2011

A spatial mediator model for integrating heterogeneous spatial data

Hsine-jen Tsai
Iowa State University

A spatial mediator model for integrating heterogeneous spatial data

by

Hsine-Jen Tsai

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:
Leslie Miller, Major Professor
Shashi Gadia
Sree Nilakanta
Wallapak Tavanapong
Johnny Wong

Iowa State University

Ames, Iowa

2011

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

The complexity and richness of geospatial data create specific problems in heterogeneous data integration. To deal with this type of data integration, we propose a spatial mediator embedded in a large distributed mobile environment (GeoGrid). The spatial mediator takes a user request from a field application and uses the request to select the appropriate data sources, constructs subqueries for the selected data sources, defines the process of combining the results from the subqueries, and develop an integration script that controls the integration process in order to respond to the request. The spatial mediator uses ontologies to support search for both geographic location based on symbolic terms as well as providing a term-based index to spatial data sources based on the relational model. In our approach, application designers only need to be aware of a minimum amount about the queries needed to supply users with the required data. The key part of this research has been the development of the spatial mediator that can dynamically respond to requests within the GeoGrid environment for geographic maps and related relational spatial data.

CHAPTER 1. INTRODUCTION

Every local system designer tends to develop the database that can meet his/her organization's specific needs. It results in huge diversity in a multiple heterogeneous data sources environment. In this environment, a variety of sources and applications use different data models, representations and interfaces. System designers need to develop integrated systems that allow users to access and manage information from multiple heterogeneous data sources. One reason for such need has been that environments for data access have changed from centralized data systems into multiple, distributed data sources. Another more recent cause for the attention of integration technologies is the emergence of e-commerce and its needs for accessing data repositories, application and legacy source that located across the organization intranet or on the Internet (Hammer & Pluempitiwiriyawej, 2001). While there are more varied and complicated data available to users the integration of heterogeneous data becomes more challenging (Ram, Park & Lee, 1999). How to provide this capability has become an important and active research area in the information system community.

There are two major tasks that an integration system needs to accomplish in order to solve the problem. First, a set of suitable data sources containing data that correspond to the response needed to answer a user's query should be located. Second,

after data sources have been found, the system not only needs to bring together all data required it also needs to resolve the heterogeneity among these data.

The primary problem that an integration system faces is to resolve heterogeneity among data sources which include conflicts of naming, scaling, formatting, computational, and granularity among multiple data sources. Identification inconsistencies and constraint mismatches also are included in this problem. The heterogeneity can be classified into two types, namely semantic and syntactic heterogeneity (Kim & Seo, 1991), (Wache, et al. 2001). The latter is sometimes referred as structural heterogeneity (Wache, et al. 2001). Structural heterogeneity means that different information sources use different structure to store their data while the semantic heterogeneity refers to the differences in the meaning of the data that is interchanged between data sources (Wache et al. 2001).

A number of researchers have worked in the area of integrating heterogeneous data (Tuchinda et al. 2004, Thakkar et al. 2007, Park & Ram 2004, Ghulam 2010, Hribernik et al. 2010, Weiderhold 1992). Several data integration architectures have been designed by projects like TSIMMIS (Chawathe et al. 1994), COIN (Moulton et al. 2002), MOMIS (Bergamaschi et al. 1999). One approach to building an integration system is the data warehouse approach (Fan & Poulovassilis 2003, Golfarelli & Rizzi 1998, Wu et al. 2001) which pre-fetches, merges and resolves existing discrepancies

between sources and then stores integrated information in the central repository to answer users' queries. Another approach is referred to as mediation. It provides users with an integrated view of the underlying source. Data remains stored at their local sites. Users can query the mediator, which locates relevant data sources and integrates each individual result into a format that can satisfy users' requests (Wiederold, 1992, Athanasiadis & Janssen 2008, Michalowski, et al. 2004, Thakkar et al. 2003).

The problem is even more interesting when one considers the integration of spatial data. Not only are there the typical problems of heterogeneity, but in addition the data types available in spatial data repositories represent very rich data types. Even a cursory glance at spatial data sources indicates the need of combining everything from maps to general data that includes some type of location. For example, crime data typically includes the location where a crime occurred.

Another difference between integration of traditional relational data and geographic data is that more human participation is needed in the integration of geographic data. In some cases, experts of the geographic domain are needed to participate in the integration process. For example, a specialist called a geomatician is used to refer to professionals who gather, process and deliver geographic data to users by using a CASE tool of a integration system (Coimbra 2009).

Many approaches to provide solutions for integration of spatial data have been put forward (Park & Ram 2004, Goodchild et al. 2007, Visser et al.2002, Vidal et al. 2009). In addition there is research that utilizes ontologies to solve the semantic heterogeneity among multiple data sources (Vidal et al. 2009, Janowicz et al. 2010) and others employ scheme query mapping mechanism to resolve the heterogeneity problem (Park & Ram 2004, Ghulam 2010), there are few approaches that provide a comprehensive approach for generating geographic maps and related spatial data in a mobile environment built on exploiting data quality. Our work aims to develop a spatial mediator system that can provide this need.

