A Distributed System Architecture for Spatial Data Management to Support Engineering Modeling

by

Ayman Ismail

B.Sc. Construction Engineering
American University in Cairo, 1995

Master of Business Administration
American University in Cairo, 1997

Submitted to the Department of Urban Studies and Planning
in partial fulfillment of the requirements for the degree of

Master in City Planning

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

ARCHIVES
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUL 1 9 1999
LIBRARIES

© Ayman Ismail, 1999. All rights reserved

The author grants hereby to MIT permission to reproduce and distribute publicly paper and electronic copies of this document in whole or in part.

Author ....
Department of Urban Studies and Planning
May 20, 1999

Certified by ....
Joseph Ferreira
Professor of Urban Planning and Operations Research
Department of Urban Studies and Planning
Thesis Supervisor

Accepted by ....
Associate Professor Paul Smoke
Chair, MCP Committee
Department of Urban Studies and Planning

A Distributed System Architecture for Spatial Data Management to Support Engineering Modeling

by

Ayman Ismail

Submitted to the Department of Urban Studies and Planning
on May 20, 1999 in partial fulfillment of the
requirements for the degree of Master in City Planning

Abstract

This research seeks ways to manage the process of analysis and synthesis of geographic data to support collaboration among researchers, planners, and engineers working on a spatial problem. This question is addressed on two levels. The first level examines the abstraction and representation of the analysis process, using the Unified Modeling Language. The second level examines the distributed environment that enables such collaboration, and proposes a three-tier distributed system architecture. The interdisciplinary Urban Respiration project provides a context and examples illustrating the need for such design. A prototype application is developed to test and understand the applicability of the proposed designs.

Thesis Supervisor: Joseph Ferreira
Title: Professor of Urban Planning and Operations Research

Acknowledgements

I would like to thank Professor Joseph Ferreira for constantly challenging me with new ideas and endeavors and for making the Planning Support Systems group such a lively learning environment. My thanks also go to John Evans and Nadine Alameh for their great support, thoughtful discussions, and motivation.

Special thanks go to Mike Shiffer, Qing Shen, Tom Grayson and Phil Thompson for making PSS and CRL a great place, to work, learn and collaborate.

I can never thank enough my family for challenging me to set high goals, and achieve them, for providing me with inspiration and for their continuous confidence in me.

Table of Contents

List of Figures

1. Introduction

1.1 Overview

Typical planning and engineering modeling problems require the collaboration of different users from different disciplines. They are data intensive and require mixing data sets from different sources and of different types and formats. Many system architectures exist to support modeling, data management, and collaboration for engineering problem solving. Most of these cater to the specific needs of a group of users and problems, but none satisfy the combined needs of engineering modeling and spatial analysis. This research addresses the following question: *how can we manage the process of analysis and synthesis of these data sets to support collaboration among different researchers, planners and engineers working on a spatial analysis problem?*

This research begins by analyzing existing computing architectures in terms of their ability to handle the demanding planning and engineering modeling. Geographic Information Systems (GIS) provide useful tools for performing spatial analysis to support planning and engineering modeling. These tools are usually not integrated with engineering modeling and analysis tools, which often have different data models. Emerging distributed information systems and collaborative tools offer the opportunity to create systems that support such integration.

This research looks at ways users of Geographic Information Systems and engineering modeling systems interact, and what underlying system architecture might support such interaction. It also examines the need for an abstract modeling language to capture and communicate analysis processes. By developing a prototypical distributed system architecture, the research suggests how users might use the Unified Modeling Language as an abstract graphical language to describe and communicate their modeling exercises. The prototype design links different data sources and servers in a distributed environment.

1.2 Problem Definition

The main question addressed through this research is how to manage the process of analysis and synthesis of geographic data shared among different researchers, planners, and engineers. This question is addressed on two levels. The first level examines the abstraction and representation of geographic analysis processes. The second level examines the distributed environment that enables such collaboration.

Users' interaction with GIS software highlights the issue of abstracting and representing the analysis process. Typical GIS software allows the user to operate on data that has a spatial dimension. This data is usually organized either as vector components (points, lines and polygons) or as raster grids with other attached attributes that describe these geographic entities.[1] During the analysis process the user performs some spatial functions on these data objects. This analysis process could include several steps, use several layers of data, generate intermediate

[1] Understanding GIS.

layers or incorporate other (non-spatial) data sets. It often requires several iterations through the process to experiment with various parameters and test possible hypotheses.

A sophisticated spatial analysis process is often faced with difficulties in

- recording or capturing the steps of the analysis process,
- describing the required steps to perform a certain analysis in an abstract format rather than a list of software-specific commands,
- building several nested layers of analysis that make it easier to understand the problem,
- documenting the process in a reproducible fashion,
- managing the different data sets involved in the analysis, and
- automating the process to allow for iteration or repetition.

An abstract model to represent a spatial analysis process could offer ways to alleviate the above difficulties.

Current GIS system architectures highlight the issue of operating in a distributed environment. Traditional Geographic Information Systems are based on a stand-alone model. All the required data is collected on a single powerful machine. The required software (which is usually a collection of packages from a single vendor) resides on the same machine. Data layers gathered from different sources are processed to prepare for a specific analysis. Such processing includes

- converting file formats for the geographic layers and the database files to ensure compatibility,
- converting between different coordinate system and projections to ensure geographic consistency, and
- customization and filtering of the data records and attributes to suit the specific analysis purpose.

A distributed environment, when available, is usually used for exchanging files and other types of communication[2].

Many of today's developments in distributed GIS are targeted towards the World Wide Web[3]. The current focus is on letting Internet users view geographic data and perform a limited set of queries on them through a web browser. Little attention is focused on developing system architectures and interfaces that allow for distributed analysis integrating several data sources and servers. In part this is due to the huge interoperability problems that exist in the GIS world.

The system architecture and prototype developed in this research attempt to examine the issue of operating in a distributed environment. Using the current GIS and programming tools, what are the major limiting factors to the development of a distributed GIS architecture?

[2] Ferreira et al, p. 99.
[3] Alameh, p. 97.

1.3 System Users

The identification of the potential users of this system architecture and prototype is important in determining the desired system characteristics and design objectives. The Urban Respiration project (discussed in detail in Chapter 2) is a typical case of interdisciplinary collaborative research. Each group has a different set of modeling needs, from Weather modeling, to atmospheric pollution analysis, to spatial analysis. The data used are captured and generated through different scientific devices and tools that are extremely diverse in their data models and file formats. In such a context, integrating data from different sources and using them in different applications is difficult. From a Geographic Information Systems perspective, these data need to be overlayed on standardized layers such as roads and landuse, and used in further analysis.

The targeted users for this research are in a variety of disciplines; a technically sophisticated planner, engineer or researcher. Their work is focused on the specific needs of a modeling problem, yet they operate in a collaborative interdisciplinary environment that requires the integration of different sources of information. Their analysis often includes a spatial component.

This research focuses on the needs of this user group. How can these users model and communicate their analysis process in a distributed environment? What system architectures will help them work together more effectively?

1.4 Objectives

In light of the problem definition and system users' needs outlined earlier, the objective of this thesis is to design a system architecture and prototype to support the spatial modeling process in a distributed environment.

This can be further divided into the following goals:

- Modeling of the spatial analysis process using an abstract modeling language.
- Understanding the current system architecture used in distributed GIS.
- Examining the different design alternatives for a distributed GIS system architecture.
- Implementing a prototype for the designed architecture.
- Evaluating the tools used in the implementation.

1.5 Research Methodology

The Thesis starts with a literature review that examines, first, Geographic Information Systems and engineering modeling. The next part examines a typical problem of collaboration involving geographic analysis, and introduces the Urban Respiration project in detail. The last part examines distributed system architectures. The Client/Server architecture used in the design of the prototype is given special attention.

The next step in the Thesis describes Unified Modeling Language (UML) and its possible application in this context. A subset of UML is selected as a graphical modeling language in the prototype. UML components are described briefly with a focus on using a subset to model geographic analysis.

In the next step, the characteristics and components of the system architecture are identified. The design alternatives for each component and the corresponding implementation limitations are examined. This leads to a proposed architecture that is then implemented in the prototype. A full description of the prototype and its different components follows. Using some examples from the Urban Respiration Project, the prototype is tested. This leads to a discussion on the limitation in the prototype and the reasons behind these limitations.

The conclusion includes an evaluation of the system architecture and prototype according to the earlier design criteria and user requirements. Some future work is suggested as a follow-up to this Thesis.

1.6 Thesis Overview

The Introduction covers the main objectives and overall context of this Thesis. Chapter 2 presents a background and a literature review. Chapter 3 describes modeling of geographic analysis processes using the Unified Modeling Language. Chapter 4 examines the system architecture and design alternatives. It ends with a description of the selected system architecture for the prototype implementation. Chapter 5 provides a description of the developed prototype and some examples form the Urban Respiration project to demonstrate it application.

2. Background

This research draws upon three bodies of knowledge, geographic information systems, engineering modeling and system architecture design. This chapter presents a background and overview of the current research trends in each, while concentrating on the issues most relevant to the problem at hand.

2.1 Geographic Information Systems

"A geographic information system (GIS) is a computer based information system that enables capture, modeling, manipulation retrieval, analysis and presentation of geographically referenced data.[4]" A GIS holds two types of data; geographic entities (an abstract representation of space) and attributes describing these geographic entities.

2.1.1 What GIS Can Do

What differentiates a GIS from other types of database systems is its ability to answer questions that have a spatial dimension. Here are a few typical functions of a GIS:

- Overlaying spatial data from different sources in a single map reveals a lot about the relationships among the different layers. This can be visually enhanced by using symbology, thematic mapping and other cartographic and visualization techniques.
- Questions about location such as locating places and proximity, locating objects at specific places and selecting places that satisfy certain criteria.
- Network analysis makes it possible to get the shortest path (time-wise or distance-wise) between two points or a series of points.
- Terrain analysis makes it possible to build terrain models and answer questions about path, visibility and viewsheds by incorporating a third dimension (elevation).
- Layer based analysis includes overlaying different layers to select locations that satisfy a specific spatial characteristics. For example areas with high density and education level within 10 miles from transit stations.
- Spatio-temporal analysis that includes comparing spatial information through time[5].

A GIS makes these functions practical by providing efficient data structures that can store, index and retrieve massive amounts of data referenced to a geographic coordinate system.

GIS can be applied to a number of fields and applications. It is often used in areas such as environmental science, landuse planning, utility management, automated mapping, remote sensing and cartography.

[4] Worboys, p. 1.
[5] Worboys, pp. 4-11.

2.1.2 Distributed GIS

Traditional Geographic Information Systems are based on a stand-alone model, which requires full access to the whole data set required for analysis (either locally or through a network file system). With the proliferation of the Internet as a set of standards and communication protocols, a distributed approach to GIS becomes practical. In a client/server environment, the spatial data and geo-processing server become available through a network connection to several client applications.

Interoperability problems on the applications and data levels represent the main challenges for distributed GIS.

Current developments are more inclined towards Web-based GIS, which allows the user to browse and query spatial data through a web browser interface [6]. Products like Internet Map Server from ESRI allow the user to publish a map on the World Wide Web with very limited viewing functionalities [7].

2.2 Engineering Modeling

One of the applications of Geographic Information Systems is in the field of engineering and scientific modeling. The Urban Respiration project is an example of the use of GIS to support analysis and modeling in such a context.

2.2.1 The Urban Respiration project

The Urban Respiration project is a multi-disciplinary project funded by the United States National Astronautics and Space Administration (NASA). It addresses a broad set of modeling and measurement issues concerned with urban metabolism and respiration and involves researchers from several institutions. The project, which started in 1997, aims at improved measurement of trace gas emissions and atmospheric chemistry in metropolitan areas, and a better understanding of how landuse influences air pollution patterns [8].

Participants in the project include Aerodyne Research, the Department of Urban Studies & Planning, and the Chemical Engineering Department at MIT, the University of Washington and the University of New Hampshire.

Aerodyne Research's work is focused on mobile monitoring equipment for *in situ* trace gas measurement. Their mobile van is capable of measuring the concentrations of trace gases in real-time while traveling around the city. The van records the spatial coordinates and time stamps for the measurement points using the Global Positioning System (GPS). The Chemical Engineering Department at MIT has developed models of atmospheric chemistry and air pollution in and above metropolitan areas. The University of Washington is focusing on the measurement and modeling of meteorological conditions. And the University of New Hampshire is focusing on the

[6] Alameh, pp. 24-26.
[7] Internet Map Server.
[8] Ferreira et al.

measurement and modeling of aerosol dynamics. The Department of Urban Studies & Planning at MIT is focusing on the spatial information infrastructure and urban land use models. This includes the development of distributed spatial information systems to facilitate collaboration among the different groups [9].

2.2.2 An Example of Collaboration Involving Geographic Analysis

As part of the Urban Respiration project, an analysis framework was developed that requires processing raw data from field readings, creating several raster layers that represent a spatial model of pollution distribution and finally overlaying these models produced from roads, landuse, meteorological, and other layers. The following example demonstrates the range of analysis and collaboration required in this context.

One goal of the project is to investigate the relationship between air pollution and landuse patterns. This included building a geographic model for the area of Manchester, New Hampshire that included landuse, transportation, demographic, topography, weather, and pollution data. The landuse, transportation, demographic, and topography layers were available from standard data sources at different levels of resolution. The weather and pollution data sets were the result of the experiments of two other collaborators in the project.

