PSE. The reference implementation can be regarded as an extended version of GeoVISTA *Studio*, which uses the concepts and techniques introduced in this thesis to automate the construction of front-end user applications. With the reference implementation, a geographical analyst does not need to manually browse, select, and connect heterogeneously developed software components in order to use them. Instead, (s)he can browse through and choose from categories of high-level tasks, and the system can automatically generate a fully functioning application. In other words, the technical details about how to meaningfully integrate different components are hidden from end users, and the geographical analyst can then focus on their specific analysis tasks rather than being distracted with tool building.

7.3 Discussions and Directions of Future Research

The research in this thesis produces cross-disciplinary results that overlap geography, software system development, and knowledge representation, which may lead to interesting further studies towards all three areas.

7.3.1 Applications of the Semantic Geospatial PSE

With the computational infrastructure provided in this thesis, one natural follow-on study would be to apply the system to support further geographical projects. For instance, the problem of regional environment vulnerability assessment is an example that the Geospatial PSE can immediately contribute to. Generally speaking, environment vulnerability reflects a given geographical region's ability to sustain hostile events (e.g. natural hazards), which can be assessed by evaluating a set of biophysical and social-economic factors (Canter 1996). During the past several years, environment vulnerability assessment has been studied intensively in the HERO project (Human Environment Relation Observatory), a multi-disciplinary and multi-institution research project based on the collaboration of researchers from four universities (Clark University, Kansas State University, University of Arizona, and Pennsylvania State University). More details about the HERO project can be found at the website of the Penn State HERO observatory http://hero.geog.psu.edu/SRB-HERO/SRB-HERO.html. One of the most important goals of this project is to develop a protocol that describes the overall process of vulnerability assessment, which provides a feasible "pathway" or procedure for environment decision makers to analyze the local consequences of environment impacts, and defines a standardized framework whereby environment managers of different regions can share and monitor their results.

The Semantic Geospatial PSE provides an ideal platform to implement the HERO vulnerability assessment protocol. According to published HERO documents (Hero 2003), vulnerability assessment can be standardized as a procedure that analyzes relevant physical and social economic factors to determine a region's coping ability to given natural hazards. From a semantic point of view, the concepts of natural hazards and the factors that impact coping ability can both be represented as ontologies. The assessment procedure can be easily encoded as hierarchical tasks, with the most atomic ones

engineered as specialized software functionalities (e.g. those calculating the values of a given factor). Combining the concepts of natural hazards, the ontologies of coping factors, and the task knowledge about assessment procedure under the semantic model proposed in this thesis, we can obtain a collection of reusable semantics that can be employed to share assessment procedures for different regions and automate the computation of region-specific vulnerability indicators.

7.3.2 Knowledge Acquisition and Visualization

Knowledge acquisition is the process of obtaining structured knowledge from unstructured, usually informal knowledge sources such as text documents, surveyed data, subjective thinking, expertise, and experience (Scott, Clayton et al. 1991). While the research in this thesis solves the semantic aspects of geographical problem solving at the infrastructure level, the next biggest issue for geographical analysts concerns knowledge acquisition, since capturing problem solving semantics is a prerequisite for the Semantic Geospatial PSE.

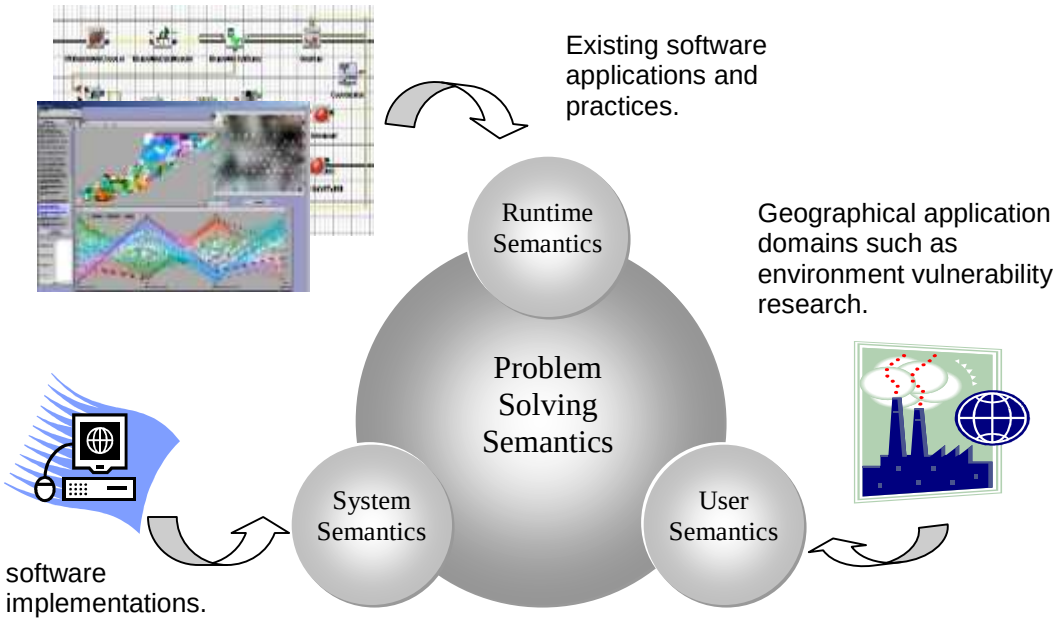


Figure 7.1 The acquisition of problem solving semantics.

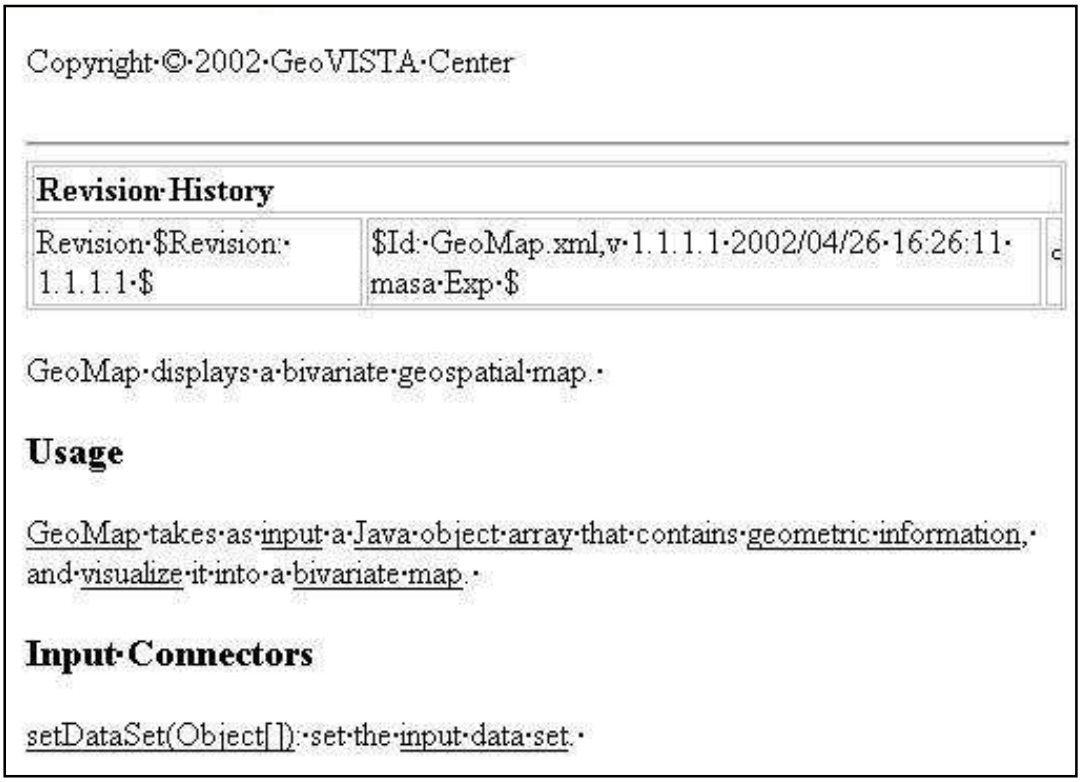
Figure 7.1 depicts the general process of knowledge acquisition as it applies here. As shown in the diagram, according to the proposed semantic model, building problem solving semantics for computational resources requires the integration of three kinds of knowledge:

- User semantics can be collected from user application domains, including high-level concepts and tasks related to a targeted class of problems such as environment vulnerability assessment;
- System semantics can be collected from software implementations, e.g. detailed specifications of data structures, software interfaces, and component implementation.
- Runtime semantics can be collected from existing applications, which may include reusable structures of component composition and task decomposition.