The work that our research group has done with the Census Bureau has led us to believe that the most critical need is the development of an environment that makes integrated spatial data available in the field. Most of the agency applications that involve spatial data are used in the field.

To this end, our research group has proposed and implemented the GeoGrid infrastructure for providing spatial data to users in a mobile environment. The focus of this thesis is on the spatial mediator that takes a user request from a field application and develops a script that defines the process of how the available data sources are used to respond to the request. It should be noted that while this remains

a significant problem, the mobile environment proposed does provide some restrictions that make it more manageable than the general problem.

We propose a spatial mediator model that utilizes ontologies to help user application designers to use domain terms to access data from multiple heterogeneous spatial data sources. In our approach, application designers only need to be aware of a minimum amount about the queries needed to supply users with the required data.

The application designers along with the data and tool suppliers provide the basic information about their applications, data sources, and/or tools, respectively. The spatial mediator uses this information along with the user request in order to place the major burden of dealing with the system heterogeneity on the spatial mediator.

The main contribution of our work has been the development of comprehensive approach to generating spatial data results for a very real problem – an infrastructure for supporting dynamic access to heterogeneous spatial data in a mobile environment. The key part of this research has been the development of the spatial mediator that can dynamically respond to requests for geographic maps and related relational spatial data.

Secondary contributions have been made while building the tools necessary to implement the spatial mediator. Our weighted ontology (Tsai et al. 2001, Tsai et al.

2003) is an example of our contribution is this area. Our approach of combining the weighted ontology with the semantic data model is another example (Tsai et al. 2003).

The remainder of this thesis is organized as follows. In Chapter 2, we review the background material for the concepts required by the spatial mediator. Chapter 3 looks at our ontology model and its application to our spatial mediator. The spatial mediator model is defined in Chapter 4 and evaluated in Chapter 5. Chapter 6 provides a conclusion and some thoughts for future work on these topics. Supplementary material is given in the appendices.

CHAPTER 2. LITERATURE REVIEW

An overview of literature related to spatial mediation of geographic data is presented in this chapter. We collect and categorize relevant research into three areas: ontology supported data integration, mediator-based data integration and spatial data quality. We discuss current literature in these areas and also look into some related issues.

In Section 2.1, we review related work in the field of ontology supported data integration. We also look at some ontology based geographic data integration systems. Several mediator based data integration systems are reviewed in Section 2.2. Literature related to spatial data quality is discussed Section 2.3.

2.1 Ontology-based data integration

An ontology is built as an explicit representation of semantics of each data source. An excellent discussion of ontologies, regarding the actual range of knowledge that an ontology can successfully represent, can be found in (Brewster & O'Hara 2004). A detailed discussion of evaluating ontology tools and ontology contents can be found in (Guraino et al. 2009).

In (Weng et al. 2006), the authors develop an automated technology of ontology construction by using the theory of formal concept analysis to serve as the groundwork in assembling the different levels of ontological concepts. Cui et al.

(2009) developed top-down and bottom-up construction methods. The bottom-up approach makes use of formal concept analysis methods and the Wikipedia is used as the corpus for the acquisition.

Methods have been developed for building geospatial ontologies. Baglioni et al. (2007) define new relations in geospatial terms that express spatial properties. A geospatial ontology can be extracted from these relations. The ontology could be used as the basis for an advanced user query system.

2.1.1 Ontology support for heterogeneous data integration

In dealing with multi-database systems, ontologies can be used effectively to organize keywords as well as database concepts by capturing the semantic relationships among keywords or among tables and fields in a relational database. By using these relationships, a network of concepts can be created to provide users with an abstract view of an information space for their domain of interest. We discuss several ontology-based integration systems in the following paragraphs.

Seng & Kong (2009) introduced an ontology-aided integration approach that allows a query over multiple intelligent information sources. Their ontology is used to enhance both structural and semantic interoperability.

Project OBSERVER (Mena et al.1998) considers the metadata description and ontologies for each different information source and provides knowledge on the

vocabulary used in the source. The information source is viewed by using its relevant semantic concept which can be chosen from pre-existing domain specific ontologies.

In the Web domain, a global ontology can be used as a modeling tool and serve as a base of integration.

SOBA, an ontology-based information extraction and integration system, focuses on processing structured data, text and image caption (Buitelaar et al. 2008). It is capable of acquiring factual knowledge for a certain domain based on a given ontology.

HELIOS, an ontology-based knowledge sharing system in the P2P area, employees peer ontologies to allow information search and knowledge sharing (Castano et al. 2003). The peer ontology is the ontology inside each peer that describes knowledge about itself.

There are several differences between our ontology search model and the projects mentioned above. First of all, we define multiple relations between terms to present different relationship between terms. The relations in these systems are simpler. Secondly, the ontology model serves as a search module in our approach. We define and use weights between terms inside the ontology to aid the search. The existing ontology systems don’t consider the weights between terms inside the

ontology, their focus are on the mapping between schemas of data sources. Third, the semantic network model connected to our ontology model is used to resolve the semantic heterogeneity between data sources. These systems use the ontology itself to handle the semantic differences between data sources.