The measurement of trace-gas levels was done using a mobile van equipped with a GPS and trace gas measuring equipment. Several rounds of data collection were performed. Each round involved some pre-designed experiments to test some hypotheses, which required measurements along a certain path and time duration. To prepare for these experiments, researchers used GIS data layers including landuse, roads, and emission sources. Each round of measurement produced large files containing time-stamped geo-referenced pollution concentrations. These output files needed several processing steps to identify and remove special measurement and timing errors before others can make use of them.

In parallel with the trace-gas measurement experiments, another team was responsible for measuring the weather conditions in the same area. Their work was used to calibrate their weather models (using standard Penn State/NCAR Mesoscale Modeling System MM5)[10]. These models simulate the weather conditions for the duration of the experiments. The simulation is run in a nested fashion, beginning with a coarse 5 Km grid cell size in a large area (eastern U.S.), and then zooming in to a high resolution 1 Km grid cell size centered on the study area. The result is a large 3-dimensional grid with different weather parameters (e.g. temperature, humidity, and wind speed and direction) at several resolutions (5, 3, 1 Km). The creation of this model is computationally demanding (3 days of processing on a fast, dedicated server) and results in huge data files.

To build the spatial model, these data sets needed to be integrated with the other landuse and transportation layers. The first step after data collection and processing was to communicate the different types of models used and to discuss issues of data types, file formats and sizes. The second step was to move the files (in this case the files were transferred using Internet's FTP, File Transfer Protocol). The third step was to convert the data files from their application-

[9] Ferreira et al.
[10] MM5 documentation.

specific formats into a "GIS friendly" format. This required several applications to open, convert, and reformat the files. At the end, the output was ready to be added as a GIS layer to the model.

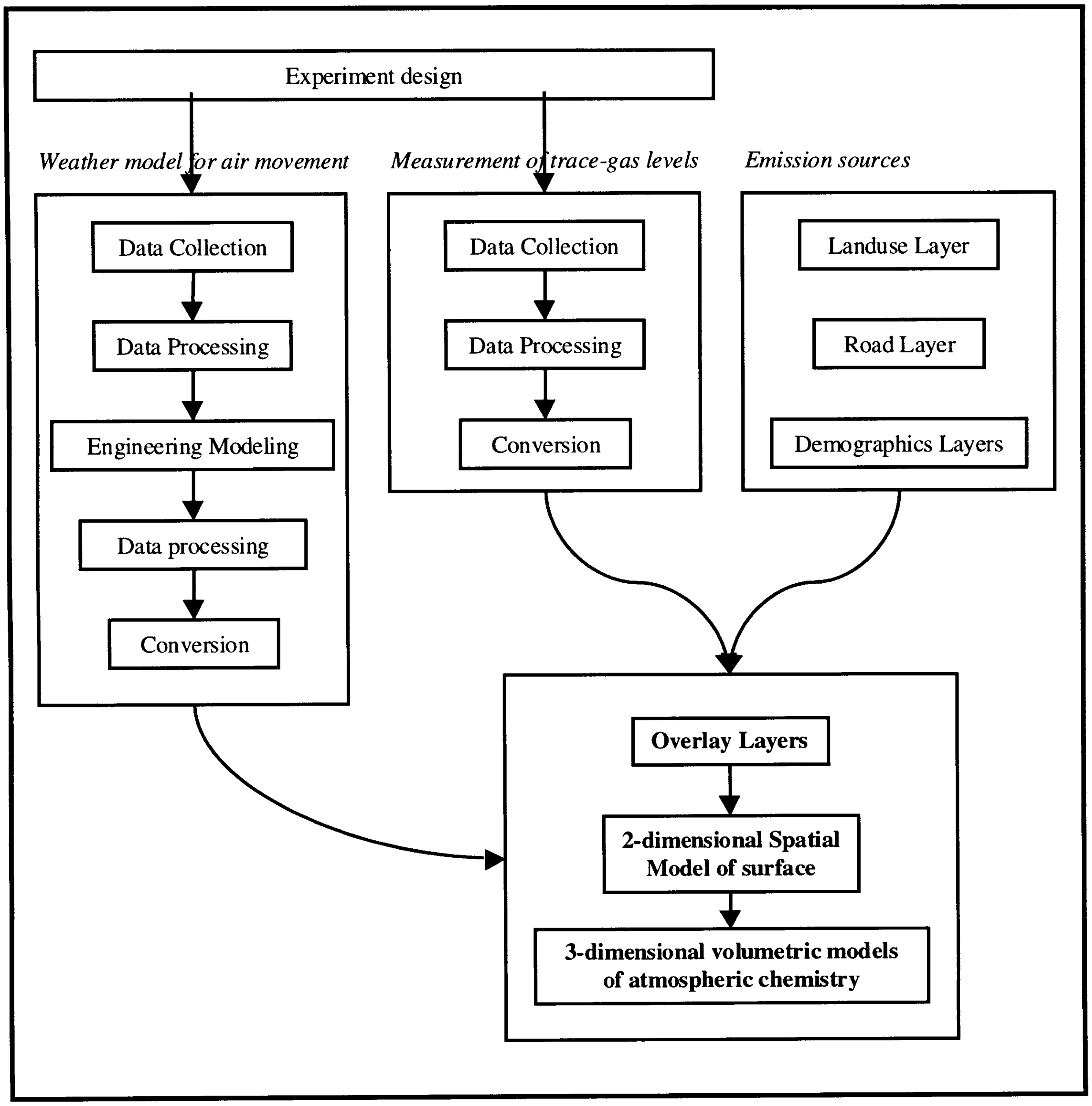


Figure 2.1: Example of collaboration involving a geographic analysis process from the Urban Respiration project.

This example describes a typical situation that involves collaboration among different researchers, each focusing on a specific type of modeling. Integrating the data resulting from these heterogeneous models into a single homogeneous geographic model proves not to be a trivial task. Each model has a different data model based on its specific processing needs. Outputs are produced at different levels of aggregation. For example, landuse layers could have different levels of aggregation, ranging from coarse layers covering the whole state, to more detailed layers covering the metropolitan area at the parcel level. The choice of scale for mixing data layers is also dependent on the requirements of the spatial and atmospheric models that it feeds into. One of the objectives of this exercise is to develop different models of surface layers at different resolution and levels of aggregation and see which aspects or assumptions are more important in the overall model. For example, what is the impact of using a more detailed landuse layer on the overall model accuracy?

At this stage the final outcome is a GIS model that includes all of the previous layers (and others). This will be further used to perform analysis that attempts to understand the relationship between the trace-gas measurements and the landuse, roads, weather and other parameters.

From a *data* perspective, this example involves several geographically-referenced data sets from different sources. Each has its own format and data structure that is customized to its initial application and discipline. From a *processing* perspective, the interaction among the different researchers is naturally in a distributed environment: different parts of the analysis are performed at different locations and using different software. A distributed architecture is naturally suited to such collaborative environment. From a *user* perspective, the nature of the project requires collaboration on one large problem. Each researcher is handling a small portion of the problem, yet the whole model needs to be integrated at the end. An abstract modeling language, one that is discipline- and technology-independent is a promising approach to facilitating this interdisciplinary, distributed collaboration.

2.3 System Architecture

A system architecture refers to the design of an information system with either its software or hardware components or both. The architecture of a system always defines its broad outlines, and may define precise mechanisms as well.

An "open" architecture allows the system to be connected easily to devices and programs made by other manufacturers. Open architectures use off-the-shelf components and conform to approved standards. A "closed" architecture is one whose design is proprietary, making it difficult to connect the system to other systems[11].

One of the main reasons for interoperability problems in GIS is that most of the systems were designed as closed systems using proprietary data structures and file formats that take advantages of emerging hardware and software technology in new ways. As GIS technologies matured, there have been efforts in the GIS community to address the interoperability issue through the creation of the Open GIS Consortium (OGC); an industry-wide consortium dedicated to creating open software interfaces to allow GIS systems to interact[12].

2.3.1 Mainframe Architecture

With mainframe software architectures all intelligence is within the central host computer. Users interact with the host through a terminal that captures keystrokes, sends them to the host, and receives screen-painting instructions (either text or graphics) in response. Mainframe software architectures are not tied to a hardware platform. User interaction can be done using PCs and UNIX workstations. A limitation of mainframe software architectures is that they do not easily support graphical user or access to multiple databases from geographically dispersed sites. Recently, mainframes have reemerged as servers in distributed client/server architectures[13].

2.3.2 File Sharing Architecture

Original PC networks were based on file sharing architectures, where the server downloads files from the shared location to the desktop environment. The requested user job is then run (including logic and data) in the desktop environment. File sharing architectures work if shared usage is low, update contention is low, and the volume of data to be transferred is low. Typically, this architecture can only satisfy about 12 users simultaneously. In the 1990s, PC local area network computing changed because the capacity of the file sharing was strained as the number of connected users increased and graphical user interfaces became popular [14].

2.3.3 Client/Server Architecture

Client/server architecture emerged as a result of the limitations of file sharing architectures. This approach is most common for a database application, where a database server replaces the file server. Using a relational database management system (RDBMS), user queries could be developed and displayed using local forms, and only the queries would be sent via the network,

[11] http://www.webopedia.com.
[12] Buehler.
[13] Edelstein.
[14] Schussel, Edelstein.

rather than a total file transfer. This reduced network traffic and improved multi-user updating. In client/server architectures, Remote Procedure Calls (RPCs) or structured query language (SQL) statements are typically used to communicate between client and server[15].

The term client/server was first used in the 1980s in reference to personal computers connected through a network. The actual client/server model started gaining acceptance in the late 1980s. The client/server software architecture is a versatile, message-based and modular infrastructure that is intended to improve usability, flexibility, interoperability, and scalability as compared to centralized, mainframe, time sharing computing.

A client is defined as a requester of services and a server is defined as the provider of services. A single machine can be both a client and a server depending on the software configuration[16]. Several variations of the Client/Server architecture exist, as follows.

Two-tier Architecture

With two-tier client/server architectures, the user-system interface is usually located in the user's desktop environment and the database management services are usually in a server that is a more powerful machine that services many clients. Processing management is split between the user system interface environment and the database management server environment. The database management server provides stored procedures and triggers. A number of software vendors provide tools to simplify development of applications for the two-tier client/server architecture[17]

Three-tier Architecture

Three-tier architectures emerged to overcome the limitations of the two-tier architecture. In the three-tier architecture, a middle tier was added between the client and the server environment. There are a variety of ways of implementing this middle tier, such as transaction processing monitors, message servers, or application servers. The middle tier can perform queuing, application execution, and database staging. For example, if the middle tier provides queuing, the client can deliver its request to the middle layer and disengage because the middle tier will access the data and return the answer to the client. In addition, the middle layer adds scheduling and prioritization for work in progress. The three-tier client/server architecture has been shown to improve performance for groups with a large number of users (in the thousands) and improves flexibility when compared to the two-tier architecture.

Distributed/Collaborative Enterprise Architecture

The distributed/collaborative enterprise architecture emerged in 1993. This software architecture is based on Object Request Broker (ORB) technology, but goes further than the Common Object Request Broker Architecture (CORBA) by using shared, reusable business models (not just objects) on an enterprise-wide scale as building blocks to provide a range of applications. The benefit of this architectural approach is that standardized business object models and distributed object computing are combined to give an organization flexibility to improve effectiveness organizationally, operationally, and technologically. An enterprise is defined here as a system

[15] Schussel, Edelstein.
[16] Schussel, Edelstein.
[17] Schussel, Edelstein.

comprised of multiple business systems or subsystems. Distributed/collaborative enterprise architectures are limited by a lack of commercially available object orientation analysis and design method tools that focus on applications[18].

[18] Shelton, Adler.

3. Modeling of Geographic Analysis Process

This chapter examines the need for an abstract, standard, graphical language for modeling of geographic analysis process. The Unified Modeling Language is selected for that purpose. This section presents an overview of UML components and sketches how it could be used in the context of collaborative geographic analysis.

3.1 The Need for Modeling of Geographic Analysis Process

As described earlier, the complexity of a typical geographic analysis exercise makes it difficult to capture all the steps involved in the process. An abstract modeling language could be used to capture this process. The earlier example from the Urban Respiration project provides some desired characteristics for this modeling language, which should:

- describe the problem in a discipline-independent way, so as to allow collaboration and communication across different disciplines,
- support analysis and synthesis of the problem at different levels of abstraction by decomposing a problem into smaller parts that can be tackled by different researchers from different disciplines and then integrated seamlessly,
- encapsulate implementation details while focus on conceptual problem, and
- provide an application-independent description for the problem.

The Unified Modeling Language (UML) is a promising vehicle for meeting the above requirements. The *Unified Modeling Language* is a standard language for modeling large systems, especially ones that are software intensive. It could be used to visualize, specify, construct and document the components of such a system [19]. UML represents the state-of-the-art in the modeling of large systems, especially object-oriented software. It offers a flexible and wide range of modeling capabilities that could be used to describe a problem at different levels of abstraction, from the conceptual level to the implementation level.

A model is a simplification of reality, the design blueprint of a system or process. Models are usually built to assist in the understanding of the systems or processes we are developing. In this context, UML performs four crucial functions:

- Visualizing the system at the development stage,
- Specifying the structure and behavior of the system,
- Providing a guiding template during the implementation or construction of the system,
- Documenting the system during the design and final stage [20].