These three kinds of knowledge can then be combined to develop problem solving semantics, including proxy representations of individual resources, formal rules of component composition, as well as the state transition operators and task methods required by automated planning. In the immediate future, two knowledge acquisition methods can be explored and tested for the Semantic Geospatial PSE.

1. Verbal Protocol Analysis. Verbal protocol analysis is the conventional methodology for knowledge engineers to extract and build structured knowledge from informal descriptions (Newell and Simon 1972). Such informal descriptions, normally specified in natural languages, are considered as "verbal protocols" that contain firsthand information about a knowledge domain. Verbal protocol analysis can be accomplished in a similar fashion to other methods of qualitative analysis, where concepts and relations are extracted and generalized from relevant keywords identified in a text document. Verbal protocols can be obtained from different sources, including software documents, published articles, and transcripts of user interviews. Figure 7.2 shows a verbal protocol for the GeoMap component, which is derived from the documentation of the component.

As displayed in the pictures, the keywords that carry the most important information are identified in the document with underscores, and based on them, concepts and relationships can be further created according to the syntax of the selected knowledge representation language. While traditionally these tasks are carried out by knowledge engineers, text mining tools based on natural language processing can be used to extract knowledge from text documents automatically or semi-automatically (Fan, Wallace et al. 2006). It will be an interesting research topic to examine how to obtain useful verbal protocols for geographical problem solving, and study different methods' performance in extracting and creating geographical problem solving semantics (including both manual and automated ones).


Copyright·©·2002·GeoVISTA·Center

Revision·History		
Revision·$Revision:·1.1.1.1·$	$Id:·GeoMap.xml,v·1.1.1.1·2002/04/26·16:26:11·masa·Exp·$	c

GeoMap·displays·a·bivariate·geospatial·map.·

Usage

GeoMap·takes·as·input·a·Java·object·array·that·contains·geometric·information,·and·visualize·it·into·a·bivariate·map.·

Input·Connectors

setDataSet(Object[]):·set·the·input·data·set.·

Figure 7.2 Verbal protocol analysis for the GeoMap component. The verbal protocol is the software document for the component, where relevant keywords are identified by underscores.

2. Interactive Knowledge Acquisition. While verbal protocol analysis provides a systematic method to collect concepts and relationships from text descriptions, it does not address the later stage of knowledge acquisition, where concepts and relationships must be converted into the format of a knowledge representation language. In addition, sometimes it is more convenient to obtain structured knowledge directly from subjective sources without intermediate verbal protocols, because sometimes the text descriptions for a knowledge domain are absent or inaccurate. Therefore, in many cases it is necessary to use an interactive knowledge development tool to build the desired knowledge representation, either based on verbal protocols or directly from human thinking.

Figure 7.3 depicts an example of interactive knowledge acquisition using *ConceptVista*, an ontology development and visualization tool built in the GeoVISTA center (http://www.geovista.psu.edu/ConceptVISTA/index.jsp). *ConceptVista* uses a concept map as the basic visual metaphor to facilitate knowledge creation. A concept map is a directed graph that represents concepts as nodes labeled with keywords and relationships as edges marked with "linking words" (essentially keywords for relationships) (Buzan 1995). In *ConceptVista*, users can visually create a concept map via basic mouse gestures (e.g. drag-and-drop), and the resulting concept map is stored in OWL format. In figure 7.3, the basic concepts of regional environment vulnerability assessment are organized into a class hierarchy, which can be further used to create the use semantics of computational resources. In the near future, how to evaluate the pros and cons of interactive knowledge acquisition tools such as *ConceptVista*, how to build more effective concept visualization interfaces for knowledge acquisition, and how to apply such tools to facilitate the collaboration among different human knowledge communities, are all interesting research questions.

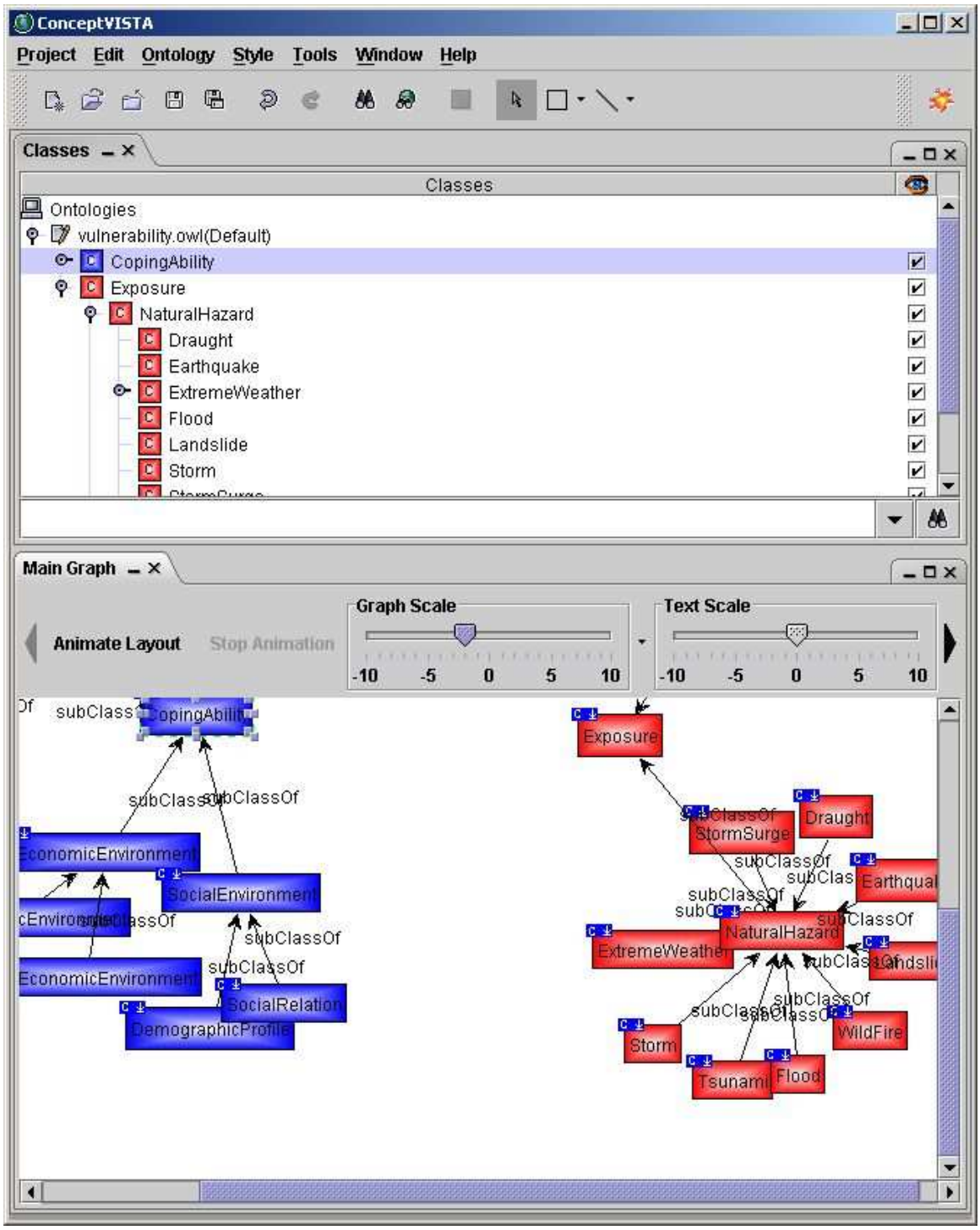


Figure 7.3 Concepts of regional environment vulnerability visualized in *ConceptVista*.