2.1.2 Ontology supported geospatial information integration

Geospatial data is very diverse and dynamic. The geospatial information may be unstructured or semi-structured, and usually there is no regular schema to describe them. As the amount of geospatial data grows, a problem of interoperability between multiple geospatial data sources has gained a growing attention. Many approaches to provide solutions have been developed. Using ontologies to support the interoperability is one of them.

The use of formal ontologies for geographical information integration is introduced in (Cruz & Calnan 2002, Stuckenschmidt et al. 2002, Wache et al. 2001, Visser et al. 2002). They propose an intelligent architecture for semantic-based information retrieval. This architecture uses underlying ontologies and an inference engine that has the ability to derive new knowledge.

The paper proposed by O'Brien and Gahegan (O'Brien 2005) presents a framework for representing, manipulating and reasoning with geographic semantics. They use ontologies to describe methods, data and human experts inside their

environment. Among them an entity's ontology describes users, inputs, outputs, and semantic changes. Our work uses an entity ontology to describe spatial concepts of the real world.

In Vidal et al. (2009), they propose an approach that rewrites a query based on a domain ontology into sub-queries submitted over multiple data sources and combines data resulting from those sub-queries. In particular, their approach takes advantage of DL (Description Logics) reasoning to remove sub-queries that are not consistent.

The difference between their work and our approach is that they focus on using ontologies with formal semantics to support the translation process between data sources and our approach is to use ontology as a search model to locate data sources for the response to a user's query. Multiple ontologies are used to enhance the semantic integration process (Peachavanish & Karimi 2007). In our ontology model, we introduce the function ontology to enrich our terms in the ontology model and thus to enhance our search.

2.2 Mediator to support the interoperability among multiple data sources

The mediator concept was introduced by (Wiederhold 1992). A mediator is designed to provide a uniform interface to a number of heterogeneous data sources. Given a user query against the global schema, the mediator decomposes it into

multiple local sub-queries and sends them to the appropriate data sources, merges the partial results and reports the final answer to the user. We discuss several mediator-based integration systems in the following paragraphs.

In Athanasiadis & Janssen (2008), a mediator based knowledge manager component is presented. The component exploits ontologies and semantic modeling to support the data exchange between heterogeneous data sources.

The Mediator Prometheus (Michalowski et al. 2004, Thakkar et al. 2003) is a mediator system that uses planning techniques to expand a given query into a set of operations. These operations specify how to access the appropriate data sources, including web services, web sources and databases. It utilizes a “local_as_view” approach to map between the relations in the mediated schema (domain relations) and the source relations. A technique called tuple-level filtering was introduced by Prometheus. The filtering technique can reduce the number of web service requests thus optimizes the execution of the composed web services.

Artemis, a query formulation and planning system, provides the ability of integrating scientific data on the grid (Tuchinda et al. 2004). It enables users to easily query metadata catalogs on the grid. It has an ontology-based query formulation system that exploits semantic web tools to model metadata attributes. It employees a

query mediator based on planning techniques that can dynamically updates its domain model.

Several prototype mediator architectures have been designed by projects like TSIMMIS (Chawathe et al. 1994), COIN (Moulton et al. 2002), MOMIS (Bergamaschi et al. 1999).

TSIMMIS focuses on the automatic generation of wrappers and mediators which conducts mapping the information in an Object-Exchange Model (OEM) to the underlying structured or unstructured data. OEM is used to represent a piece of data. Fields inside the OEM of each object describe semantic of the data. The process of integration inside TSIMMIS requires human participation. In some cases, integration may be automated by a mediator under the guidance of end users. Our approach is different from it is that we use semantic network model and ontology to specify the semantic of the data and the automatic generation of wrappers and mediators is not our focus.

A context interchange (COIN) mediator is an automated reasoning engine which helps users resolve semantic conflicts between their own context and context of data sources (Moulton et al. 2002). They define “context” as the implicit understanding of the relationship between data elements and structures and the real world that data represents. Each data source and receivers decide how to construct

abstract conceptualization and how that is represented in data and programs. They argue that their context interchange approach is a well suited solution in a large-scale interoperable database system of a dynamic environment.

The MOMIS (Mediator environment for Multiple Information Sources) can be considered as a hybrid method of mediator and query language approach ((Bergamaschi et al. 1999).). It aims to integrate and query both a structured data source (i.e. relational database) and a semi-structured data source (i.e. object-oriented data source). While all of these are interesting approaches to providing mediation that supports unstructured or semistructured data, they are not usable for spatial data due to the rich complex structure of geographic data.

2.2.1 Mediation systems for geographic data

With the growing use of geographic information systems much work has been conducted in regards to the research topic of the geographic data integration. The nature of geographic data creates more challenges for supporting interoperability between multiple geographic data sources. Geographic data has diverse and dynamic characteristics and may be unstructured or semi-structured, and usually there is no regular schema to describe them. Reviews of geographic data interoperability and integration efforts are provided in (Park & Ram 2004, Goodchild et al. 2007, Visser et al. 2002).