[19] Booch et al, 1998, p13.
[20] Booch et al, 1998, p5.

3.2 Overview of the Unified Modeling Language

The emergence of object-oriented technology in the late seventies and early eighties resulted in several competing modeling languages and processes, including the *Booch method*[21], *Object-oriented Software Engineering*[22] and the *Object Modeling Technique*[23]. Each method offered a modeling language for system analysis and design using object-oriented technology and each was best fit for a specific domain of problems. Since 1994 the authors of these three methodologies have been collaborating to produce a unified modeling language. In 1997 the Object Management Group (OMG) adopted UML as the standard, and a revised version 1.3 was released in fall 1998 [24].

UML is currently considered the standard way to communicate object-oriented system designs. Because UML is a graphical communication language and not a systems design process or methodology, it can be used with any of the earlier methodologies in whole or in part. In most cases it is possible to represent a system using a small subset of the UML [25].

3.2.1 UML Elements

UML vocabulary includes three kinds of elements:

1. Things
2. Relationships
3. Diagrams

Things are abstractions of the different model elements. They are tied together using *Relationships. Diagrams* are meaningful groups of things and their relationships [26].

UML divides things into four different categories:

1. Structural things
2. Behavioral things
3. Grouping things
4. Annotational things

Structural things are the static elements of the system, either physical or conceptual. They include seven types: classes, interfaces, collaborations, use cases, active classes, components, and nodes. *Behavioral things* are the dynamic elements of the system which show the system's behavior over time and space. There are two kinds of behavioral elements: interaction and state machine. *Grouping things* are packages or folders that are used to organize the model by grouping related elements of the model together. *Annotational things* are comments that can be attached to elements to add description or comments. There is only one kind, called a note. [27]

[21] Booch, 1994.
[22] Jacobson, 1994.
[23] Rumbaugh et al, 1991.
[24] Booch et al, 1998, xviii.
[25] Booch et al, 1998, xviii.
[26] Booch et al, 1998, p. 17.
[27] Booch et al, 1998, pp. 18-23.

UML divides relationships into four different categories:
1. Dependency
2. Association
3. Generalization
4. Realization

A *dependency* is a semantic relationship between things where a change in the independent thing affects the semantic of the dependent thing. An *association* is a structural relationship that describes a connection among objects. A *generalization* is a parent/child relationship. The child object (specialized element) shares the structure and behavior of the parent object (generalized element) and is substitutable for it. A *realization* is a semantic relationship between classifiers where one classifier carries out (realizes) a certain function for another classifier.[28]

3.2.2 UML Diagrams

UML divides diagrams into nine different kinds:
1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Statechart diagram
7. Activity diagram
8. Component diagram
9. Deployment diagram

A *Class diagram* is a static perspective of the system showing the classes, interfaces, and their relationships. The *Object diagram* shows the instances of the Objects and their relationships. It represents the system prototype. A *Use case diagram* represents the behavior of the system by showing a set of use cases and actors with their relationships. A *Sequence diagram* and *Collaboration diagram* are both types of the *Interaction diagram.* Interaction diagrams represent a dynamic perspective of the system by showing the messages exchanged among the different objects. A *Sequence diagram* is an interaction diagram that orders the messages among the different system objects by time. A *Collaboration diagram* is an interaction diagram that orders the messages according to the structural organization of the objects. A *Statechart diagram* represents a dynamic perspective of the system by showing a state machine, which consists of states, transitions, events and activities. An *Activity diagram* is a special kind of the statechart diagram. It shows the flow of activities within the system. A *Component diagram* represents a static perspective of the system. It shows the organization and dependencies among the different components of the system. These components usually represent the system classes, interfaces or collaborations. A *Deployment diagram* shows a static perspective of the system run-time processing nodes configuration. Through these nine types of diagrams, UML presents a graphical language that can describe all aspects of the system whether static or dynamic[29].

---

[28] Booch et al, 1998, pp. 23-4.
[29] Booch et al, 1998, pp. 24-6.

3.3 Using Activity Diagrams to Model Geographic Processes

Activity diagrams in particular seem useful for describing geographic analysis processes in an abstract fashion. An activity diagram represents a dynamic sequential model of the system. It shows the flow of an object as its state changes due to the application of a certain activity or series of activities. Among the nine different diagrams available in UML, activity diagrams are the most suitable subset for modeling geographic processes. It models a dynamic aspect of the system by tracing the state of an object after the application of some activities. In this context an Activity Diagram will graphically describe the change of state of a map layer (or layers) through the application of a sequence of activities. An Activity represents a spatial function to be performed on the input layer(s) and would results in a change in the state of the layer(s).

3.3.1 Activity Diagrams Elements

Activity diagrams usually contain the following elements:

1. Action States
2. Activity states
3. Transitions
   - Sequential transitions
   - Branching
   - Forking and Joining

An *Action state* represents a process that occurs within the system. For example multiplying each value in a grid layer by an integer value resulting in a modified grid layer. An *Action state* is atomic and can not be further decomposed. An *Activity state* is similar to an action state but can be further decomposed. It represents another activity diagram. *Transitions* are used to pass control from one action or activity state to the next one. Simple sequential transitions move directly from one state to the next state. *Branching* is a kind of transition that occurs when there is a decision to be made. The next action or activity depends on the result of this Boolean decision. Forking and joining are other special kinds of transitions. They are used when there are several concurrent paths of actions or activities to be performed.[30]

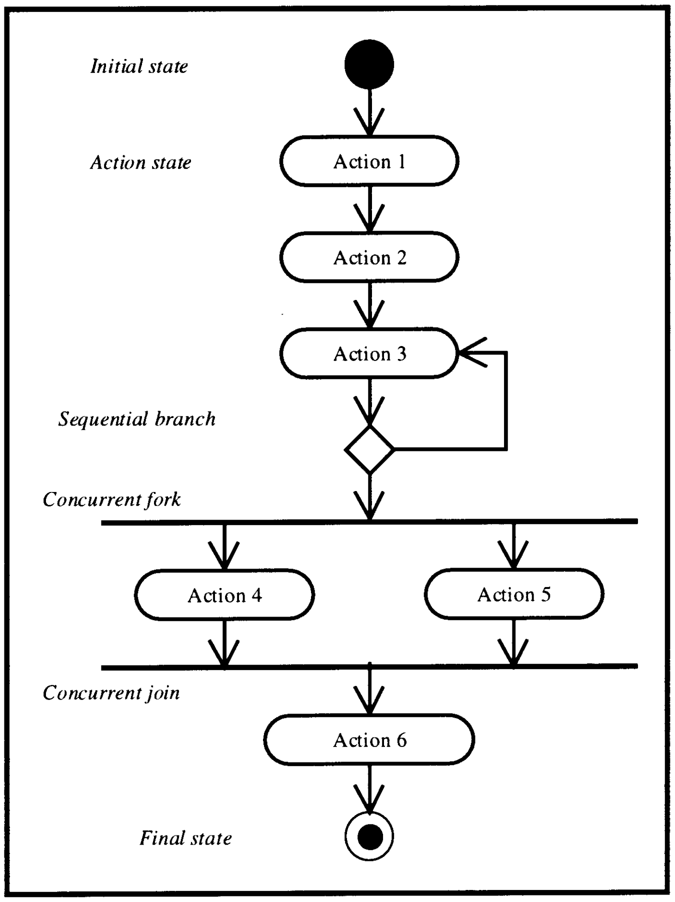


Figure 3.1: Elements of an Activity Diagram

[30] Booch et al, 1998, pp. 260-5.

3.3.2 An Example from the Urban Respiration Project

Having all the layers from the earlier Urban Respiration example integrated on the same GIS, they can be used to perform geographic analysis that attempts to understand the relationship between air pollution measurements and landuse, roads, weather, and other parameters. This example uses UML activity diagrams to design and model the analysis process.

The first part of the process includes building a model representing the relationship between road network and levels of certain trace-gases associated with auto-emissions. The hypothesis is that surface levels of these trace-gases are proportional to road class (highways, major roads, and local roads) and decrease proportionally as the distance to the road increase. Using the road layer, 4 grids are built each representing a road class. Grid cell values represent the distance to the closest road. It shows a value of 0 at the road and increases as you go away from it (which is the opposite of the desired relationship). Each grid represents the effect of a road class. The four grids are then added together using different weights emphasizing, for example, highways over local roads. By applying the operator 1/x on all the values in the grid, it reflects the desired pattern, proportionally decreasing values as the distance to the road increases. It shows a hypothetical pattern of the impact of "road density". This pattern needs to be calibrated and tested later. A possible way to calibrate it is by changing the weights used in adding the grids. These weights represent parameters that can be modified in several iterations to test several hypotheses about this relationship.

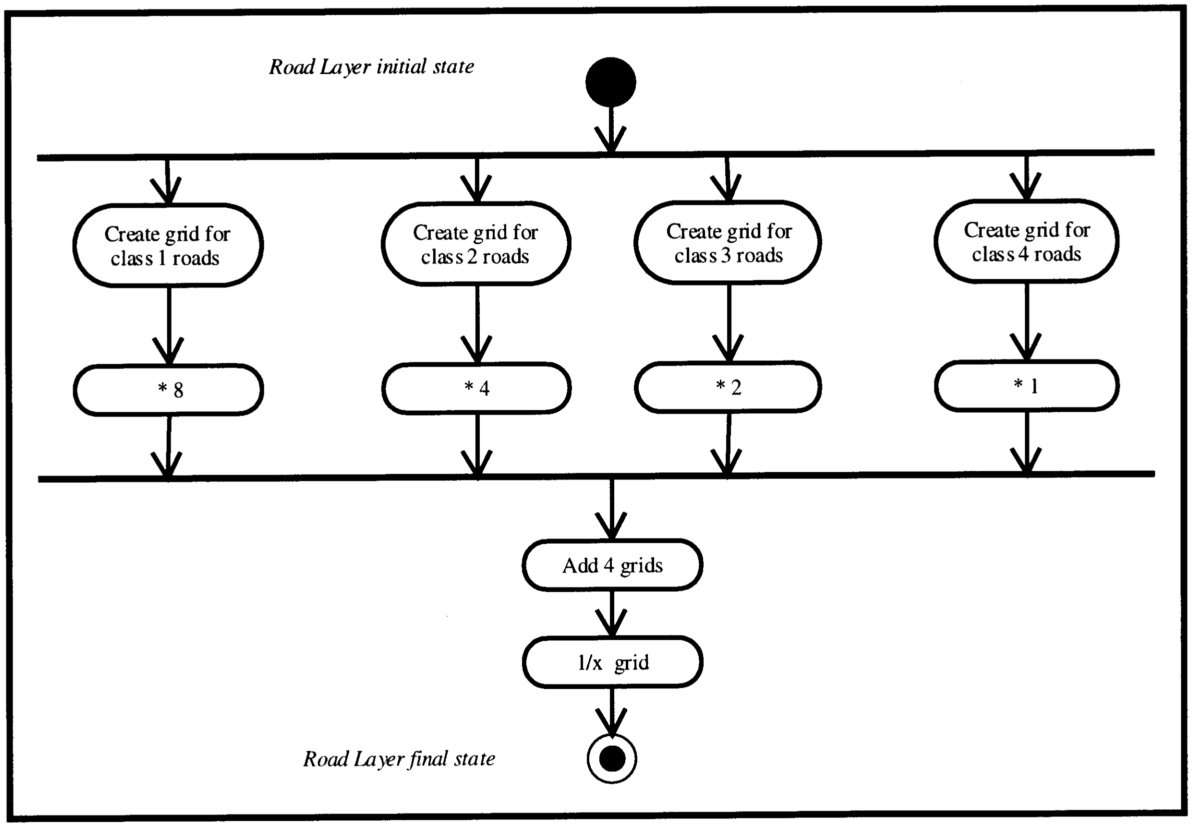

Road Layer initial state
Create grid for class 1 roads
Create grid for class 2 roads
Create grid for class 3 roads
Create grid for class 4 roads
* 8
* 4
* 2
* 1
Add 4 grids
1/x grid
Road Layer final state