7.3.3 Building Robust, Extensible and Usable Systems

Although the reference implementation introduced in this thesis is adequate in demonstrating the main ideas of a Semantic Geospatial PSE, it is still raw by the standard of a mature software system, lacking the robustness and reliability required by real world applications. In the future, it may be necessary to build a more robust and usable Semantic Geospatial PSE to support real world projects. One natural approach is to extend the current version of GeoVISTA *Studio* with the semantic infrastructure described in this thesis. The resulting system will then take advantage of the reliable execution engine and mature visual programming interface offered by GeoVISTA *Studio*, while utilizing the automation power provided by the semantic inference engine.

Furthermore, a mature Semantic Geospatial PSE must provide extensible interfaces to other software inter-operation standards. As mentioned in previous discussions, both GeoVISTA *Studio* and the reference implementation are based on the JavaBeans component standard. While this standard has been widely adopted in open software communities and a JavaBeans-based PSE can benefit from the platform-independent nature of the Java language (which means the PSE can run seamlessly on different operating systems), from a user's perspective, it cannot directly support the case in which the desired software functionalities are not delivered in Java. For example, software components in ArcGIS, one of the most widely used GIS packages, are engineered in Microsoft's Component Object Model (COM). As mentioned briefly in chapter two, non-JavaBeans software modules can be incorporated by building "wrapping" modules for them. Those wrapping modules, or "wrappers", are specially engineered JavaBeans components that delegate data communication and software operation requests to the wrapped non-JavaBeans modules. As a result, they can work exactly as a JavaBeans component under the Semantic Geospatial PSE but deliver the functionality of the wrapped non-JavaBeans modules. Currently there are plenty of

technologies that support the development of such kinds of wrappers, e.g. the Java Native Interface (JNI) shipped with the standard Java platform (Gordon 1998), which allows Java programs to invoke native modules such as those in ArcGIS, as well as different JavaBeans bridges to other component models (Doherty and Leinecker 1999), which directly map the functionality of a non-JavaBeans module into a JavaBeans wrapper. It is important for a Semantic Geospatial PSE to draw upon those technologies to incorporate useful non-JavaBeans software modules.

Finally, it is critical to thoroughly study the usability of the Semantic Geospatial PSE and build more accessible user interfaces (UI) for the semantic knowledgebase and inference engine. By now the UI design issues of JavaBeans and dataflow-based visual programming have been well studied. However, the question of how to build a better user interface onto the semantic infrastructure of geographical problem solving is a completely new area. Since the main purpose of this thesis is to build the semantic infrastructure rather than develop user interfaces for it, issues related to UI design have not been systematically discussed. In the reference implementation, for example, the semantic infrastructure is only equipped with an *ad hoc* user interface, which employs a simple tree-structure to visualize the high-level categories of resources, and uses several wizards to assist front-end human-computer interaction. In the future, it will be necessary to examine alternative ways to visualize the structured semantics and explore different UI design patterns to facilitate user interaction. For example, how effective will a concept map be in helping geographers locate useful resources? Can data visualization techniques or "data views" be directly associated with the semantic inference engine to expedite goal selection and trigger automation (i.e. instead of asking a user to select a goal, the system determines the goal of the user according to the dataset she / he is visualizing)? Is it possible to use dialogues in natural language (the vocabulary of which is mapped to the ontologies in the knowledgebase) to improve human-computer interaction? All of these are interesting future research questions. The bottom line is,

since the knowledge of geographical problem solving has been structurally captured by the semantic knowledgebase, it is now possible to build different UI views from such explicit structure. Which UI views are most suitable for what kinds of user tasks, nevertheless, must be tested and evaluated by well-designed user case studies and experiments.

7.4 Representing, Sharing, and Problem Solving: Towards a Pragmatic Geography

In his original writings on pragmatism, Charles S, Peirce identified two basic forms of inquiry that are critical to any human studies: the descriptive (or positive) inquiry that examines a targeted class of phenomena (e.g. Physics and Biology), and the normative inquiry that studies meanings, values, justifications, and various ways to reach different goals (examples including Logic and Mathematics) (Peirce 1931-58). While descriptive sciences provide theories, statements, and explanations about the world, normative sciences help us produce, evaluate, share, and apply those theories, statements, and explanations in practices. In geography and GISci, descriptive inquiries have resulted in various theories, analytical methods, and computational models for different geographical processes. However, their articulations, interpretations, and applications could not succeed without a systematic normative framework. The development of a Semantic Geospatial PSE takes some first steps towards such a framework. One of the most distinguishing characteristics of this study is its focus on the general process of geographical problem solving rather than a given geospatial phenomenon. Instead of examining a specific type of geospatial data or a particular kind of analytical model, the Semantic Geospatial PSE defines a normative model to represent, share, and reuse geographical problem solving semantics, which leads to more powerful software systems and more productive geographical inquiries. Although the discussion in this thesis is focused on computational systems, in the foreseeable future, it is possible to extend these

ideas across the geographical discipline, where diversified geographical thoughts can be articulated, represented, shared, and applied meaningfully under a strictly defined, normative framework to solve different kinds of real world problems.

Appendix A

Syntax of the First Order Predicate Calculus

1. Terms

A logical *term* is a *constant*, a *variable* or a *function*. A constant is an element of an enumerable constant symbol set *CON*. A variable is an element of an enumerable variable symbol set *VAR*. In this study we use strings starting with "*?*" to denote variables. An *N-ary* function is constructed in the form $f(t_1, t_2, \ldots, t_n)$, where *f* is a *function name*, $t_1, t_2, \ldots, t_n$ are terms, and *n* is the function's *arity* or *valence*. A function name with a valence is called a *function symbol*, denoted by *f/n*. In addition, common arithmetic functions such as "+", "-", "*", "/" are expressed in conventional way, e.g. 1+2 rather than *+(1,2)*. A function returns a constant value if we replace all its variable terms with constants.

2. Predicates

A *predicate* is constructed in the form $p(t_1, t_2, \ldots, t_n)$, where *p* is a *predicate name*, $t_1, t_2, \ldots, t_n$ are terms, and *n* is the predicate's *arity* or *valence*. A predicate name with a valence is called a *predicate symbol*, denoted by *p/n*.

3. Propositional Connectives

Propositional connectives include ¬ (negation), ∧ (and) ∨ (or) → (conditional) and ↔ (bi-conditional). The symbol → sometimes reads "implies", and ↔ is also known as "if and only if", or "iff" in abbreviation.

4. Quantifiers

Quantifiers include the *universal quantifie*r ∀ and the *existential quantifier* ∃ . They are used to quantify variables. ∀*?x* reads "for all *?x*" and ∃*?x* reads "there is a *?x*". A variable is called a *free variable* if it is not quantified.