Many mediator based integration systems have been proposed in a variety of domains. (Smart et al. 2010) propose a mediation framework to retrieve and integrate distributed gazetteer resources. Researches in Michalowski et al. (2004), Park et al. (2004) and Zaslavsky (2004) propose mediation systems to support interoperability within the geographic data domain.

In Park and Ram (2004), they identified semantic conflict among data sources and stored associated knowledge in the ontology called Semantic Conflict Resolution Onto*l*ogy (SCROL). A semantic mediation service layer serves as one of the core components of the SCROL system. Several semantic mediators are employed inside the mediation service layer. Each has its own specific responsibility. They showed the model works well in the domain of geographic data. A semantic data model called Unifying Semantic Model Star (USM*) is proposed for modeling data in GIS databases (Ram et al. 1999). The USM* is used in their SCROL system to manage federated and local schema.

MIX (Mediation of Information using XML) is a mediation-based approach for integrating information in the GIS domain (Gupta et al. 1999, 2000, Zaslavsky et al. 2003). Each data source exports a model of information it contains in the form of an XML definition. This XML model is used as a structural description of the data exchanged by the components inside the mediator architecture. Each source is

queried with an XML-based query language, XMAS (Ludäscher et al. 1999). XMAS allows object fusion and pattern matching on the input XML data.

With the growing numbers of GIS data and resources over the Internet, there is an increasing demand for geospatial information services to support interoperation of massive repositories of heterogeneous geospatial data. VirGIS is a mediation platform that utilizes an ontology and provides an integrated view of geographic data (Essid et al. 2006). Its data is constructed as an ontology by using common Semantic Web techniques. The mediator in VirGIS provides a global virtual view that allows local data sources to be accessed as one integrated source.

A WFS-based mediation system that addresses the integration of GIS data and tools is proposed in (Boucelma & Colonna 2004). It focuses on the expressive power of query language and provides an approach of the integration of query capability available at the source.

One major difference between our approach and these works is that they don't handle map type geographic data and they don't consider the quality of a geographic data. More important it is common in these systems that predefined queries are used and the solutions are known so the task is to execute the solution. The needs for a truly dynamic solution that can generate solutions on the fly are critical in the GeoGrid environment.

2.3 Quality driven geospatial data integration

Data quality and metadata are crucial for the development of Geographic Information Systems. Lassoued et al. (2007) proposes a quality-driven mediation approach and system that allow a community of users to share a set of autonomous, heterogeneous and distributed geospatial data sources with different quality information. A common vision of the data that is defined by a global schema and a metadata schema is shared by users. Devillers et al. (2005) develop a design of a tool that can manage heterogeneous data quality information and provide functions to support expert users in the assessment of the fitness for use of a given data source.

In Hariharan et al. (2005), they develop approximate algorithms for answering queries based on the local analysis of the query region. The quality of answers improves progressively as the local analysis goes deeper. Data sources are ranked by a weighted score function that is based on two criteria: spatial coverage and information content which base captures how much of the query-specified keywords are present in a data source.

The QGM (Quality-driven Geospatial Mediator) supports efficient and accurate integration of geospatial data from a large number of sources (Thakkar et al. 2007). It features an ability to automatically estimate the quality of data provided by a source by using the information from another source of known quality.

There are several differences between our works and theirs. First of all, we use quality of geographic data to rank and select data sources before we dynamically generate integration script. Secondly, we identify several geographic attributes of a geographic data to represent the quality of a geographic data. Third, we allow the user/applications designer to specify their perspectives of the quality of geographic data.

CHAPTER 3. ONTOLOGY MODEL

Integration of data continues to be a problem. The number of databases available to users continues to grow and it is difficult for users to get all of the data they need from a single data source. It is clear that such a user will need data integration software to make use of multiple data sources.

In dealing with multi-database systems (Hribernik et al. 2010, Ghulam 2010), ontologies can be used very effectively to organize keywords as well as database concepts by capturing the semantic relationships among keywords or among tables and fields in a relational database (Seng & Kong 2009, Vidal et al. 2009, Buitelaar et al. 2008, Baglioni et al. 2007). By using these relationships, a network structure can be created to provide users with an abstract view of an information space for their domain of interest. Ontologies are well suited for knowledge sharing in a distributed environment where, if necessary, various ontologies can be integrated to form a global ontology.

Database owners find ontologies useful because ontologies can be used to form a basis for integrating individual databases by using identification of logical connections or constraints between the information pieces. Ontologies can provide a simple conversational interface to existing databases and support extraction of information from them. Because of the distinctions made within an ontological

structure, they are used to support database cleaning, semantic database integration, consistency-checking, and data mining (Brewster & O'Hara 2004).

In this chapter we look at our ontology model (Tsai et al. 2001, Tsai et al. 2003) to set the role of ontologies in our approach to data integration, i.e. the spatial mediator. There are two major components in the spatial mediator that utilize our ontology model, namely the Geographic Symbolic Ontology (***GSO***) and the Semantic Network Ontology (***SNO)***. These two components are both based on the ontology models presented in this chapter.