Figure 3.2: UML Activity Diagram for the roads Model

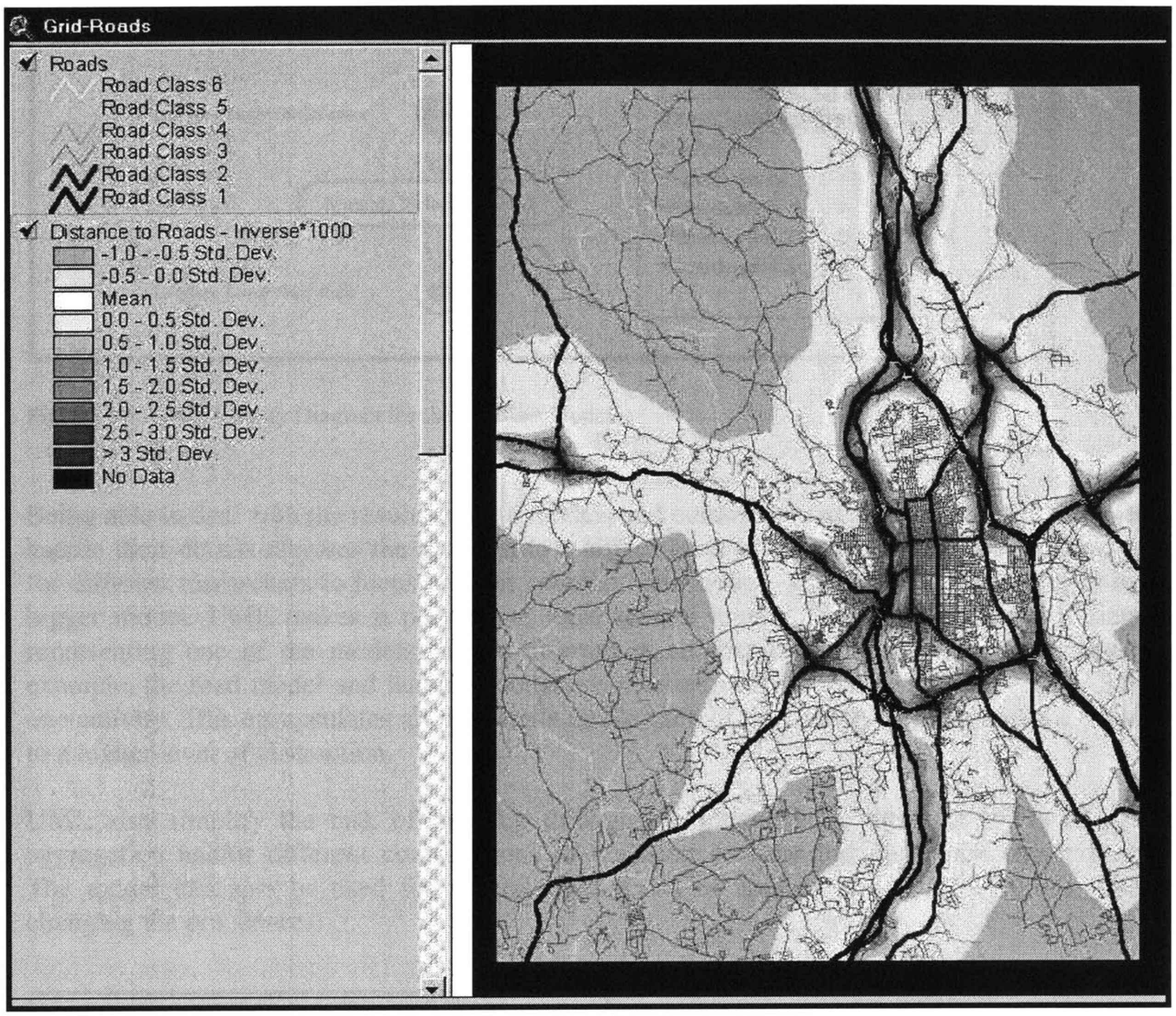


Figure 3.3: Raster grid for road model

Similarly, a model is built to represent a hypothetical relationship between landuse and levels of trace-gas emissions. Each landuse category (e.g. industrial, commercial, residential) is assumed to have a particular average level of emissions represented by a lookup table. A grid is created to represent this relationship. The values in the lookup table are parameters that can be used to verify and calibrate this relationship.

In the Urban Respiration project, other factors are also taken into considerations (weather and demographics) but are not described in this document. The final model represents an aggregation of all these different factors to create one grid showing the pollution concentration of trace gases. This model needs to be verified and calibrated using the real field measurements.

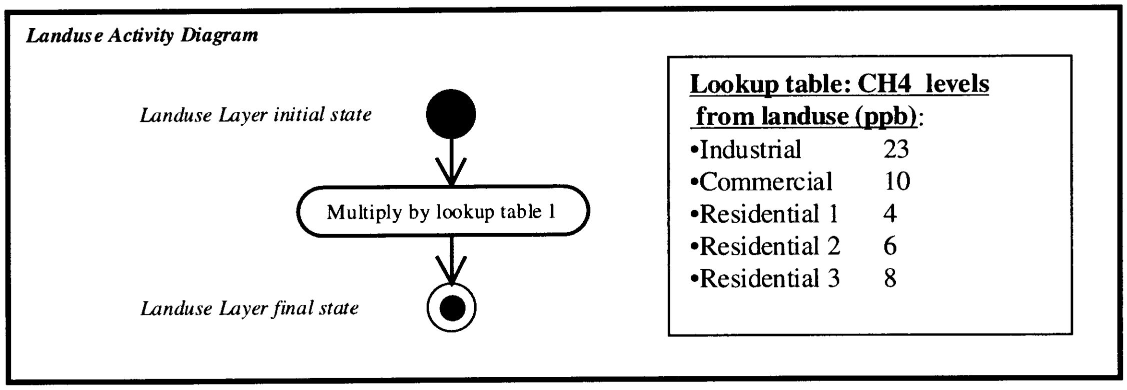

Landuse Activity Diagram
Landuse Layer initial state
Multiply by lookup table 1
Landuse Layer final state
Lookup table: CH4 levels from landuse (ppb):
•Industrial 23
•Commercial 10
•Residential 1 4
•Residential 2 6
•Residential 3 8

Figure 3.4: UML Activity Diagram for the Landuse Model

Being able to deal with the results of both models and combine them together without the need to handle their details elevates the problem to a higher level of abstraction and creates a platform for different researchers to focus on their specific contribution which can then be plugged in the bigger model. UML makes it possible to have several levels of *nesting*. An activity diagram representing one of the models can be inserted in another diagram as a single activity. For example, the road model and landuse model are inserted in the combined model (figure 3.5) as one activity. This encapsulates all the details inside each of the models and elevates the problem to a higher level of abstraction.

UML also simplify the task of iterating through the model using different levels of spatial aggregation and/or different combinations of emission sources and dispersion characteristics. The model can also be used for several trace gases or applied at several locations without changing the procedures.

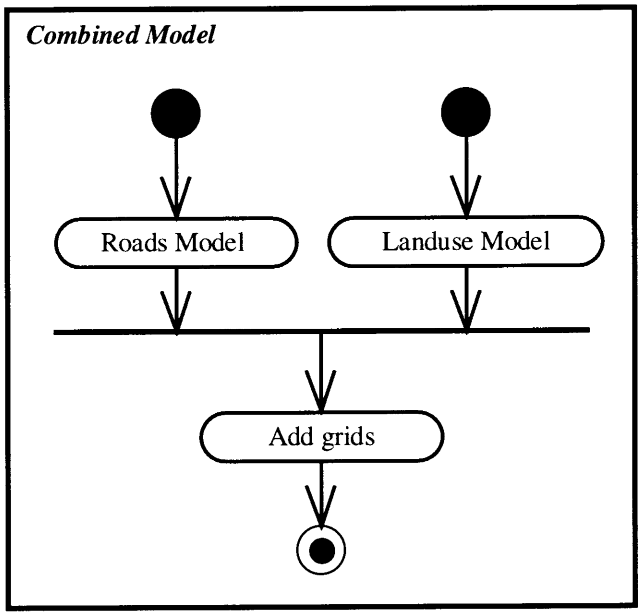


Figure 3.5: UML Activity Diagram for the combined roads and landuse model

4. System Architectures for Managing GIS Processing

As described earlier, a system architecture refers to the overall design of the system components and their interactions. This chapter provides a description for the desired system functionality and characteristics and the system components. Several alternative system architectures are then examined to select a prototype system architecture. This chapter examines the prototype architecture and implementation in more details.

4.1 System Functionality

The main idea of this project is to design and prototype a system architecture to manage the geographic analysis process in a distributed environment. The system should allow the users to:

- model the analysis process using an abstract modeling language such as the Unified Modeling Language, which allows the user to graphically represent, document, and communicate the analysis process in a standard language, and re-run it as a macro using different parameters and data sets,
- have access to spatial data repositories stored on different data servers in different formats,
- have access to different geo-processing services available on different application servers in a distributed environment, and
- provide a user-friendly client with a Graphical User Interface to manage this process.

4.2 System Characteristics

To achieve the previous design goals, the system architecture and prototype need to provide a modular, open, distributed and user-friendly design.

1. A modular design:
    - uses Component-based software development,
    - allows for adding or calling components (services) on demand,
    - integrates products from different vendors and platforms, and
    - can be customized to different users' needs.

2. An open design:
    - allows different software component to communicate using standard application programming interfaces (APIs),
    - allows for the easy development of new components that integrate seamlessly, and
    - is portable across different systems and platforms.

3. A distributed design:
    - is aligned with the current distributed organizational needs,
    - allows collaboration by sharing data & services, and
    - utilizes the Internet infrastructure.

4. A user-friendly design:
   - has a fast learning curve with lower training costs,
   - attracts a large user base, and
   - allows quick prototyping of an analysis model.

The implemented prototype satisfies some of these desired characteristics but not all of them. The final discussion will include an evaluation of the prototype and the implementation tools used, and their limitations.

4.3 Proposed System Architecture

To achieve the previous design goals and characteristics, three-tier client/server architecture with an application server is proposed.

There are four main components in this architecture:
1. The Client
2. The Application Server (Geo-processing Server)
3. The Database Server (Spatial Database Server)
4. The Communication Layer

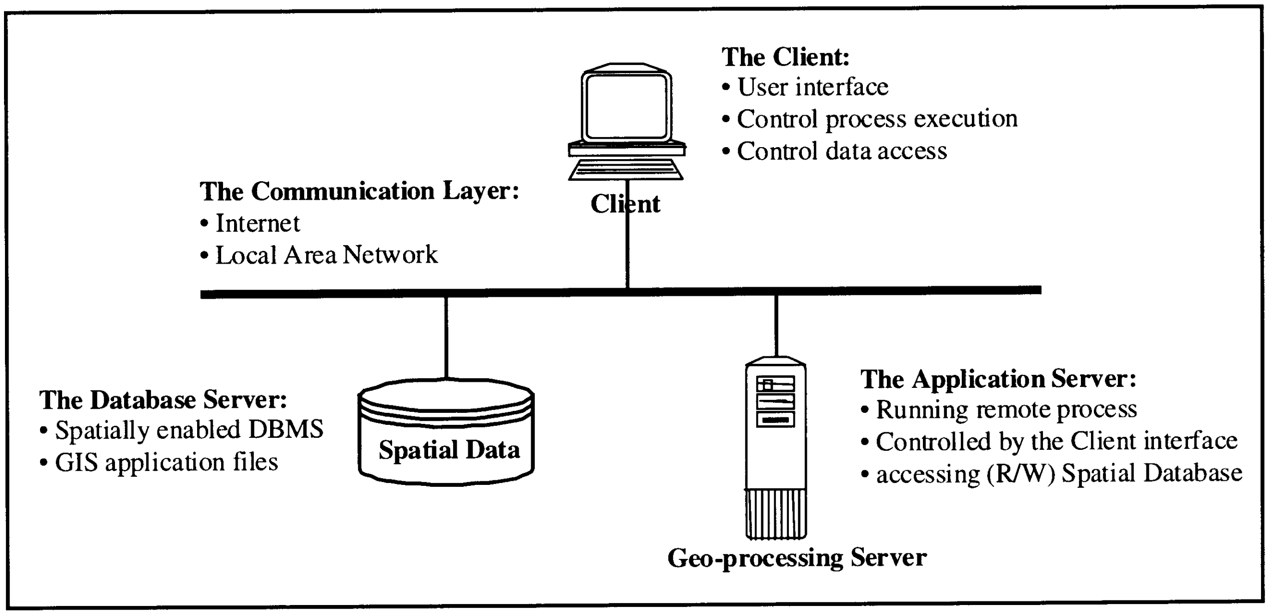


Figure 4.1: Proposed general system architecture

The *Client* represents the main module in the system. It contains the user interface and the modeling module. It allows the user to design a graphical model for the analysis process using UML and to display the associated map layers. It is also responsible for controlling the execution of the analysis using the geo-processing server and handling the database access on the different data servers.

The *Geo-processing Server* represents the application server that executes the geographic processes on geographically referenced data. The details of the server location and management are transparent to the user. Several servers could be available on the network and provide a range of geo-processing services on demand.

The *Spatial Database Server* represents a repository of geographically referenced data. The data could be stored either in a database system or as files. The details of the communication between the client and the data server are transparent to the user.

The *Communication layer* represents the network communication between the client, application server and database server. This layer should be transparent to the user.

4.4 Design Alternatives

Several design alternatives exist for each of the four main system components.

4.4.1 The Client

Two main alternatives are available for the implementation of the client prototype. The first, and probably most suitable to the above-mentioned criteria, is using a platform-independent component. An example is Java and ActiveX components.
The Java model is widely portable, flexible and Internet friendly. Java provides seamless access APIs to databases using JDBC. The drawbacks of using Java are the primitive user interface and the implied security model that restricts access to local data on the client. Another limitation is the slow execution due to the presence of the Java Virtual Machine as an extra interpreting layer. This is extremely important when handling massive geographic data. Some of these limitations still exist even if the prototype is developed as a Java application. ActiveX components have less security limitations but are restricted to using a Microsoft browser.[31]
The second alternative is to use a platform dependent development environment. This will allow for the full exploitation of the platform specific strengths and access to system resources, while limiting the portability of the client application. It also allows for the development of a more sophisticated Graphical User Interface.