5. Atoms and Literals

Atoms are predicates. Some languages include the equivalence symbol "=", and in those cases atoms also contain *t1=t2*, where *t1* and *t2* are arbitrary terms. *Literals* are predicates or negation of predicates. For example, if *p* is a predicate, then *p* is both an atom and a literal, and ¬*p* is a literal.

6. Formulae

Formulae can be recursively defined as the follows:

- Each atom and literal is a formula;
- If φ and ψ are formulae then $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi$, and $\varphi \leftrightarrow \psi$ are formulae;

- If φ is a formulae, then $(\forall ?x)\ \varphi$ and $(\exists ?x)\ \varphi$ are formulae, where $?x$ could be any variables.

Finally, a formula always returns a *truth value*, i.e. either true or false. A formula is called a *sentence* if it contains no free variables.

7. Deduction

Let φ be a formulae and Γ denote a set of formulae. An *argument* can be written in the form $<\Gamma, \varphi>$, where Γ is the premise and φ the conclusion of the argument. A *deductive system* is a system that shows whether φ is true given all formulae in Γ are true. Deduction is usually based on a set of structural rules defined by the *proof theory* of the deductive system. If the argument is deduced to be true, we say Γ implies φ and φ is a logical consequence of Γ, denoted by $\Gamma \vdash \varphi$.

8. Theory

A *logical theory* is a set of sentences. A theory Γ is *consistent* if and only if it implies no contradictory logical consequences. In other words Γ is *consistent* if and only if there is no such a formula φ that $\Gamma \vdash \varphi$ and $\Gamma \vdash \neg\varphi$.

Appendix B

The First Order Model Theory

1. Signature

A signature is a tuple $K = (CON, FUN, PRED)$, where *CON* is a set of constant symbols, *FUN* is a set of function symbols, and *PRED* is a set of predicate symbols. Given a signature *K*, we denote its *CON*, *FUN*, and *PRED* as *CON(K)*, *FUN(K)*, and *PRED(K)*. In addition, we use *K*-terms, *K*-constants, *K*-functions, *K*-predicates, *K*-literals, and *K*-formulae to denote terms, constants, functions, predicates, and literals constructed based on *K*. A language L_K is a first order language that contains only *K*-terms and *K*-formulae.

2. Variable Occurrence

Let *t* be a term. We use *vars(t)* to denote the set of all variables occurring in *t*. More specifically:

- $vars(?x) = \{?x\}$;
- $vars(c) = \emptyset$, where *c* is a constant;
- $vars(f(t_1, t_2, \ldots, t_n)) = \bigcup_{i=1}^{n} vars(t_i)$.

Let *T* be a set of terms. We use *vars(T)* to denote the set of all variables occurring in *T*:

- $vars(T) = \bigcup_{t \in T} vars(t)$.

Let φ be a formula. We use $vars(\varphi)$ to denote the set of all variables occurring *freely* in φ, which can be recursively defined as the follows:

- $vars(t_1 = t_2) = vars(t_1) \cup vars(t_2)$, where t_1 and t_2 are terms;
- $vars(p(t_1, t_2, \ldots, t_n)) = \bigcup_{i=1}^{n} vars(t_i)$, where *c* is a constant;
- $vars(\neg\varphi) = vars(\varphi)$;
- $vars(\varphi \wedge \psi) = vars(\varphi) \cup vars(\psi)$;
- $vars(\varphi \vee \psi) = vars(\varphi) \cup vars(\psi)$;
- $vars(\varphi \rightarrow \psi) = vars(\varphi) \cup vars(\psi)$;
- $vars(\varphi \leftrightarrow \psi) = vars(\varphi) \cup vars(\psi)$;
- $vars((\forall ?x)\varphi) = vars(\varphi) \setminus ?x$, where the quantified variable $?x$ is execluded;
- $vars((\exists ?x)\varphi) = vars(\varphi) \setminus ?x$, where the quantified variable $?x$ is execluded.

It is noticeable that a logical sentence is then a formula φ where $vars(\varphi) = \emptyset$.

Finally let Γ be a set of formula, and denote the set of free variables occuring in all formulae of Γ.

- $vars(\Gamma) = \bigcup_{\varphi \in \Gamma} vars(\varphi)$.

3. Model and Interpretation

Let K be a signature and D denote an enumerable set called a domain of discourse. A *K-model* over D, denoted by M, is a bijective function that satisfies the following properties:

- Each constant $c \in CON(K)$ is mapped to an element of D, denoted by $M(c)$;
- Each function symbol $f/n \in FUN(K)$ is mapped to a function $M(f): D_1 \times D_2 \times ... \times D_n \rightarrow D_0$, where each $D_i \subseteq D$.
- Each predicate symbol $p/n \in PRED(K)$ is mapped to *n-ary* relation $M(p) \subseteq D_1 \times D_2 \times ... \times D_n$, where each $D_i \subseteq D$.

We conveniently use the following notation: $M(K)$ denotes all mappings defined by the model M; $M(CON(K))$ denotes all mappings for constants; $M(FUN(K))$ denotes all mappings for function symbols, and $M(PRED(K))$ denotes all mappings for predicate symbols. Finally, we have $M(K) = M(CON(K)) \cup M(FUN(K)) \cup M(PRED(K)))$

Let M be a K-model and X denote a set of variables. A *variable binding* β is a function that map X to $CON(K)$, denoted by $\beta: X \rightarrow CON(K)$. More specifically, for each $?x \in X$, we have a *variable assignment* $\beta(?x) = c \in CON(K)$. The 2-tuple $I = (M, \beta)$ defines a *K-interpretation*, where M is I's model and β is I's variable binding.

Let $I_1 = (M_1, \beta_1)$ and $I_2 = (M_2, \beta_2)$ be two K-interpretations over X, and $?x \in X$ denote a variable. We say I_2 is a *?x-variant* of I_1 if $M_1 = M_2$ and $\beta_1(?y) = \beta_2(?y)$ for each $?y \in (X \setminus \{?x\})$. This means I_1 and I_2 agree on all variable assignments except the one for $?x$.

4. Denotation

Let K be a signature, X be a set of variables, D be a domain of discourse, $I = (M, \beta)$ be a K-interpretation over X, and t denote a K-term, where $vars(t) \in X$. We use $I(t) \in D$ to denote t's *denotation* under I, which can be recursively defined as the follows:

- If t is a constant, then $I(t) = M(t)$;
- If t is a variable, then $I(t) = M(\beta(t))$;
- If t is function $f(t_1, t_2, \ldots, t_n)$, then $I(t) = M(f)(I(t_1), I(t_2) \ldots, I(t_n))$.

5. Model of A Formula

Let K be a signature, M be a K-model, X be a set of variables, and φ denote a K-formula, where $vars(\varphi) \in X$. Suppose $I=(M, \beta)$ is an K-interpretation over X. W say that *I interprets* φ, denoted by $I \vDash \varphi$, if the formula φ meet one of the following conditions:

- $I \vDash (t_1 = t_2)$ iff $I(t_1) = I(t_2)$, where φ is $(t_1 = t_2)$ and t_1, t_2 are two terms;
- $I \vDash p(t_1, t_2, \ldots, t_n)$ iff $(I(t_1), I(t_2) \ldots, I(t_n)) \subseteq M(p)$; where φ is $p(t_1, t_2, \ldots, t_n)$;
- $I \vDash \neg\psi$ iff $I \nvDash \psi$, where φ is $\neg\psi$;

- $I \vDash (\psi_1 \wedge \psi_2)$ iff $I \vDash \psi_1$ and $I \vDash \psi_2$, where φ is $\psi_1 \wedge \psi_2$;
- $I \vDash (\psi_1 \vee \psi_2)$ iff $I \vDash \psi_1$ or $I \vDash \psi_2$, where φ is $\psi_1 \vee \psi_2$;
- $I \vDash (\psi_1 \rightarrow \psi_2)$ iff $I \nvDash \psi_1$ or $I \vDash \psi_2$, where φ is $\psi_1 \rightarrow \psi_2$;
- $I \vDash ((\forall ?x)\psi)$ iff $I' \vDash \psi$ for all ?x-variants $I'$of I, where φ is $(\forall ?x)\psi$;
- $I \vDash ((\exists ?x)\psi)$ iff $I' \vDash \psi$ for some ?x-variants I' of I, where φ is $(\exists ?x)\psi$.