We present the definition and examples of our ontology model in Section 3.1. The search algorithm of the ontology is presented in Section 3.2. The ontology model has been expanded to incorporate a function ontology to allow users to include tools into the query process. The definition of our function ontology model is introduced in Section 3.3 and we present the semantic network of ***SNO*** in Section 3.4.

3.1 Ontology model

From an artificial intelligence viewpoint, an ontology is a model of some portion of the world and is described by defining a set of representational terms (Neches et al. 1991). In an ontology, definitions associate the names of entities in a universe of discourse (e.g., classes, relations, functions, or other objects) with

human-readable text describing what the names mean, and formal axioms that constrain the interpretation and well-formed use of these terms (Gruber 1993).

The main motivation behind ontologies is that they allow for sharing and reuse of knowledge bodies in computational form. In the Knowledge Sharing Effort (KSE) project (Neches et al. 1991), ontologies are put forward as methods to share knowledge bases between various knowledge-based systems. The basic idea was to develop a library of reusable ontologies in an uniform formalism that each system developer was supposed to adopt. Originally, the term ontology comes from where it is employed to describe the existence of beings in the world. Artificial Intelligence (AI) deals with reasoning about models of the world. Therefore, it is not strange that AI researchers adopted the term ontology to describe what can be (computationally) represented of the world in a program.

Many definitions of ontologies have been put forward (Guarino et al. 2009, Sowa 2001). One that seems to best characterize our view of the essence of an ontology (Gruber 1993, p199): “An ontology is a formal, explicit specification of a shared conceptualization”. *Conceptualization* refers to an abstract model of some phenomena in the world by having identified the relevant concepts of those phenomena. *Explicit* means that the type of concepts used, and the constraints on their use are explicitly defined. For example, in medical domains, the concepts are diseases and symptoms, the

relations between them are causal, and a constraint is that a disease cannot cause itself. *Formal* refers to the fact that the ontology should be machine readable, which excludes natural language. *Shared* reflects the notion that an ontology captures consensual knowledge, that is, it is not private to some individual, but accepted by a group.

Since the various definitions of ontologies have varied, the next subsection looks at the formal definition of ontologies that we introduced in (Tsai et al. 2001, Tsai et al. 2003).

3.1.1 Ontology model definition

Definition 3.1:

An *ontology* is defined as $\boldsymbol{O} = <T, R, S>$ where

$T = \{t_i | \ i = 1..n\}$ is a set of terms, where each term refers to a set of real-world objects,

$R \subseteq T \times T = \{r_i \mid i = 1..m\}$ is a set of relations between terms, defined as $R = \{(t_1,t_2) | t_1, t_2 \in T\}$, and

S is a set of operations needed to create, maintain and search the ontology structure defined by T,R.

To make our ontology model sufficiently rich to handle the needs of the spatial mediator, several types of relation types have been defined between terms (*is-a, has-instance, part-of, contains, is-closely-related, synonym, antonym*). Our

implementation is based on viewing the ontology structure as a directed graph (*T*, *R*,) where each node t $\in$ T represents a term and is labeled with the term. The edges of the directed graph represent the relation between terms and are labeled with the relation type. Some terms used in examples below are adapted from Cote (2006).

Definition 3.2:

The relation type t_1 *is-a* t_2 is a relation between t_1 and t_2 such that t_1 is a subtype of t_2.

A simple example of *is-a* is "car *is-a* vehicle". The graph structure of the relation type is shown in Figure 3.1a and the example is given in Figure 3.1b.

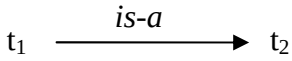


Figure 3.1a: The *is-a* relation between t_1 and t_2.

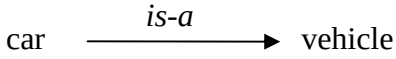


Figure 3.1b: The *is-a* relation between car and vehicle.

Definition 3.3:

The relation type t_1 *has-instance* t_2 is a relation between t_1 and t_2 such that t_1 is a super type of t_2.

A simple example is "vehicle *has-instance* car". The graph structure of the relation type is given in Figure 3.2a and the example is given in Figure 3.2b.

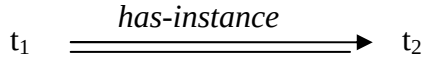


Figure 3.2a: The *has-instance* relation between t_1 and t_2.

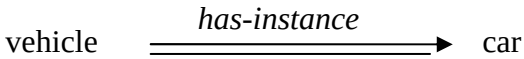


Figure 3.2b: The *has-instance* relation between vehicle and car.

Note that the two relation types point in different directions in the directed graph. Figure 3.3 shows this relationship for car and vehicle.

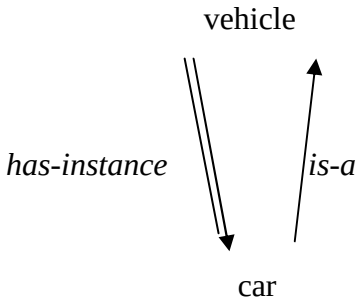


Figure 3.3: The *is-a* and *has-instance* relations between car and vehicle.

Definition 3.4:

The relation type t_1 *is-part-of* t_2 is a relation between t_1 and t_2 such that presence of t_1 implies the presence of t_2, but the occurrence of t_2 doesn't imply the presence of t_1.