4.4.2 The Geo-processing Server

A geo-processing server is an application server that is capable of providing geographic services. A geographic service is a computational process applied on geographically referenced data. In a client/server environment, a geo-processing server would receive a request to perform a geographic service. The required data can be either sent as a stream to the server or sent as a query from another database server. An example is the MIT Orthophoto browser. The server is sent a request to provide a specific image layer. The coordinates of the required image are sent as a stream of data to the server. The server accesses an image data repository on a data server, performs the process of preparing the required image and then sends it in the required format[32].

[31] www.javasoft.com, www.microsoft.com
[32] ortho.mit.edu, Evans.

4.4.3 The Spatial Database Server

Spatial data can be stored and managed in two different ways. The first way is to store the data in an ordinary file system, usually using proprietary formats (e.g. shape files). This is only practical with a small amount of data or a limited number of users. Currently most spatial data is stored in this way resulting in many data management problems.
The second way is to use a spatially enabled database Server. Some relational database vendors have started to provide extensions that allow for improved storage of spatial data within the database server. Using Oracle Spatial Cartridge for example, spatially referenced data can be stored, indexed and accessed through Oracle database[33]. Other vendors are planning to support similar configurations.

4.4.4 The Communication Layer

Operating in a distributed environment requires handling several layers of complexity in terms of communication among different platforms using different protocols. The objective of this prototype is to avoid these layers of complexity and provide the user with a simple transparent access to the data and geo-processing servers.

Using a distributed object model like Microsoft's COM (Component Object Model), Sun Microsystems' JAVA Beans or OMG's CORBA (Component Object Resource Broker) provides a possible solution[34]. In this model the application is divided into small components that reside on different machines and communicate with each other using a resource broker. This three-tier client/server architecture with an object resource broker is useful when developing a new environment. It is not applicable when integrating several vendor-specific environments since current GIS software is not designed to work with these emerging architectures in a non-proprietary way.

Another possibility is by invoking Remote Procedure Calls. RPCs provide a mechanism for communication among different applications. It is based on the concept of procedural programming where a procedure is called to perform a specific function. It is passed in parameters and returns back modified values. RPC extends this concept from within one application to among different applications. Sun Microsystems' implementation of RPC allows for communication among different platforms and has long been supported on UNIX platforms.

[33] www.oracle.com

[34] www.microsoft.com, www.javasoft.com, www.omg.org

4.5 Implemented System Architecture

Selecting from the previous alternatives to develop a system architecture for the prototype implementation is a challenging task. Some of the alternatives provide better performance but are not easy to implement. The following architecture was selected for the prototype implementation:

The *client* is developed in a win32 environment using Visual C++. Although a platform specific implementation limits the use of this prototype, it allows for the full utilization of a powerful set of programming tools and graphical user interface. The next decision was to use Arc/Info as an *application server* using RPC as a *communication layer*. The choice was between using a well-developed commercial package as an application server or building a limited number of geographical services from scratch. The former alternative provides a powerful library of geographic services, but restricts the *data* format to Arc/Info's proprietary file format. The later alternative severely limits the services available in the prototype. Running Arc/Info as a server implies using RPC at the communication layer. Arc/Info Inter-Application Communication (IAC) interface allows Arc/Info to receive calls from other applications and executes them locally. The main limitation is that the data involved in such a call must reside in a network location accessible to the Arc/Info server. It also must be in a location accessible to the client through a mapped network drive in order to display the processed layers. RPC masks several layers of complexity from the user and treats the remotely controlled program as a local extension of the client. Arc/Info and Arc/View accept RPCs that allow for the exploitation of their functionalities. Other custom libraries could be compiled to accept RPCs with a few modifications.

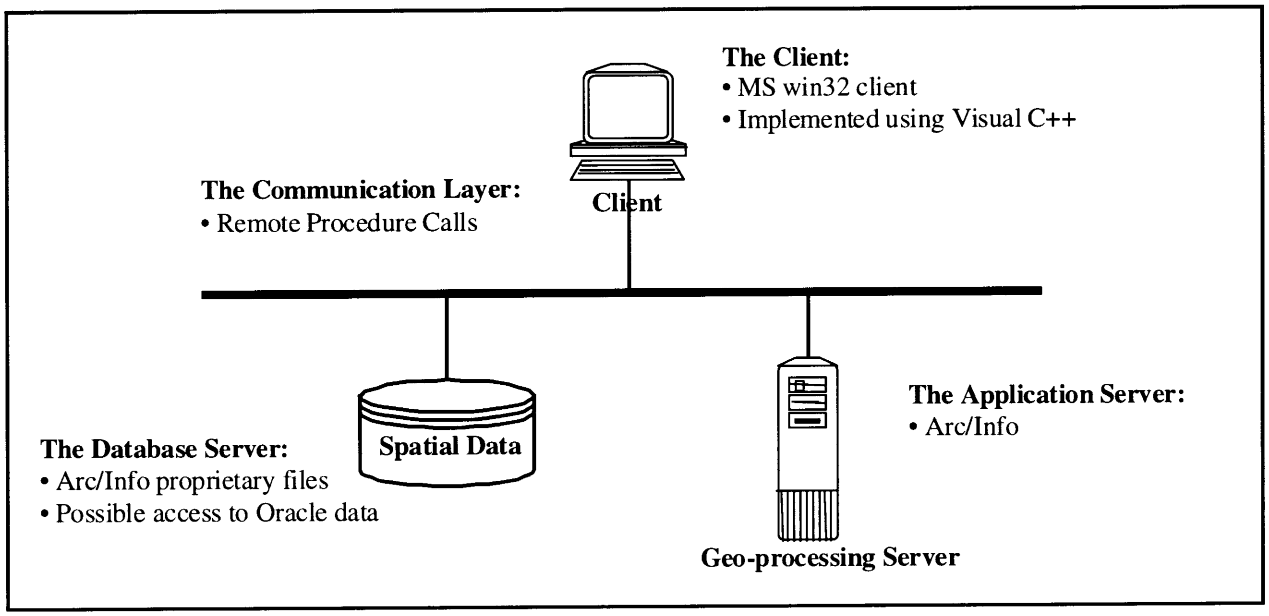


Figure 4.2: Implemented system architecture

5. Prototype Implementation

A prototype was implemented on a win32 platform using Visual C++ to illustrate and test some of the previous concepts and system architecture. This chapter describes the prototype implementation. Next chapter uses some of the examples discussed earlier from the Urban Respiration project to illustrate how the prototype functions, and to highlight some of the implementation issues and tradeoffs.

5.1 The Client Application

The client application (named Geo) contains three main components. The first view contains a tree listing of all the map layers in the current workspace. The second view contains a UML diagram. The third view contains a map display.

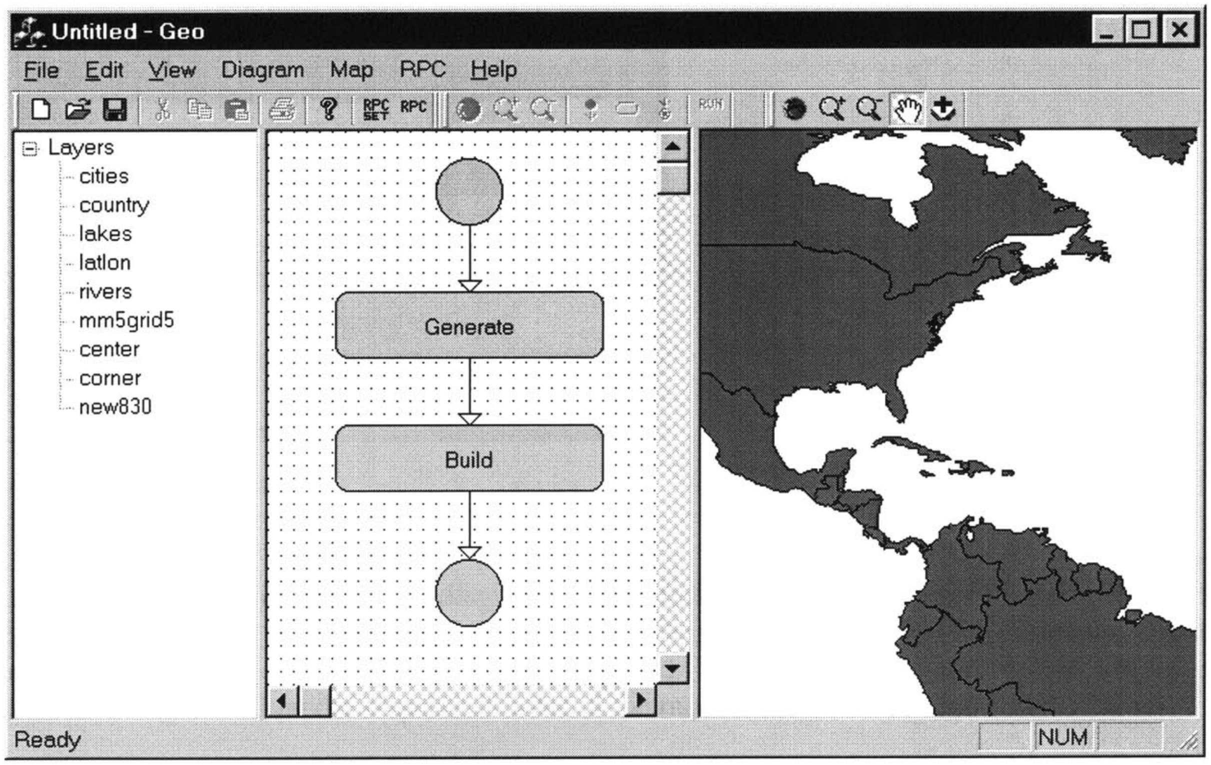
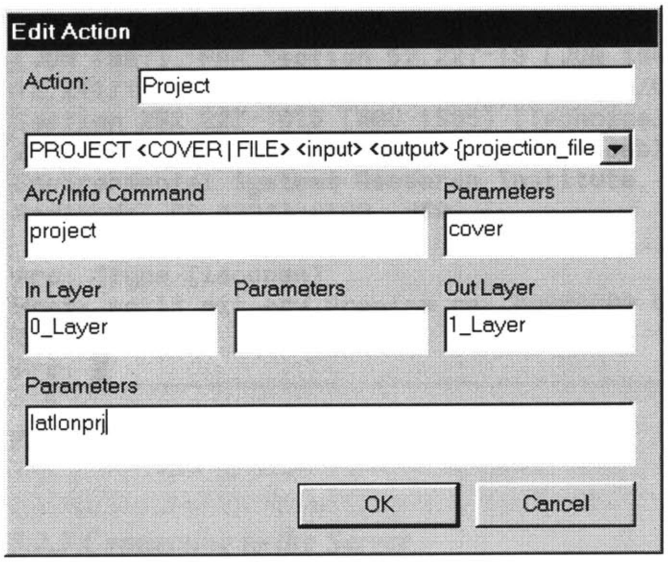


Figure 5.1: Client screen

5.2 Using the Client

5.2.1 Creating the Model

Using the Diagram view, the user can design the UML model for the analysis. The prototype implementation is limited to a single path linear model starting with an initial node that refers to the initial layer, then adding several actions each implementing an Arc/Info command, and ending with a final state which refers to the output layer of this model. The model is executed sequentially. The output layer of each action is considered as the input layer for the following actions. Since Arc/Info uses a proprietary format, at the end of model run all the layers used (initial, intermediary and final) are converted to a shape file format, which is viewable through the map view.


Edit Action

Action: Project

PROJECT <COVER | FILE> <input> <output> {projection_file

Arc/Info Command: project

Parameters: cover

In Layer: 0_Layer

Parameters:

Out Layer: 1_Layer

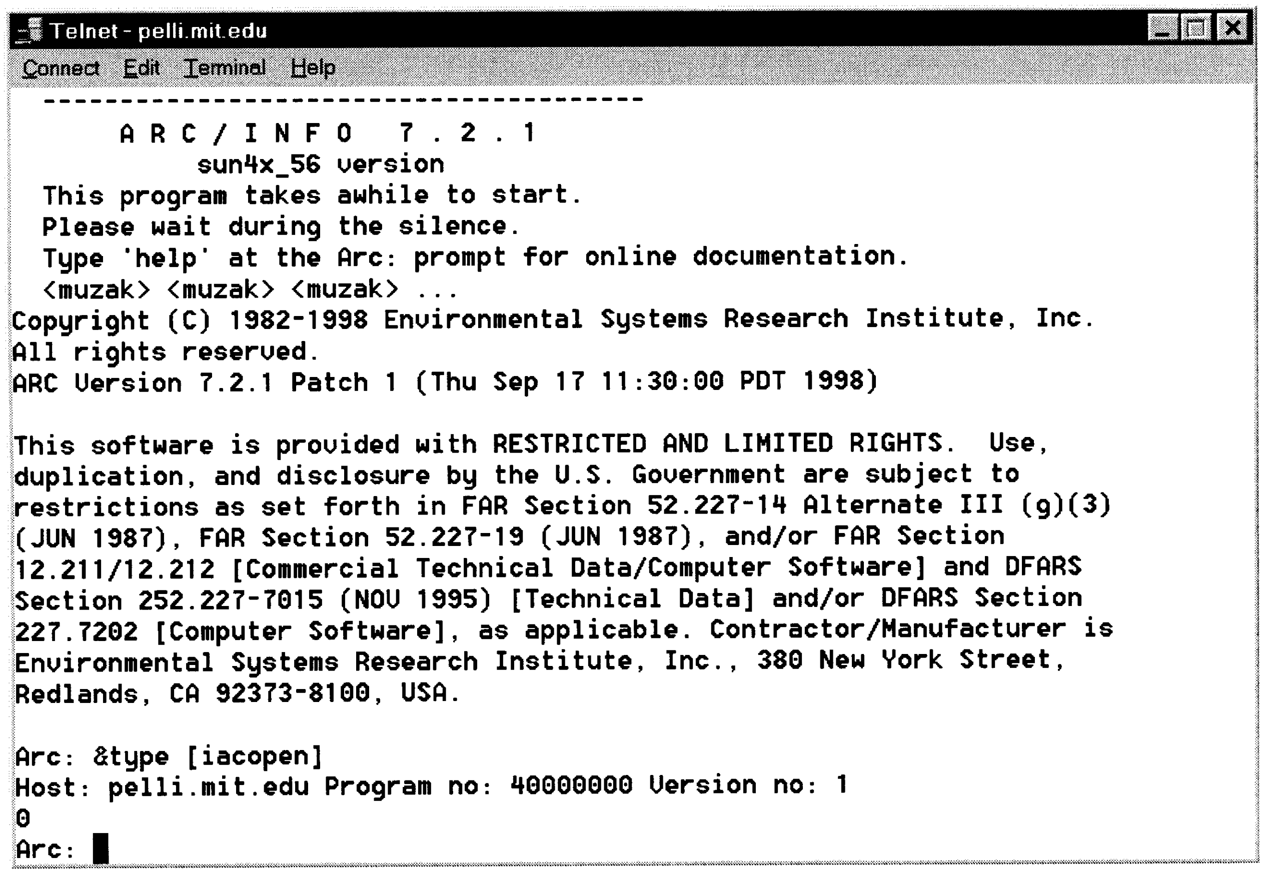
Parameters: latlonprj

Figure 5.2: Action Dialog

By double clicking on any of the diagram components, the user can change the parameters of this specific component from the Edit Action dialog. The parameters contain the command, input and output layers, and any other parameters required. The input and output layers are assigned automatically to follow the sequential order of the model.