For a K-model M and a K-formula φ, if there is a K-interpretation $I = (M, \beta)$ that interprets φ, we say that M is a model of φ, denoted by $M \vDash \varphi$. In other words, $M \vDash \varphi$ if and only if there is a variable binding β such that $(M, \beta) \vDash \varphi$. For a domain of discourse D, we say φ is *valid* if all K-interprations over $vars(\varphi)$ interpret φ. On the other hand, φ is *satisfiable* if there is at least one K-interpretation over $vars(\varphi)$ that interprets φ., i.e. φ has at least one model. Otherwise φ is *unsatisfiable*.

5. Model of A Set of Formulae

Let K be a signature, Γ be a set of K-formulae, X be a set of variables where $vars(\Gamma) \subseteq X$, and $I=(M, \beta)$ denote a K-interpretation over X. We say that I interprets Γ, denoted by $I \vDash \Gamma$, if and only if I interpets every formula $\varphi \in \Gamma$.

For a K-model M and a set of K-formulae Γ, if there is a K-interpretation $I = (M, \beta)$ that interprets Γ, we say that M is a model of Γ, denoted by $M \vDash \Gamma$. In other words, $M \vDash \Gamma$ if and only if there is a variable binding β such that $(M, \beta) \vDash \Gamma$. For a domain of discourse D, we say Γ is *valid* if all K-interprations over $vars(\Gamma)$ interpret Γ. On the other hand, Γ is *satisfiable* if there is at least one K-interpretation over $vars(\Gamma)$ that interprets Γ., i.e. Γ has at least one model. Otherwise Γ is *unsatisfiable*.

6. Semantic Entailment

Let K be a signature, φ be a K-formula, Γ be a set of K-formulae, and D denote a domain of discourse. Suppose $M(\Gamma)$ is the set of all models of Γ and $M(\varphi)$ is the set of all models of φ. If $M(\Gamma) \subseteq M(\varphi)$, we say that Γ semantically entails φ, denoted by $\Gamma \vDash \varphi$. More specifically, φ is a entailment of Γ if and only if all models of Γ are also models of φ.

References

Albrecht, J. (1996). Universal Analytical GIS Operations: a task-oriented systematization of data structure-independent GIS functionality leading towards a geographical modeling language. Vichta, Germany, University of Vichta.

Albrecht, J. (1997). VGIS - A GIS shell for the conceptual design of environmental models. Innovation in GIS. Z. Kemp. London, Taylor & Francis**:** 154-165.

Andersen, P. B. (1997). A theory of computer semiotics: semiotic approaches to construction and assessment of computers. Cambridge, Cambridge University Press.

Baader, F., D. Calvanese, D. McGuinness, D. Nardi and P. F. Patel-Schneider, Eds. (2003). The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge, Cambridge University Press.

Baker, V. (1999). "Geosemiosis." Geological Society of America Bulletin **111**(5): 633-645.

Barwise, J. and J. Etchemendy (1987). The Liar: An Essay in Truth and Circularity. Oxford, Oxfore University Press.

Batty, M. and P. Longley (1986). "The Fractal Simulation of Urban Structure." Environment and Planning A **18**: 1143-1179.

Batty, M. and Y. Xie (1994). "Modelling inside GIS. Part1: Model Structure, exploratory spatial data analysis and aggregation." International Journal of Geographical Information Systems **8**(3): 291-307.

Behrens, J. O., D. D. Moyer and G. Wunderlich (1974). Land Title Recording in the United States: A statistical Summary. Washington D.C., U.S. Department of Agriculture and Bureau of the Census, U.S. Department of Commerce.

Bennett, D. A. (1997). "A framework for the integration of geographical information systems and modelbase management." International Journal of Geographical Information Science **11**(4): 337-367.

Bhaskar, R. and H. A. Simon (1977). "Problem solving in semantically rich domains: An example form engineering thermodynamics." Cognitive Science **1**: 193-215.

Bishr, Y. (1997). Semantic Aspects of Interoperable GIS, Wageningen Agricultural University.

Bishr, Y. (1998). "Overcoming the Semantic and Other Barriers to GIS Interoperability." International Journal of Geographical Information Science **12**(4): 299-314.

Blum, A. L. and M. L. Furst (1997). "Fast planning through planning graph analysis." Artificial Intelligence **90**(1-2): 281-300.

Brodaric, B. and M. Gahegan (2001). Learning Geoscience in Situ: Implications for Geographical Knowledge Representation. ACM-GIS 2001: The Ninth Association for Computing Machineary International Symposium on Advances in GIS, Atlanta, GA.

Burris, S. N. and H. P. Sankappanavar (1981). A Course in Universal Algebra, Springer Verlag.

Buzan, T. (1995). The MindMap book. London, BBC Books.

Canter, L. W. (1996). Environment Impact Assessment. New York, NY, McGraw Hill.

Chandler, D. (2001). Semiotics: The Basics. London, Routledge.

Chang, S. K. (1986). Visual languages and iconic languages. Visual Languages. S. K. Chang, T. Ichikawa and P. Ligomenides. New York, Plenum Press**:** 1-7.
Chin, G., L. R. Leung, K. Schuchardt and D. Gracio (2002). New paradigms in problem solving environments for scientific computing. 7th international conference on Intelligent user interfaces, San Francisco, California, USA, ACM Press New York, NY, USA.
Chomsky, N. (1986). Knowledge of Language: Its Nature, Origins, and Use, Paeger Paperback.
Chorafas, D. N. (1996). Visual Programming Technology, Mcgraw-Hill (Tx).
Chrisman, N. R. (1998). Academic Origins of GIS. The History of Geographic Information Systems: Perspectives from the Pioneers. T. W. Foresman. Upper Saddle River, NJ, Prentice Hall PTR**:** 33-43.
Crosier, S. J., M. F. Goodchild, L. L. Hill and T. R. Smith (2003). "Developing an infrastructure for sharing environmental models." Environment and Planning B: Planning and Design **30**(487-501).
Culler, G. J. and B. D. Fried (1963). An On-line computing center for scientific problems. Pacific Computer Conference, IEEE, Piscataway, NJ.
Curry, M. (1995). GIS and the inevitability of ethical inconsistency. Ground Truth. J. Pickles. New York, Guilford Press**:** 68-87.
Curry, M. (1995). "Rethinking rights and responsibilities in geographic information systems: beyond the power of the image." Cartography and Geographic Information Systems **22**: 58-69.
Davidson, D. (1969). Truth and meaning. Philosophical. J. W. Davis, Hingham**:** 1-20.
Davis, M. (2000). Engines of Logic: Mathematicians and the Origin of the Computer. New York, W. W. Nortion.
DiBiase, D. (1990). "Visualization in the Earth Sciences." Earth and Mineral Sciences Bulletin **59**(2): 13-18.
Dobson, J. E. (1983). "Automated Geography." Professional Geographer **35**(2): 135-143.
Dobson, J. E. (1985). "Automated Geography and Emergency Management." The Information Society **3**(4): 347-359.
Dobson, J. E. (1993). "The geographic revolution: a retrospective on the age of automated geography." The Professional Geographer **45**(4): 431-439.
Dobson, J. E. (1993). "A Rationale for the National Center for Geographic Information and Analysis." The Professional Geographer **45**(2): 207-215.
Dodig-Crnkovic, G. (2006). Philosophy of Information, a New Renaissance and the Discrete Charm of the Computational Paradigm. Computing, Philosophy and Cognition. L. Magnani. London, King's College Publications.
Doherty, D. and R. C. Leinecker (1999). JavaBeans Unleashed, Sams.
Dummet, M. (1996). Frege and Other philosophers, Oxford University Press, USA.
Eastman, J. R. (1987). "IDRISI: a collective geographic analysis system project." Cartographica **28**(3): 65-74.
Eckstein, R., M. Loy and D. Wood (2002). Java Swing, O'Reilly Media, Inc.
Egenhofer, M. and H. T. Bruns (1995). Visual Map Algebra: A Direct-Manipulation User Interface for GIS. Visual Database Systems 3, Visual Information Management, Proceedings of the Third IFIP 2.6 Working Conference on Visual Database Systems. S. Spaccapietra and R. Jain. Lausanne, Switzerland, Chapman & Hall**:** 235-253.