A simple example of *is-part-of* is "sea floor *is-part-of* sea". The graph structure of the relation type is shown in Figure 3.4a and the example is given in Figure 3.4b.

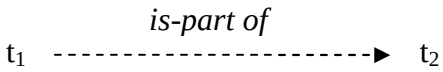


Figure 3.4a: The *is-part-of* relation between t_1 and t_2.

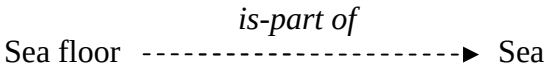


Figure 3.4b: The *is-part-of* relation between sea floor and sea.

Definition 3.5:

The relation type t_1 *contains* t_2 is a relation between t_1 and t_2 such that presence of t_2 implies the presence of t_1, but the occurrence of t_1 doesn't imply the presence of t_2.

A simple example of *contains* is "sea *contains* sea floor". The graph structure of the relation type is shown in Figure 3.5a and the example is given in Figure 3.5b.

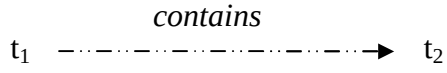


Figure 3.5a: The *contains* relation between t_1 and t_2.

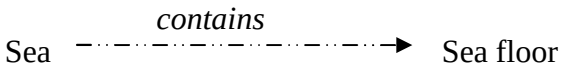


Figure 3.5b: The *contains* relation between sea floor and sea.

Definition 3.6:

The relation type t_1 *synonym* t_2 is a relation between t_1 and t_2 such that t_1 and t_2 are not identical but have same meanings.

This relation is symmetric. A simple example is “brook *synonym* creek”. The graph structure of the relation type is given in Figure 3.6a and the example is given in Figure 3.6b.

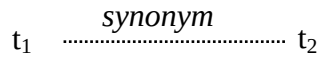


Figure 3.6a: The *synonym* relation between t_1 and t_2.

Figure 3.6b: The *synonym* relation between brook and creek.

Definition 3.7:

The relation type t_1 *antonym* t_2 is a relation between t_1 and t_2 such that t_1 and t_2 have opposite meanings.

This relation is symmetric. A simple example is “female *antonym* male”. The graph structure of the relation type is given in Figure 3.7a and the example is given in Figure 3.7b.

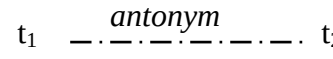


Figure 3.7a: The *antonym* relation between t_1 and t_2.

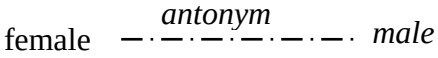


Figure 3.7b: The *antonym* relation between *female* and *male.*

Definition 3.8:

The relation type t_1 *is-closely-related* t_2 is a relation between t_1 and t_2 such that t_1 and t_2 are not considered as synonyms but are generally used together.

A simple example of *is-closely-related* is "automobile *is-closely-related* car". The graph structure of the relation type is shown in Figure 3.8a and the example is given in Figure 3.8b.

t_1 —*is-closely-related*— t_2

Figure 3.8a: The *is-closely-related* relation between t_1 and t_2.

automobile —*is-closely-related*— car

Figure 3.8b: The *is-closely-related* relation between automobile and car.