5.2.2 Configuring the Server

Before the model can be run, the Arc/Info server needs to be opened and connected to the client. To open the Arc/Info server, run Arc/Info and then open the AI-IAC (Arc/Info Inter-Application Connectivity). Arc/Info responds by giving the server name, program number and version number. These three parameters are required for the client to correctly connect to this Arc/Info IAC session (note that the numbers are in hexadecimal). Arc/Info remains available for the local user as well as the remote user. It maintains a queue of commands entered. If the Arc/Info session is closed, the client connection to the server is lost.

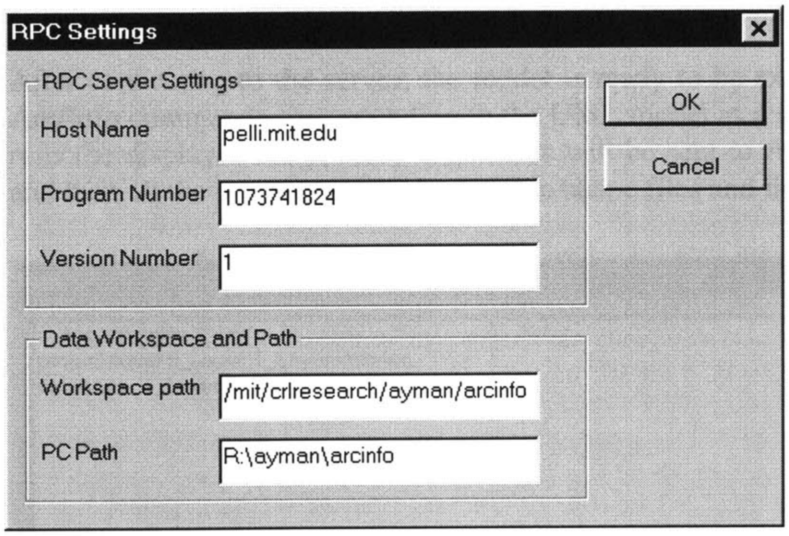


```
  --------------------------------------
       A R C / I N F O   7 . 2 . 1
            sun4x_56 version
  This program takes awhile to start.
  Please wait during the silence.
  Type 'help' at the Arc: prompt for online documentation.
  <muzak> <muzak> <muzak> ...
Copyright (C) 1982-1998 Environmental Systems Research Institute, Inc.
All rights reserved.
ARC Version 7.2.1 Patch 1 (Thu Sep 17 11:30:00 PDT 1998)

This software is provided with RESTRICTED AND LIMITED RIGHTS.  Use,
duplication, and disclosure by the U.S. Government are subject to
restrictions as set forth in FAR Section 52.227-14 Alternate III (g)(3)
(JUN 1987), FAR Section 52.227-19 (JUN 1987), and/or FAR Section
12.211/12.212 [Commercial Technical Data/Computer Software] and DFARS
Section 252.227-7015 (NOV 1995) [Technical Data] and/or DFARS Section
227.7202 [Computer Software], as applicable. Contractor/Manufacturer is
Environmental Systems Research Institute, Inc., 380 New York Street,
Redlands, CA 92373-8100, USA.

Arc: &type [iacopen]
Host: pelli.mit.edu Program no: 40000000 Version no: 1
0
Arc: █
```

Figure 5.3: Opening Arc/Info Server RPC

5.2.3 Connecting to the Server

On the client side, the RPC connection needs to be configured. The RPC settings dialog box contains the Arc/Info server settings and the workspace settings. The host name, program number and version number must be identical to the ones in the Arc/Info IAC response (note that numbers are in decimal and not hexadecimal).



RPC Server Settings

Host Name: pelli.mit.edu

Program Number: 1073741824

Version Number: 1

Data Workspace and Path

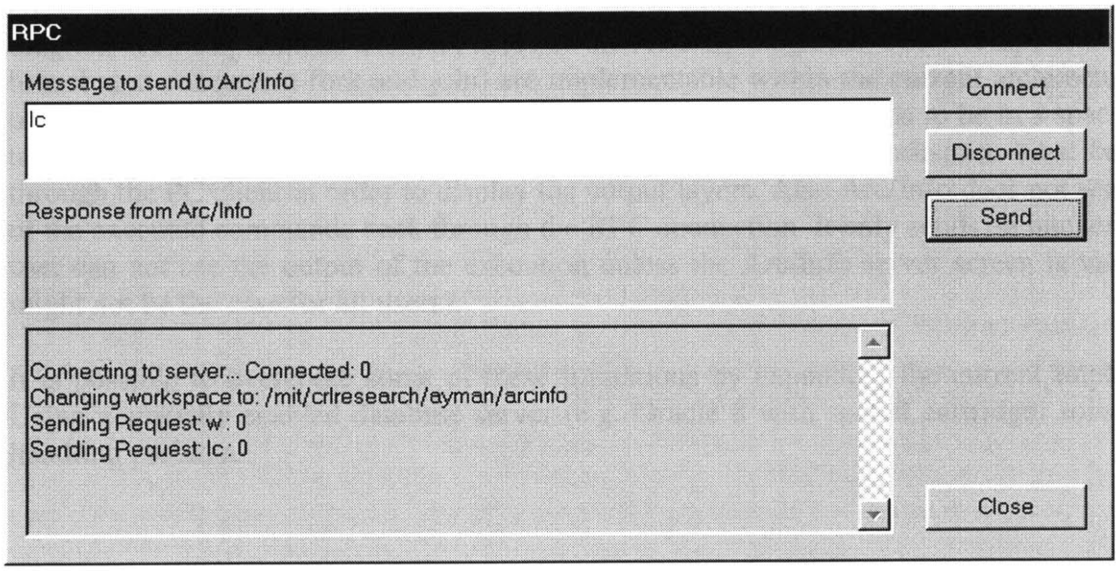
Workspace path: /mit/crlresearch/ayman/arcinfo

PC Path: R:\ayman\arcinfo

Figure 5.4: Configuring the client RPC connection settings

The data workspace represents the workspace that Arc/Info uses to access the data layers and to store temporary files. This workspace must be accessible to Arc/Info as read/write, and must also be accessible to the PC client as a mapped network drive. This way Arc/Info can use the data and the output can be displayed in the client window.

The next step is to connect to the Arc/Info server and test the connection through the RPC Dialog. If the connection is successful the Dialog displays a "Connected" message and changes the workspace to the one indicated in the settings. Through the top screen any messages can be passed to Arc/Info through the RPC connection. The output of these messages is displayed on the server screen. Only the status (0=successful, otherwise error) is passed back through RPC. After connecting to the server, the client is ready to run the model.



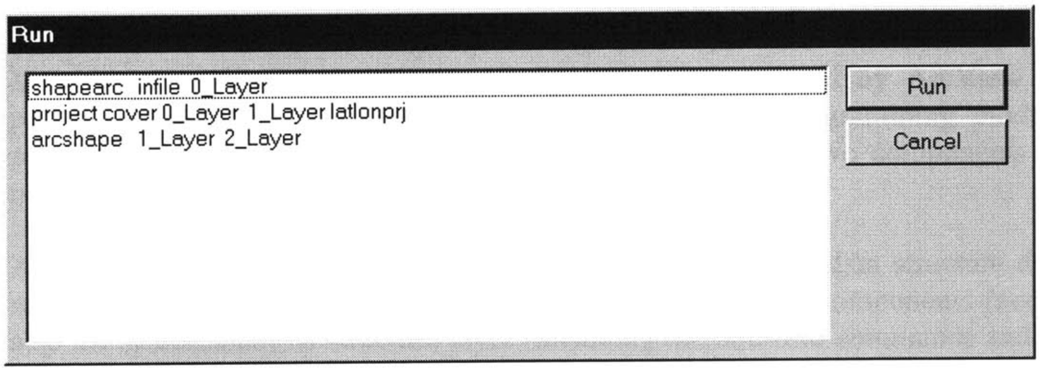
Message to send to Arc/Info: lc

```
Connecting to server... Connected: 0
Changing workspace to: /mit/crlresearch/ayman/arcinfo
Sending Request: w : 0
Sending Request: lc : 0
```

Figure 5.5: Connecting to and testing Arc/Info Server RPC

5.2.4 Running the Model

After connecting to the server, the model is ready to be executed. The Run Dialog displays the Arc/Info commands corresponding to the UML model as a script for final verification. If the user runs the displayed script these commands will be sent to Arc/Info through the RPC connection, and then all the layers will be converted to shape files and listed in the tree view.



```
shapearc  infile 0_Layer
project cover 0_Layer 1_Layer latlonprj
arcshape  1_Layer 2_Layer
```

Figure 5.6: Running the model

5.2.5 Other Functions

Some of the other functions available in the prototype are full extent, zoom and pan for both the diagram view and the map view. The user can also add any background map layers to the map view from any other accessible workspace.

5.3 Prototype Limitations

It is important to differentiate between the limitations of the prototype that are due to time constraint and those due to limitations in the implementation tools and system architecture. The diagram view implements a small subset of the Activity diagrams, other components (sequential branch and concurrent fork and join) are implementable within the current architecture. The use of Arc/Info as a server implies a set of restrictions. All the data needs to be in a space accessible to the Arc/Info server and in its proprietary format. The same workspace must be accessible through the PC client in order to display the output layers. Also Arc/Info does not send the result of the executed commands back through the RPC connection. It only sends an integer status. The user can not see the output of the execution unless the Arc/Info server screen is visible (which might not be the case for all users).

It is possible to overcome some of these limitations by expanding the current implementation. Using a spatially enabled database server (e.g. Oracle 8 with spatial cartridge) solves the data-handling problem.

5.4 Evaluation of the Prototype Implementation

The design was implemented using Visual C++ on win32 environment. On one hand, Visual C++ requires a very long learning curve as it restricts the programmer with a rigid framework (Microsoft Foundation Classes). On the other hand, it offers a very powerful rapid development visual environment while still maintaining the strength of C++. It allows the use of ActiveX components, which could speed the development time, by using components that could be added directly to the interface. MapObjects and Add Flow are examples of such components. MapObjects makes it possible to display map layers created by ArcView (shape files) and perform some spatial analysis on them. AddFlow is a diagramming module that makes it possible to create and manage diagrams and charts. These two components were used in the prototype for the map display and UML diagram.

A major challenge in the design of the prototype was in the data structure design. The design requires a data structure that can capture the diagram and its components (nodes and links) and also the spatial analysis structure layer (including the Arc/Info commands and parameters). This data structure is highly dependent on the application server and the data model of the spatial data. Having an interoperable data model for the spatial data would simplify this problem.

In general, the main limitations during the implementation phase were on the level of application and data interoperability. It is very hard to communicate among different GIS applications. This is made even harder by the lack of a common data model.

Client Implementation

Implementing the client prototype using Visual C++ on win32 platform offered access to a powerful Application Programming Interface (API) and Graphical User Interface (GUI). As it stands, the prototype is can not be easily ported to other platforms without involving a lot of programming. This does not present a big restriction. Currently win32 is the most common operating platform for desktop computers. The important issue is to be able to access a variety of data servers and application servers on other platforms. The use of win32 for the client does not present any major restrictions on this side.