Egenhofer, M. and J. Richards (1993). The geographer's desktop: a direct-manipulation interface for map overlay. Autocarto II, Minneapolis, MN.
Egenhofer, M. J. (2002). Toward the Semantic Geospatial Web. ACM-GIS, McLean, VI, ACM Press.
Ehrenfels, C. F. v. (1890). Uber Gestaltqualitaten (On Gestalt-qualities). Vierteljahresschrift fur wissenschaftliche Philosophie. **Jg. 14**.
Eiter, T. and G. Gottlob (1992). "On the complexity of propositional knowledge base revision, updates and counterfactuals." Artificial Intelligence **57**(227-270).
Emerson, E. A. (1990). Temporal and modal logic. Handbook of Theoretical Computer Science. Cambridge, MA, The MIT Press.
Erol, K., J. Hendler and D. Nau (1994). HTN Planning: Complexity and Expressivity. 12th National Conference on Artificial Intelligence (AAAI-94).
Erol, K., D. Nau and J. Hendler (1994). UMCP: A sound and complete planning procedure for hierarchical task network planning. Second International Conference of AI Planning Systems.
ESRI, I. (2004). The ArcGIS Desktop Developer Guide.
Fan, W., L. Wallace, S. Rich and Z. Zhang (2006). "Tapping the power of text mining." Communications of the ACM **49**(9): 76-82.
Farrugia, J. (2003). Model-Theoretic Semantics for the Web. 12th international conference on the World Wide Web, Budapest, Hungary, ACM Press, NY.
Fayyad, U., G. Piatetsky-Shapiro and P. Smyth (1996). "The KDD process for extracting useful knowledge from volumes of data." Communications of the ACM **39**(11): 27-34.
Fikes, R. E. and N. J. Nilsson (1971). "STRIPS: A new approach to the application of theorem proving to problem solving." Artificial Intelligence **2**: 189-208.
Fonseca, F. T., K. A. V. Borges, M. J. Egenhofer and C. A. Davis Jr (2000). "Ontologies and knowledge sharing in urban GIS." Computers, Environment and Urban Systems **24**(3): 251-271.
Fonseca, F. T. and M. J. Egenhofer (1999). "Ontology-Driven Geographic Information Systems." ACM GIS: 14-19.
Foresman, T. W. (1998). GIS Early Years and the Threads of Evolution. The History of Geographic Information Systems: Perspectives from the Pioneers. T. W. Foresman. Upper Saddle River, NJ, Prentice Hall PTR**:** 3-16.
Gahegan, M. (1996). "Specifying the transformations within and between geographic data models." Transactions in GIS **1**(2): 137-152.
Gahegan, M. (2005). Beyong tools: visual support for the entire process of GIScience. Exploring Geovisualization. J. Dykes, A. M. MacEachren and M. J. Kraak. Amsterdam, Elsevier Science**:** 83-99.
Gahegan, M. and B. Brodaric (2002). Computational and visual support for geographical knowledge construction: filling in the gaps exploration and explanation. 10th International Symposium on Spatial Data Handling, Springer, New York.
Gahegan, M. and B. Brodaric (2002). Examining Uncertainty in the Definition and Meaning of Geographical Categories. 5th International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences, Melbourne.
Gahegan, M., F. Hardisty, M. Takatsuka and M. Wheeler (2002). "Introducing Geo VISTA Studio: An integrated suite of visualization and computational methods

for exploration and knowledge construction in geography." Computers, Environment and Urban Systems **26**(4): 267-292.
Gahegan, M., M. Takatsuka, M. Wheeler and F. Hardisty (2000). GeoVISTA Studio: A Geocomputational Workbench. The 5th International Conference on GeoComputation, University of Greenwich, United Kingdom.
Gahegan, M., M. Wachowicz, M. Harrower and T. M. Rhyne (2001). "The integration of geographic visualization with knowledge discovery in databases and geocomputation." Cartography and Geographic Information Science **28**(1): 29-44.
Gallopoulos, E., E. Houstis and J. R. Rice (1994). "Computer as Thinker/Doer: Problem-Solving Environments for Computational Science." IEEE Computational Science & Engineering **1**(2): 11--23.
Gallopoulos, S., E. Houstis and J. R. Rice (1994). "Problem-solving environments for computational Science and Engineering." IEEE Computational Science and Engineering **1**: 11-23.
Garner, B. J. and J. F. O'Callaghan (1998). The Development of GIS in Australia: From Infancy towards Maturity. The History of Geographic Information Systems: Perspectives from the Pioneers. T. W. Foresman. Upper Saddle River, NJ, Prentice Hall PTR**:** 3-16.
Garvan, F. (2002). The Maple book. Boca Raton, Fla., Chapman & Hall.
Gelfond, M. and V. Lifschitz (1993). "Representing action and change by logic programs." Journal of Logic Programming **17**: 301-322.
Gentzen, G. (1934). Investigations Into Logical Deduction. Collected Paper. M. E. Szabo. Amsterdam, North-Holland**:** 68-128.
Getis, A. and J. K. Ord (1992). "The analysis of spatial association by use of distance statistics." Geographical Analysis **24**: 189-206.
Ghallab, M., D. Nau and P. Traverso (2004). Automated Planning: Theory and Practice, Morgan Kaufmann.
Girard, J.-Y. (1987). "Linear Logic." Theoretical Computer Science **50**(1): 1-102.
Godel, K. (1992). On Formally undecidable propositions of principia mathematica and related systems. New York, Dover Publication.
Goguen, J. (1999). An introduction to algebraic semiotics, with applications to user interface design. Computation for metaphor, analogy and agents. C. Nehaniv. Berlin, Springer Verlag**:** 242-291.
Goodchild, M. F. (1988). "A spatial analytic perspective on geographical information systems." International Journal of Geographical Information Systems **1**(327-334).
Goodchild, M. F. (1991). "Just the facts." Political Geography Quarterly **10**: 335-337.
Goodchild, M. F. (1992). "Geographical data modeling." Computers & Geosciences **18**(4): 401-408.
Goodchild, M. F. (1992). "Geographical information science." International Journal of Geographical Information Systems **6**(1): 31-45.
Goodchild, M. F. (1992). "Integrating GIS and spatial data analysis: problems and possibilities." International Journal of Geographical Information Systems **6**(5): 407-423.
Goodchild, M. F. (2000). What is GIS? The NCGIA Core Curriculum in GIScience. NCGIA.