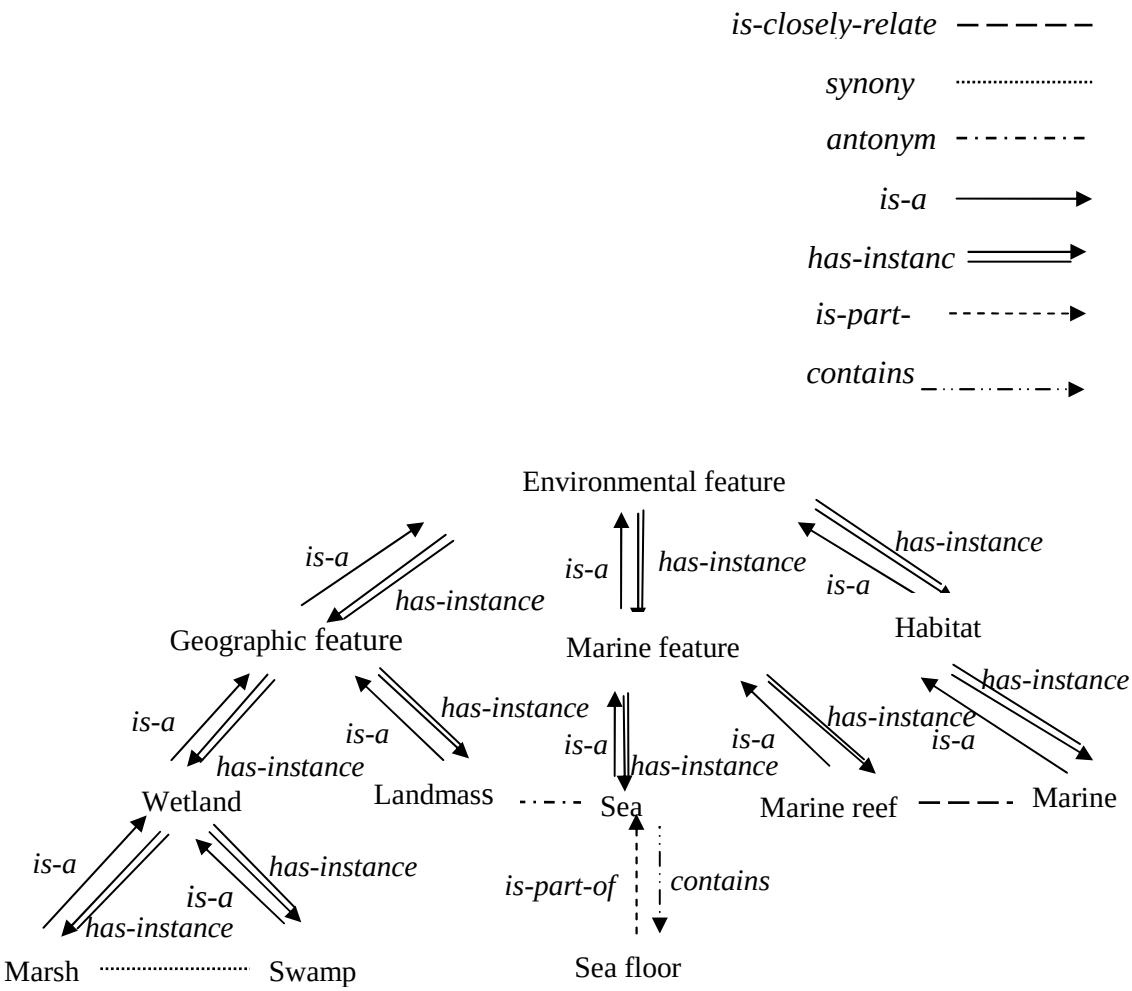


Figure 3.9: A fragment of a sample ontology represented by a graph.

In Figure 3.9, a fragment of an ontology is represented by a directed acyclic graph (dag) with more general terms higher in the dag and more specific terms lower in the dag. The *is-a* and *part-of* are directed. Relations *synonyms* and *antonym* and *is-closely-related* are not directed edges. “Marine feature” *is-a* “Environmental feature” and “Sea” *is-a* “Marine feature”. “Sea floor” *is-part-of* “Sea” indicates wherever there is a “Sea floor” there must be a “Sea” and a “Sea” *contains* “Sea floor”. Both “Marsh” and “Swamp” *is-a* “Wetland” and these two terms are *synonyms*. “Sea” and “Landmass” are *antonyms*. “Marine reef” and “Marine habitat” are

connected by a *is-closely-related* relation since “Marine reef” is an example of a good “Marine habitat” and they are most likely referred together in the study of marine habitation.

The set of operations given in the definition of the ontology need to be able to create, maintain and search the directed graph used to store the ontology structure. The most interesting operation is the search operation. Before looking at the search algorithm we examine an extension of our basic ontology structure in the next subsection.

3.1.2 The weighted ontology

To enhance the search operation, we add the notion of edge weights to create a weighted ontology.

Definition 3.9:

A weighted ontology is defined as $\theta = <T,R,W,S>$, where

$T = \{t_i | \ i = 1..n\}$ is a set of terms such that each term refers to a set of real world objects,

$R \subseteq T \times T = \{r_i | \ i = 1..m\}$ is a set of relations between terms, defined as $\{(t_1, t_2) | \ t_1, t_2 \in T\}$,

W is a set of weights $\{w_i | i = 1..m\}$, where each weight w_i is assigned to a relation r_i to indicate the value of following the relation in a search, and

S is a set of operations needed to create, maintain, and search the ontology structure defined by *T, R, W*.

The weighted ontology supports the same relation types as the ontology defined in the previous section. In the remainder of this thesis we will use the term ontology to mean weighted ontology.

To more clearly describe what we mean by a weighted ontology we will examine the concept in the directed graph representation. We will use the term "edge" instead of "directed edge" in the remainder of the chapter. We will use "term node" and "term" interchangeably and also "relation" and "edge" will be used as interchangeable words in the remainder of the chapter. We use the directed graph notation $\varphi = (\eta,\xi)$ to represent the ontology structure where η is the set of terms used to represent the domain and ξ is the set of edges connecting the nodes representing the terms. Let ω be the set of weights such that ω_j □ ω is the weight for the $E_i \varepsilon \xi$. We use the weights to prune the search of the ontology. Let $I(\xi)$ be the set of the *is-a*, *has-instance, part-of, contains* and *is-closely-related* relations in the ontology. For $E \varepsilon I(\xi)$, the weights are used to estimate the relative closeness of t_2 to t_1 when t_1 relation

t_2 defines an edge in the graph. Going through a term like “Geographic feature” would in general not produce good search results. To block the search, the weights on an edge (like an *is-a* edge) are set to larger values if the relationship is more abstract. The weights on an edge range from zero to infinity. The only requirement for the weight values is that they provide the type of search requirements that the user wants. In our

Current implementation they are static, but we have considered allowing users to have their own weighting scheme based on the type of search that they want to conduct. An example of a weighting scheme that has worked well in our implementation of the ontologies with static weights used in the spatial mediator is shown in Table 3.1.