Server Implementation

Arc/Info was used as a geo-processing server. This is an example of using a current application without any major modification as an application server. It offers a powerful collection of spatial analysis libraries for a wider user base. The main limitation comes at the data access level. Files have to be used in a shared workspace in the proprietary Arc/Info file format. This model is useful in a LAN setting with a high bandwidth. It is not useful in an Internet setting. Users must have access to the shared workspace, and need to handle security and file access issues at a very low level. This is not the intention of the designed architecture. The nature of geographic data and topology does not allow for the application server to be fed small data objects and process them sequentially. It requires the whole data set to be handled simultaneously and passed as one contingent data set. Even if a spatially enabled database management system is used, it is not possible to separate the whole data set from the application server. The Database server has to be responsible for performing the spatial queries and thus acting as an application server also.

Communication Layer Implementation

In this prototype, Remote Procedure Calls were used as the main communication protocol. RPC succeeded in masking most of the communication complexities of working on multi platforms. Most of the problems of communication between the client and the application server occur at this level. For new GIS applications to be able to integrate with such a distributed architecture they need to support standards like RPC, CORBA, COM, etc. If this were the case then it would be possible to pass different packets of data between the client and different applications.

6. Conclusion

The main goal of this research was to learn how to manage the analysis and synthesis of geographic data to support collaboration among different researchers, planners, and engineers working on complex spatial analysis problems. The first research phase examined the abstraction and representation of the analysis process using the Unified Modeling Language. The second phase examined distributed environments to enable such collaboration, using the Urban Respiration project for its context and examples; and developed a prototype application to test and understand the applicability of the proposed designs.

6.1 Modeling of Geographic Processes

UML activity diagrams proved to be a useful conceptual tool for modeling geographic processes. UML offers a standard communication language that transcends the different disciplinary boundaries. This enables and supports collaboration among the different researchers, planners and engineers. The urban respiration project example showed how a UML activity diagram could encapsulate the details of the application-dependent implementation. This focuses the researcher's attention on the conceptual problem, which could later be implemented in any application. The roads and landuse example illustrated UML's ability to approach the problem at several levels of abstraction using nesting, and to combine the results of different components at a later stage. As GIS software moves towards Object-Oriented implementations, it will become easier to translate the conceptual analysis process from UML directly to a specific application. This research focused on the use of activity diagrams to model the analysis process; further research could examine the applicability of other diagrams and elements in geographic analysis context.

6.2 Distributed System Architecture

The proposed distributed system architecture is in line with current trends in technology, but proved not to be fully implementable using current software and communications tools. In the GIS context, a three-tier system architecture promises to offer geographic services to a wider audience that is traditionally unable to handle the complex task of managing such a system. The data management side is handled by a database management system, which relieves the user from data management details and provides a better institutional framework for handling data sources. Conceptually, geo-processing servers fits within the current Internet infrastructure and can make GIS services openly accessible to a wider user group. Moving the data and topology to a spatially-enabled database server overcomes the problem of passing large data files. Smaller data objects can be sent to the geo-processing server, processed, and the response sent back to the client. This might be more suitable for vector data where the topology can be stored on the database server (compared to raster data that is hard to manage using the database model). Emerging commercial technologies (Oracle Spatial data cartridge and ESRI Spatial Database Engine) might make this distributed model more feasible within the near future. Further research could examine the applicability of these emerging tools in implementing a similar architecture.

6.3 Prototype Implementation

The prototype implementation revealed the importance of several related issues: interoperability, metadata and standards. By design, current GIS software is based on closed systems making it hard to communicate among the different software or exchange data directly without processing and conversion. Further more, geographic data sets usually do not have their metadata encapsulated in their files. This creates the problem of trying to understand the nature of the data at hand. The lack of standards for metadata and inter-process communication makes it hard for GIS applications to communicate at both levels. The prototype implementation illustrates the great need for standards that enable GIS applications to communicate and exchange data seamlessly. This is a prerequisite for any major advancement of distributed GIS. Further research in the area of GIS interoperability could build on current standards in distributed systems that are widely adopted in industry and create extensions that allows GIS data and applications to be part of the mainstream information systems. For example, efforts to embed spatial referencing information using the Extended Markup Language (XML) may permit improved GIS interoperability using emerging industry standards for communication and distributed processing.

Fully implemented, the envisioned architecture would provide a platform to support engineering modeling and analysis. Using UML as an abstract modeling language would provide a software-independent and discipline-independent vocabulary that could also encapsulate implementation details. The urban respiration project example illustrates how a geographic analysis could be modeled independently of any specific implementation software, hardware, or networking. The model can be communicated and shared among researchers from different disciplines, thus reducing the need to discuss details of *how* the model is implemented, and focusing on the issues of *what* analysis is to be performed. The distributed environment provides the underlying infrastructure for such collaboration. Having the data maintained by each party through a database server removes the hurdles of sharing and managing the data. Each party could develop their part of the model and share the results with the rest of the collaborators. Each component (for example the weather model or the atmospheric chemistry model) can be implemented and managed by one group, while being accessible to the rest of the collaborators through the distributed environment. For example, the weather models are complex and demanding, but the output can readily be queried and extracted in standard forms to provide the wind velocity and temperature estimates needed for the atmospheric chemistry model. Additional components could be seamlessly added to the UML model without the need to modify the rest of the processes. For example, trace-gas emissions from new industrial point sources could be "inserted" seamlessly into the model via the activity diagram, and the aggregation and integration steps could be re-run to generate new estimates of the surface level distributions. Having access to geo-processing servers would make it possible to use a few GIS functions without having to install and manage a full system within each research lab. For example, inserting a geo-referenced digital orthophoto as a background to Manchester model output layers could reveal important insights about how to interpret the analysis. Similarly, being able to perform coordinate transformation through a server would solve problems of moving across different geographic projection systems.

6.4 Impact on GIS Users

From a user perspective, expressing and managing the analysis process on a conceptual level makes geographic analysis more appealing and accessible.

The current model is based on stand-alone computing, where all the analysis is performed on a single machine (although some data might reside in a mapped network drive). Collaboration is done by communicating files either through the Internet (using ftp protocol) or by physically exchanging the storage media. Integrating engineering and planning models proves not to be a trivial task either on the conceptual level of implementation level.

The designed system architecture provides a platform for collaboration. This is illustrated using the Urban Respiration example. The user frames the problem in a diagram using a standard language, utilizes different data sets available through the distributed environment, implements the model through geo-processing servers at different locations and from different vendors. The model can be modified and iterated through to test possible hypotheses. This makes complex spatial analysis accessible to a wider range of users.

6.5 Choices in Design and Implementation

The design and implementation of the prototype application involved making many choices about the approach and technology, which affected the conclusions of this research. Examining these choices reveals the complex interdependent nature of the problems addressed.

Modeling of geographic processes could be captured in several ways, for example, using a text macro or non-standard diagramming. As this research takes the approach of building on existing standards, it examined possible ways of capturing this process using a standard language. Using a non-standard graphical language is still a possible option, but would lack many of the features that UML offers. The issue of whether UML supports all the functions needed to model geographic processes requires further research.

The prototype implementation revealed a major problem with using an abstract modeling language. The model represented in UML describes geographic processes using simple diagrams that encapsulate the details. To run this abstract model, these processes have to be translated into implementation-specific commands and scripts. Different GIS applications perform similar commands in a conceptually different manner, for example, creating a buffer grid using Arc/Info requires a different procedure and parameters than, say, MapInfo or ArcView. The translation of the abstract model into software-specific commands requires a mapping layer specific to each software. The prototype implements this layer at the client side. Thus, each element in the activity diagram includes all the Arc/Info commands and parameters. Other options include having this layer as a set of scripts at the server side. This example illustrates the need for a common way to describe spatial analysis in a more abstract way that is software-independent.

The road model example from the Urban Respiration project uses raster grids created from the road vector layer. In the proposed system architecture, the choice of the data type affects the way it is managed and stored. Vector data can be stored either in file systems or DBMS, while spatially-referenced raster data are not yet efficiently stored in commercially available DBMS. The proposed system architecture favors a DBMS as a more efficient way for data management, while the implemented prototype uses files due to software limitations. This choice affects the types of data that this architecture can handle and the efficiency of the data management.

The previous examples highlight the complexity of these choices, which are often open-ended, interdependent and lack a clear solution. The rationale behind adopting a specific path is usually dependent on the application context.

6.6 General Discussion

This research examined ways to improve collaboration in multidisciplinary research involving complex spatial analysis problems. Modeling of spatial analysis processes proved to be a promising approach to facilitate collaboration and provide an abstract representation of complex problems. Apart from the distributed environment discussed in this research, the use of UML (or other abstract modeling tools) could enhance problem solving and collaboration. This could be implemented within the framework of current applications. For example, in addition to looking at a collection of layers as a map view, set of tables or cartographic layout, a UML view would provide information on the process of generating them. This could reflect on the way GIS users approach spatial analysis problems. Iterating through the analysis process to test several hypotheses could enhance the decision making process in issues that involve complex spatial problems.

The proposed distributed system architecture illustrates the importance of improving accessibility to information through open systems and interoperability. Data collection is one of the most expensive and time-consuming tasks in building Geographic Information Systems. The ability to have the current repositories of data accessible to a wider audience would remove one of the obstacles for the proliferation of GIS applications.

This research demonstrated how to manage the analysis and synthesis of geographic data to support collaboration among different researchers, planners, and engineers working on complex spatial analysis problems. The examples from the Urban Respiration project illustrate the importance of collaboration across disciplines and current ways of such collaboration. Using the Unified Modeling Language as an abstraction language for modeling geographic processes proved to be a promising approach.

Appendix: Sample Visual C++ Code

Class: Cdiagram View Header File:

```
//{{AFX_INCLUDES()
#include "addflow.h"
//}}AFX_INCLUDES
#if !defined(AFX_DIAGRAMVIEW_H__7C7C1DC5_E532_11D2_88B1_83D3FDB4CD37__INCLUDED_)
#define AFX_DIAGRAMVIEW_H__7C7C1DC5_E532_11D2_88B1_83D3FDB4CD37__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DiagramView.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CDiagramView form view

#ifndef __AFXEXT_H__
#include <afxext.h>
#endif

// ayman
#include "MapView.h"

#include "node.h"	// Added by ClassView
class CDiagramView : public CFormView
{
protected:
	CDiagramView();		// protected constructor used by dynamic creation
	DECLARE_DYNCREATE(CDiagramView)

// Form Data
public:
	//{{AFX_DATA(CDiagramView)
	enum { IDD = IDD_DIAGRAM_FORM };
	CAddFlowm_diagram;
	//}}AFX_DATA

// Attributes
public:
	CGeoDoc* GetDocument();
	CMapView* pMapView;

// Operations
public:
	CNode prevNode;

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CDiagramView)
	public:
	virtual void OnInitialUpdate();
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	virtual ~CDiagramView();
#ifdef _DEBUG
	virtual void AssertValid() const;
	virtual void Dump(CDumpContext& dc) const;
#endif

	// Generated message map functions
	//{{AFX_MSG(CDiagramView)
	afx_msg void OnSize(UINT nType, int cx, int cy);
	afx_msg void OnDiagramFullextent();
	afx_msg void OnDiagramInsertindiagramActivity();
	afx_msg void OnDiagramInsertindiagramConcurrentfork();
	afx_msg void OnDiagramInsertindiagramConcurrentjoin();
	afx_msg void OnDiagramInsertindiagramFinalstate();
	afx_msg void OnDiagramInsertindiagramInitialstate();
	afx_msg void OnDiagramInsertindiagramSequentialbranch();
	afx_msg void OnDiagramRun();
	afx_msg void OnDiagramSaveasactivity();
	afx_msg void OnDiagramZoomin();
	afx_msg void OnDiagramZoomout();
	afx_msg void OnDblClick();
	afx_msg void OnDiagramInsertindiagramAction();
	DECLARE_EVENTSINK_MAP()
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};
#ifndef _DEBUG  // debug version in GeoView.cpp
inline CGeoDoc* CDiagramView::GetDocument()
  { return (CGeoDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_DIAGRAMVIEW_H__7C7C1DC5_E532_11D2_88B1_83D3FDB4CD37__INCLUDED_)
```