Goodchild, M. F. and et al. (1992). "Integrating GIS and spatial data analysis: problems and possibilities." International Journal of Geographical Information Systems **6**(5): 407-423.
Goodchild, M. F. and R. P. Haining (2003). "GIS and spatial data analysis: Converging perspectives." Papers in Regional Science **83**(1): 363-385.
Goodchild, M. F., R. P. Haining and S. W. a. others (1992). "Integrating GIS and spatial data analysis: problems and possibilities." International Journal of Geographical Information Systems **6**(5): 407-423.
Goodchild, M. F. and D. M. Mark (1987). "The Fractal nature of geographic phenomena." Annals of Association of American Geographers **77**: 265-278.
Goodchild, M. F., E. Sheppard, M. J. Egenhofer, K. K. Kemp and D. M. Mark (1999). "Introduction to the Varenius project." International Journal of Geographical Information Science **13**(8): 731-745.
Gordon, R. (1998). Essential JNI: Java Native Interface, Prentice Hall.
Gruber, T. R. (1993). "An translation approach to portable ontologies." Knowledge Acquisition **5**(2): 199-220.
Grzechynka, J. and D. Harezlak (2000). Problem Solving Environments.
Heiler, S. (1995). "Semantic Interoperability." ACM Computing Surveys **27**(2): 271-273.
Hero, P. G. (2003). Assessing the Vulnerability to Environmental Hazards - A Hero Protocol.
Hodges, W. (1993). Model Theory. Cambridge, Cambridge University Press.
Hoffer, R. M. and staff (1975). Natural Resource Mapping in Mountainous Terrain by Computer Analysis of ERTS-1 Satellite Data. Agricultural Experiment Station Research Bulletin 919. West Lafayette, IN: Purdue University.
Horton, W. (1994). The Icon Book: Visual Symbols for computer systems and documentation. New York, John Wiley.
Houstis, E. N. and J. R. Rice (2000). On the future of problem solving environments. West Lafayette, IN, Computer Sciences Deparatment, Purdue Univeristy**:** 78.
Houstis, E. N. and J. R. Rice (2002). Future problem solving environments for computational science. Computational science, mathematics and software. West Lafayette, IN, USA, Purdue University Press**:** 93-104.
Houstis, E. N., J. R. Rice and T. S. Papatheodorou (1989). "Paralle ELLPACK: An expert system for parallel processing of partial differential equations." Mathematics and Computers in Simulation **31**: 497-508.
IETF (1998). RFC 2396: Uniform Resource Identifiers (URI): Generic Syntax.
Jeffrey, R. C. and J. P. Burgess (2006). Formal Logic: Its Scope and Limits, Hackett Publishing Company.
Keye, D. (2003). Loosely Coupled: The Missing Pieces of Web Services, Rds Associates Inc.
Kirby, C. and M. Pazner (1990). Graphic Map Algebra. 4th International Symposium on Spatial Data Handling, Zurich, Swtizerland.
Klerer, M. and J. Reinfelds (1968). Interactive Systems for Experimental Applied Mathematics, Academic Press.
Kraines, S., H. Komiyama, R. Batres, M. Koyama and D. Wallace (2005). "Internet-based integrated environmental assessment: Using ontologies to share computational models." Journal of Industrial Ecology **9**(3): 31-50.

Kripke, S. (1963). "Semantical Consideration on Modal Logic." Acta Philosophica Fennia **16**: 83-94.
Kripke, S. (1975). "Outline of a theory of truth." Journal of Philosophy **72**: 690-716.
Krishnan, P. S. O. (1997). "Integrated coastal water monitoring." Coastal Management **25**: 437-443.
Krueger, C. W. (1992). "Software Reuse." Acm Computing Surveys **24**(2): 131-183.
Kuhn, W. (2001). "Ontologies in support of activities in geographical space." International Journal of Geographical Information Science **15**(7): 613-631.
Lake, R. W. (1993). "Planning and applied geography: positivism, ethics and geographic information systems." Progress in Human Geography **17**: 404-413.
Lanter, D. and R. Essinger (1991). User-Centered Graphical User Interface Design for GIS, National Center for Geographical Information Analysis.
Lee, E. A. and T. M. Parks (1995). "Dataflow process networks." Proceedings of the IEEE **83**(5): 773-799.
Leuf, B. (2006). The Semantic Web: Crafting Infrastructure for Agency, Wiley.
Levesque, H., F. Pirri and R. Reiter (1998). "Foundation for the situation calculus." Electronic Transactions on Artificial Intelligence **2**(3-4): 159-178.
MacEachren, A. M. (1995). How Maps Work: Representation, Visualization, and Design. New York, The Guilford Press.
MacEachren, A. M., R. Masters, M. Wachowicz, R. Edsall and D. Haug (1999). "Constructing knowledge from multivariate spatiotemporal data: Integrating geographical visualization with knowledge discovery in database methods." International Journal of Geographical Information Systems **13**(4): 311-334.
Macmillan, W. D. (1997). Computing and the science of geography: the postmodern turn and the geocomputational twist. 2nd International Conference on GeoComputation, University of Otago, New Zealand.
Mark, D. M., B. Tversky, C. Freksa, S. C. Hirtle and R. Lloyd (1999). "Cognitive models of geographical space." International Journal of Geographical Information Science **13**(8): 747-774.
Marx, R. W. (1986). "The Tiger System: Automating the Geographic Structure of The United States Census." Government Publications Review **13**: 181-201.
Mathworks, I. T. (2003). Matlab Documentation.
McGuinness, D. L. and F. v. Harmelen (2004). OWL Web Ontology Language Overview.
Microsoft (1996). Distributed component object model protocol - DCOM/1.0.
Monmonier, M. (1992). "Authoring graphic scripts: experiences and principles." Cartography & Geographic Information Systems **19**(4): 247-260.
Morris, C. W. (1970). Foundations of the Theory of Signs. Chicago, Chicago University Press.
Newell, A., J. C. Shaw and H. A. Simon (1958). "Elements of a theory of human problem solving." Psychological Review **65**: 151-166.
Newell, A. and H. A. Simon (1972). Human Problem Solving. Englewood Cliffs, NJ, Prentice Hall.
NRC (2006). Beyond Mapping: Meeting National Needs Through Enhanced Geographic Information Science, National Academies Press.

Nyerges, T. (1993). How do people use geographical information systems? Human factors in geographical information systems. D. Medyckyj-Scott and H. M. Hearnshaw. Boca Raton, FL, Belhaven Press**:** 37-49.
Object Management Group (1997). The complete CORBA/IIOP 2.1 specification, Object Management Group.
OGC (1999). OpenGIS Simple Features Specification for CORBA.
OGC (2004). Geography Markup Language.
OGC (2004). "OGC Web Map Service Interface."
OpenDX.org (2000). OpenDX: The open source software project based on IBM's visualization data explorer.
Openshaw, S. (1987). "A Mark 1 Geographical Analysis Machine for the automated analysis of point data sets." International Journal of Geographical Information Systems **1**(4): 335-358.
Openshaw, S. (1992). Some aspects of Information Technology and the future. AURISA92, Australasian Urban and Regional Information Systems Association.
Openshaw, S. (1994). "Computational human geography: Towards a research agenda." Environment and Planning A **26**: 499-508.
Pednault, E. P. D. (1986). Towards a mathematical theory of plan synthesis., Standford University.
Peirce, C. S. (1931-58). Collected Writings. Cambridge, MA, Harvard University Press.
Penberthy, J. S. and D. S. Weld (1992). UCPOP, A Sound, Complete, Partial Order Planner for ADL. The third international conference on principles of knowledge representation and reasoning.
Pickles, J. (1993). "Discourse on method and the history of discipline: reflections on Jerome Dobson's 1993 'Automated Geography'." Professional Geographer **45**: 451-455.
Pickles, J. (1997). "Tool or science? GIS, technoscience and the theoretical turn." Annals of Association of American Geographers **87**: 363-372.
Prawitz, D. (1965). Natural Deduction: A Proof-Theoretical Study. Stockholm, Almquist & Wiksell.
Pundt, H. and Y. Bishr (2002). "Domain ontologies for data sharing - An example from environmental monitoring using field GIS." Computers and Geosciences **28**(1): 95-102.
Reiter, B. (2002). How GRASS development reflects Freee Software history and what to expect next.
Rhind, D. (1998). The Incubation of GIS in Europe. The History of Geographic Information Systems: Perspectives from the Pioneers. T. W. Foresman. Upper Saddle River, NJ, Prentice Hall PTR**:** 293-306.
Rice, J. R. (1968). On the construction of polyalgorithms for automatic numeric analysis. Interactive Systems for Experimental Applied Mathematics. New York, Academic Press**:** 301-313.
Rice, J. R. (1976). "The algorithm selection problem." Advances in Computers **15**: 65-118.
Rice, J. R. and R. Boisvert (1985). Solving Elliptic Problems Using ELLPACK. New York, Springer-Verlag.
Rice, J. R. and S. Rosen (1966). NAPSS - A Numerical Analysis Problem Solving System. ACM National Conference.

Roberts, D. D. (1973). The Existential Graphs of Charles S. Peirce. Mouton, The Hague.
Saussure, F. d. (1974). Course in General Linguistics. London, Fontana/Collins.
Saussure, F. d. (1983). Course in General Linguistics. London, Duckworth.
Schreiber, G., H. Akkermans, A. Anjewierden, R. deHoog, N. Shadbolt, W. VandeVelde and B. Wielinga (1999). Knowledge Engineering and Management: The CommanKADS Methodology. Cambridge, MA, The MIT Press.
Schuurman, N. (2000). "Trouble in the heartland: GIS and its critics in the 1990s." Progress in Human Geography **24**(4): 569-590.
Scott, A. C., J. E. Clayton and E. L. Gibson (1991). A Practical Guide to Knowledge Acquisition, Addison-Wesley Professional.
Searle, J. (1980). "Minds, brains and programs." Behavioral and Brain Sciences **3**: 417-424.
Shastri, L. (1988). Semantic Networks: An Evidential Formalization and Its Connectionist Realization. London, Pitman.
Sheppard, E. (1993). "Automated geography: what kind of geography for what kind of society?" Professional Geographer **45**: 457-460.
Smith, B. and D. M. Mark (2001). "Geographical categories: An ontological investigation." International Journal of Geographical Information Science **15**(7): 591-612.
Smith, N. (1992). "History and philosophy of geography: real wars, theory wars." Progress in Human Geography **16**: 257-271.
Souza, C. S. d. (2005). The Semiotic Engineering of Human-Computer Interaction. Cambridge, MA, The MIT Press.
Sowa, J. (1976). "Conceptual graphs for a database interface." IBM Journal of Research and Development **20**(4): 336-357.
Sowa, J., Ed. (1991). Principles of Semantic Networks: Explorations in the Representation of Knowledge. San Mateo, CA, Morgan Kaufmann.
Sowa, J. (1999). Knowledge Representation: Logical, Philosophical, and computational Foundations, Brooks Cole.
Sternberg, R. J. (1995). Conceptions of expertise in complex problem solving: A comparison of alternative conceptions. Complex problem solving: the European Perspective. P. A. Frensch and J. Funke. Hillsdale, NJ, Lawrence Erlbaum Associates**:** 295-321.
Sui, D. Z. (1994). "GIS and urban studies: positivism, post-positivism, and beyond." Urban Geography **15**: 258-278.
Sun Microsystems (1997). The JavaBeans Specification (version 1.01).
Sun Microsystems, I. (1998). JavaBeans Specification for Java2.
Szyperski, C. (1998). Component Software: Beyond Object-Oriented Programming. New York, Addison-Wesley.
Takatsuka, M. and M. Gahegan (2002). "GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis and Visualization."
Tarski, A. (1944). "The semantic conception of truth." Philosophy and Phenomenlological Research **4**: 13-47.
Tarski, A. (1983). Logic, semantics,Metamathematics, papers from 1923 to 1938. Indianapolis, Hackett Publishing Company.
Taylor, P. J. (1990). "GKS." Political Geography Quarterly **9**: 211-212.

Taylor, P. J. and M. Overton (1991). "Further thoughts on geography and GIS." Environment and Planning A **23**: 1087-1090.
The GeoVISTA Center (2005). Data Mining in GeoVISTA Studio. University Park, Department of Geography, The Pennsylvania State University.
Tomlin, C. D. (1983). A Map Algebra. The 1983 Harvard Computer Graphics Conference, Cambridge, MA.
Tomlin, C. D. (1990). Geographic Information Systems and Cartographic Modeling. Englewood Cliffs, NJ, Prentice Hall.
Tomlinson, R. F. (1967). An introduction to the Geo-Information sysem of the canada land inventory. Ottawa, Canada, Canada Department of Forestry and Rural Development.
Tomlinson, R. F. (1968). A Geographical Information System for Regional Planning. Land Evaluation. G. A. Steard. South Melbourne, Australia, MacMillan of Australia**:** 200-210.
Ungerer, M. J. and M. F. Goodchild (2002). "Integrating spatial data analysis and GIS: a new implementation using the Component Object Model (COM)." International Journal of Geographical Information Science **16**(1): 41-54.
Uschold, M. and R. Jasper (1999). A Framework for Understanding and Classifying Ontology Applications. IJCAI99 Workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden.
USGS (1998). "American National Standard for Information Systems - Spatial Data Transfer Standard."
Van de Vlag, D., R. Jeansoulin, B. Vasseur and A. Stein (2005). "An application of problem and product ontologies for the revision of beach nourishments." International Journal of Geographical Information Science **19**(10): 1057-1072.
von Eckardt, B. (1993). What is Cognitive Science? Cambridge, MA, The MIT Press.
W3C (2004). OWL Web Ontology Language Overview.
W3C (2004). Web Services Architecture.
Weerawarana, S. (1994). Problem Solving Environments for Partial Differential Equation Based Applications. Department of Computer Sciences. West Lafayette, Indiana, Purdue University.
Weerawarana, S., E. N. Houstis, J. R. Rice, A. Joshi and C. E. Houstis (1996). "PYTHIA: a knowledge-based system to select scientific algorithms." ACM Transactions on Mathematical Software **22**(4): 447-468.
Wilde, N. and C. Lewis (1990). Spreadsheet-based Interactive Graphics: From Prototype to Tool. ACM Conference on Human Factors in Computing Systems.
Young, M., D. Argiro and S. Kubica (1995). "Cantata: Visual programming environment for the Khoros system." Computer Graphics **29**(2): 22-24.
Zhang, W. (1999). State-Space Search: Algorithms, Complexity, Extensions, and Applications. Springer.

Vita

Junyan Luo enrolled in the Department of Geography at the Pennsylvania State University in 2000 to pursue a Ph.D. degree. He received his Master and Bachelor's degrees in the Department of Urban and Resources Sciences at Nanjing University, China, in 2000 and 1997 respectively. He is a student member of the Association of American Geographers (AAG). His research interests cover GeoComputation, computational semantics, knowledge representation, and visualization.