Class: Cdiagram View C++ File:

```
// DiagramView.cpp : implementation file
//

#include "stdafx.h"
#include "Geo.h"

#include "GeoDoc.h"
#include "DiagramView.h"
#include "MyTreeView.h"
#include "Action.h"
#include "ActionDlg.h"
#include "DiagramRunDlg.h"

#include "arcrpc.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CDiagramView

IMPLEMENT_DYNCREATE(CDiagramView, CFormView)

CDiagramView::CDiagramView()
	: CFormView(CDiagramView::IDD)
{
	//{{AFX_DATA_INIT(CDiagramView)
	//}}AFX_DATA_INIT
}

CDiagramView::~CDiagramView()
```

```
{
}

void CDiagramView::DoDataExchange(CDataExchange* pDX)
{
	CFormView::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CDiagramView)
	DDX_Control(pDX, IDC_ADDFLOWCTRL1, m_diagram);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDiagramView, CFormView)
	//{{AFX_MSG_MAP(CDiagramView)
	ON_WM_SIZE()
	ON_COMMAND(ID_DIAGRAM_FULLEXTENT, OnDiagramFullextent)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_ACTIVITY, OnDiagramInsertindiagramActivity)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_CONCURRENTFORK, OnDiagramInsertindiagramConcurrentfork)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_CONCURRENTJOIN, OnDiagramInsertindiagramConcurrentjoin)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_FINALSTATE, OnDiagramInsertindiagramFinalstate)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_INITIALSTATE, OnDiagramInsertindiagramInitialstate)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_SEQUENTIALBRANCH, OnDiagramInsertindiagramSequentialbranch)
	ON_COMMAND(ID_DIAGRAM_RUN, OnDiagramRun)
	ON_COMMAND(ID_DIAGRAM_SAVEASACTIVITY, OnDiagramSaveasactivity)
	ON_COMMAND(ID_DIAGRAM_ZOOMIN, OnDiagramZoomin)
	ON_COMMAND(ID_DIAGRAM_ZOOMOUT, OnDiagramZoomout)
	ON_COMMAND(ID_DIAGRAM_INSERTINDIAGRAM_ACTION, OnDiagramInsertindiagramAction)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CDiagramView diagnostics

#ifdef _DEBUG
void CDiagramView::AssertValid() const
{
	CFormView::AssertValid();
}

void CDiagramView::Dump(CDumpContext& dc) const
{
	CFormView::Dump(dc);
}

CGeoDoc* CDiagramView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CGeoDoc)));
	return (CGeoDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CDiagramView message handlers

void CDiagramView::OnInitialUpdate()
{
	CFormView::OnInitialUpdate();

	CGeoDoc* pDoc = GetDocument();
	pDoc->pDiagramView = this;
}

void CDiagramView::OnSize(UINT nType, int cx, int cy)
{
	CFormView::OnSize(nType, cx, cy);

	CRect rc;
	GetClientRect(rc);

	CWnd *pWndAddFlow = GetDlgItem(IDC_ADDFLOWCTRL1);
	if (pWndAddFlow)
```

```
                    pWndAddFlow->MoveWindow(rc);
}

void CDiagramView::OnDiagramFullextent()
{
        m_diagram.SetXZoom(100);
        m_diagram.SetYZoom(100);
}


void CDiagramView::OnDiagramZoomin()
{
        m_diagram.SetXZoom(m_diagram.GetXZoom()*1.25);
        m_diagram.SetYZoom(m_diagram.GetYZoom()*1.25);
}

void CDiagramView::OnDiagramZoomout()
{
        m_diagram.SetXZoom(m_diagram.GetXZoom()/1.25);
        m_diagram.SetYZoom(m_diagram.GetYZoom()/1.25);
}

//*********************************************************
//* Calls to Diagram *************************************
//*********************************************************

void CDiagramView::OnDiagramInsertindiagramInitialstate()
{
        char t[8];

        int index = GetDocument()->m_listCounter;
        CAction *pAction;
        pAction = new CAction;

        GetDocument()->m_actionArr[index] = pAction;
        CNode newNode = m_diagram.GetNodes().Add(1250, 200+index*1000, 500, 500);

        newNode.SetDrawColor(RGB(0, 0, 0));
        newNode.SetShape(afEllipse);
        newNode.SetText("");
        newNode.SetUserData(index);

        //pAction->m_syntax = "Arc: shapearc <inlayer> <outlayer>";
        pAction->m_index = index;
        pAction->m_command = "shapearc ";
        pAction->m_inLayer = "infile";
        itoa(index, t, 10); strcat (t, "_Layer");
        pAction->m_outLayer = t;

        newNode.SetKey(t);

        prevNode = newNode;
}


void CDiagramView::OnDiagramInsertindiagramAction()
{
        char t[8];

        int index = ++GetDocument()->m_listCounter;
        CAction *pAction;
        pAction = new CAction;

        GetDocument()->m_actionArr[index] = pAction;
        CNode newNode = m_diagram.GetNodes().Add(500, 200+index*1000, 2000, 500);

        newNode.SetDrawColor(RGB(0, 0, 0));
        newNode.SetShape(afRoundRect);
        newNode.SetText("Action");
        newNode.SetUserData(index);
```

```
		//pAction->m_syntax = "Arc: command <inlayer> <outlayer> {...}";
		pAction->m_index = index;
		pAction->m_command = "command";
		itoa(index-1, t, 10);strcat (t, "_Layer");
		pAction->m_inLayer = t;
		itoa(index, t, 10);strcat (t, "_Layer");
		pAction->m_outLayer = t;

		newNode.SetKey(t);

		if (prevNode)
				CLink link = prevNode.GetOutLinks().Add(newNode);
		prevNode = newNode;
}

void CDiagramView::OnDiagramInsertindiagramActivity()
{
}

void CDiagramView::OnDiagramInsertindiagramConcurrentfork()
{
}

void CDiagramView::OnDiagramInsertindiagramConcurrentjoin()
{
}

void CDiagramView::OnDiagramInsertindiagramFinalstate()
{
		char t[8];

		int index = ++GetDocument()->m_listCounter;
		CAction *pAction;
		pAction = new CAction;

		GetDocument()->m_actionArr[index] = pAction;
		CNode newNode = m_diagram.GetNodes().Add(1250, 200+index*1000, 500, 500);

		newNode.SetDrawColor(RGB(0, 0, 0));
		newNode.SetShape(afEllipse);
		newNode.SetText("");
		newNode.SetUserData(index);

		//pAction->m_syntax = "Arc: arcshape <inlayer> <outlayer>";
		pAction->m_index = index;
		pAction->m_command = "arcshape ";
		itoa(index-1, t, 10);strcat (t, "_Layer");
		pAction->m_inLayer = t;
		itoa(index, t, 10);strcat (t, "_Layer");
		pAction->m_outLayer = t;

		newNode.SetKey(t);

		if (prevNode)
				CLink link = prevNode.GetOutLinks().Add(newNode);
		prevNode = newNode;
}

void CDiagramView::OnDiagramInsertindiagramSequentialbranch()
{
		// TODO: Add your command handler code here
}

void CDiagramView::OnDiagramRun()
{
		CString s;
		CDiagramRunDlg dlg;

		int i = 0;
```

```
	int index = GetDocument()->m_listCounter;

	dlg.pDoc = GetDocument();
	dlg.index = index;
	if ( !dlg.DoModal() )
		return;

	if (!GetDocument()->rpcInfo.isConnected)  // if not connected, exit
		return;

	for (i=0; i<=index; i++) {
		s = GetDocument()->m_actionArr[i]->m_command + " " +
			GetDocument()->m_actionArr[i]->m_parameters1 + " " +
			GetDocument()->m_actionArr[i]->m_inLayer + " " +
			GetDocument()->m_actionArr[i]->m_parameters2 + " " +
			GetDocument()->m_actionArr[i]->m_outLayer + " " +
			GetDocument()->m_actionArr[i]->m_parameters3;
		rpcSend(s.GetBuffer(0));
	}
}

void CDiagramView::OnDiagramSaveasactivity()
{
	// TODO: Add your command handler code here
}

//*********************************************************
//* Calls to ArcInfo Actions *****************************
//*********************************************************

BEGIN_EVENTSINK_MAP(CDiagramView, CFormView)
  //{{AFX_EVENTSINK_MAP(CDiagramView)
	ON_EVENT(CDiagramView, IDC_ADDFLOWCTRL1, -601 /* DblClick */, OnDblClick, VTS_NONE)
	//}}AFX_EVENTSINK_MAP
END_EVENTSINK_MAP()


void CDiagramView::OnDblClick()
{
	CNode m_Node = m_diagram.GetSelectedNode();
	if (!m_Node)				// exit if there is no selected node
		return;

	CActionDlg dlg;

	int index = m_Node.GetUserData();
	CAction *pAction = GetDocument()->m_actionArr[index];

	dlg.m_actionText = m_Node.GetText();
	dlg.m_syntax = pAction->m_syntax;
	dlg.m_command = pAction->m_command;
	dlg.m_inLayer = pAction->m_inLayer;
	dlg.m_outLayer = pAction->m_outLayer;
	dlg.m_parameters1 = pAction->m_parameters1;
	dlg.m_parameters2 = pAction->m_parameters2;
	dlg.m_parameters3 = pAction->m_parameters3;

	if (dlg.DoModal() == IDOK)
	{
		m_Node.SetText(dlg.m_actionText.GetBuffer(0));
		pAction->m_command = dlg.m_command;
		pAction->m_inLayer = dlg.m_inLayer;
		pAction->m_outLayer = dlg.m_outLayer;
		pAction->m_parameters1 = dlg.m_parameters1;
		pAction->m_parameters2 = dlg.m_parameters2;
		pAction->m_parameters3 = dlg.m_parameters3;
	}
}
```

Bibliography

Adler, R. M. *Distributed Coordination Models for Client/Sever Computing*. Computer vol. 28 no. 4 April 1995. pp. 14-22.

Alameh, Nadine. *Internet-Based Collaborative Geographic Information System*. Master Thesis submitted to the Department Urban Studies and Planning and Civil and Environmental Engineering. MIT 1997.

Arc/Info Data Management: Concepts, data models, database design and storage. Environmental Systems Research Institute, 1994.

Berson, Alex. *Client/Server Architecture*. New York: McGraw-Hill, 1996.

Booch, Grady, James Rumaough and Ivar Jacobson. *The Unified Modeling Language User Guide*. Reading: Addison-Wesley, 1998.

Booch, Grady, James Rumaough and Ivar Jacobson. *The Unified Modeling Language Reference Manual*. Reading: Addison-Wesley, 1999.

Booch, Grady. *Object-Oriented Analysis and Design With Applications*. Reading: Addison-Wesley, 1994.

Buehler, Kurt and Lance McKee, editors. *The OpenGIS Guide: Introduction to Interoperable Geoprocessing*. Wayland: Open GIS Consortium, 1996.

Building Applications with MapObjects. Environmental Systems Research Institute, 1996.

Cashin, Jerry. *Distributed Systems: Future Trends and Technologies*. Charleston: Computer Technology Research Group, 1997.

Dickman, A. *Two-Tier Versus Three-Tier Apps*. Information week no. 553 November 13, 1995. pp. 74-80.

Edelstein, Herb. *Unraveling Client/Server Architecture*. DBMS vol. 7 no. 5 May 1994 pp. 34-7.

Environmental Systems Research Institute Homepage, 1999. URL: http://www.esri.com/.

Esri Spatial Database Engine to Support Oracle8 Spatial Cartridge. ESRI Homepage, 1997. URL: http://www.uk.oracle.com/info/news/esri.html/.

Evans, John D. *Infrastructures for sharing geographic information among environmental agencies*. Ph. D. Thesis presented to the Department of Urban Studies and Planning. MIT 1997.

Ferreira, Joseph, Ayman Ismail and Chen-Hsiang Yeang (forthcoming 1999*). A Distributed GIS for Monitoring and Modeling Urban Air Quality*. Proceedings of the International Conference on Computer in Urban Planning and Urban Management, Venice, 1999.

Fotheringham, Stewart and Peter Rogerson ed. *Spatial Analysis and GIS*. London: Taylor & Francis, 1994.

Friedel, David H. Jr. and Anthony Potts. *Java Programming Language Handbook.* Scottsdale: Coriolis Group Books, 1996.

Gallaugher, J. & Ramanathan, S. *Choosing a Client/Server Architecture. A Comparison of Two-Tier and Three-Tier Systems.* Information Systems Management Magazine vol. 13, no. 2 Spring 1996: pp. 7-13.

Getting Started with Internet Map Server. Environmental Systems Research Institute, 1996.

Jacobson, Ivar. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Reading: Addison-Wesley, 1994.

Krueckeberg, Donald A. and Arthur L. Silvers. *Urban Planning Analysis: Methods and Models.* New York: John Wiley & Sons, 1974.

Kruglinski, David J., George Shepherd and Scot Wingo. *Programming Microsoft Visual C++*. Redmond: Microsoft Press, 1998.

MapObjects Programmer's Reference. Environmental Systems Research Institute, 1996.

Microsoft Homepage, 1999. URL: http://www.microsoft.com/.

Newell, D.; Jones, O.; & Machura, M. *Interoperable Object Models for Large Scale Distributed Systems*. 30-31. Proceedings of the International Seminar on Client/Server Computing. La Hulpe, Belgium, October 30-31, 1995. London, England: IEE, 1995.

Object Management Group Homepage, 1999. URL: http://http://www.omg.org/.

Oracle Corporation Homepage, 1999. URL: http://www.oracle.com/.

Penn State/NCAR Mesoscale Modeling System (MM5) Documentation. URL: http://www.mmm.ucar.edu/mm5/doc.html/.

Rational Corporation Homepage, 1999. URL: http://www.rational.com/.

Rumbaugh, James, Michael Blaha, William Premerlani, Frederick Eddy, William Lorenson. *Object-Oriented Modeling and Design*. Prentice-Hall, 1991.

Schussel, George. *Client/Server Past, Present, and Future*. URL: http://http://www.dciexpo.com/geos/.

SDE: Spatial Database Engine. Environmental Systems Research Institute, 1996.

Shelton, Robert E. *The Distributed Enterprise:Shared, Reusable Business Models the Next Step in Distributed Object Computing*. Distributed Computing Monitor vol. 8 no. 10 October 1993. p. 1.

Stroustrup, Bjarne. *The C++ Programming Language*. Reading: Addison-Wesley, 1997.

Sun Microsystems: JavaSoft Homepage, 1999. URL: http://www.javasoft.com/.

Understanding GIS: The Arc/Info Method. Environmental Systems Research Institute, 1992.

Visual Studio: Developing for the Enterprise. Redmond: Microsoft Press, 1998.

Worboys, Michael F. *GIS: A Computing Perspective*. London: Taylor & Francis, 1995.