Table 3.1 A sample set of weights by edge (relation)

Edge Type	Weight
is-a	200
has-instance	50
is-part-of	100
contains	100
is-closely-related	0 to 10
synonym	0 to 10
antonym	0 to 10

The idea behind the weights shown in Table 3.1 is that we want to control the search. The *is-a* weight of 200 means that we are trying to minimize (block) the search task of going from a specific concept like car to a more abstract concept like vehicle. The reason being that a move in that direction weakens the search. On the other hand, we have found it to be more useful to go from abstract to specific, so we use a smaller weight (50) to make that a more probable search direction. The two edge types that are the most interesting on a positive search are *is-closely-related* and *synonym*. Assuming that edges are only used for concepts that are closely related and those that have the same meaning, these weights should be small. Setting these weights to a value in the range 0 -10 (depending on how close the meaning is) means that these edges will be exploited early in the search when they are available. The *antonym* edge plays the search role in the not concept search.

The *is-part-of* and *contains* edges have not proved as useful and we have used the weights (100) to reduce the likelihood of the search using those edges in the mediator ontology searches.

A future consideration for using the weighted ontology in the spatial mediator will be to add questions to the registration process about the nature of the way a user application will use concepts. That way, the weights will be able to reflect the user needs more closely than our current static model can.

The Figure 3.10 shows a fragment of a weighted ontology which has two *is-a* edges with different weights. Compare “Underground river” to the term “Geographic feature”, the term “Wetland” is more general, so the weight associated with “Wetland” is set 200 which is greater than 50, the weight associated with “Underground river”.

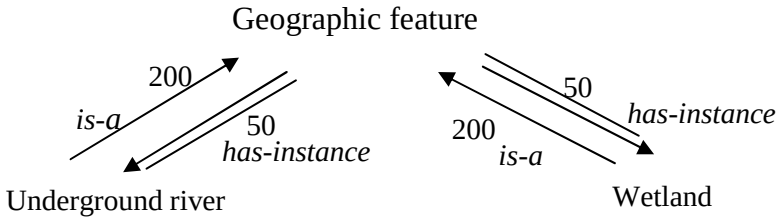


Figure 3.10: A fragment of a weighted ontology with two different weighted *is-a* and *has-instance* edge.

values associated with *synonym* edge. The meaning of term “Quagmire” is more similar with “Marsh” than the term “Swamp”, so the weight for the former pair is set to 9 and is greater than the latter which has a value: 8.

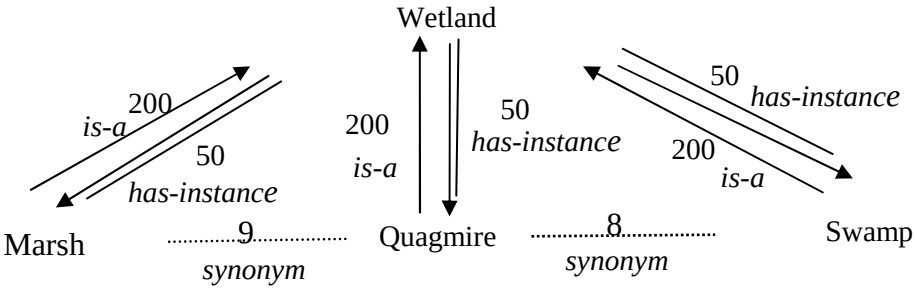


Figure 3.11: A fragment of a weighted ontology with two different weighted *synonym* edges.

Figure 3.12 shows a fragment of a weighted ontology with different weight values associated with *is-closely-related* edge. For the term “Marine reef”, “Sea

grass bed" is more closely related than the term "Marine habitat", so the weight for the former pair is set to 8 and is greater than the latter which has a value: 5.

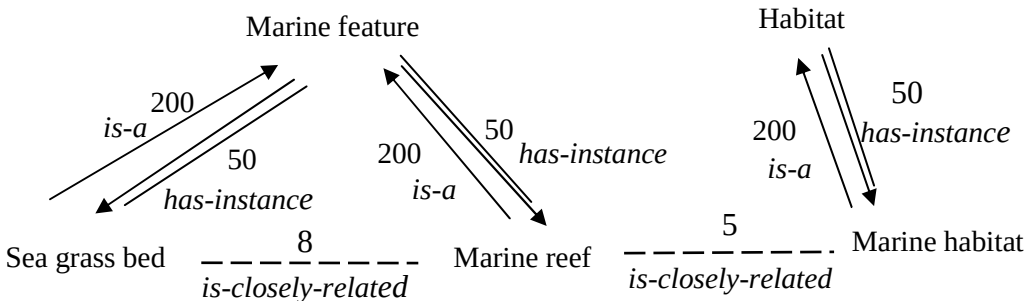


Figure 3.12: A fragment of a weighted ontology with two different weighted *is-closely-related* edges

The method of generation of the weights depends on the builder of the ontology. The weights can be assigned by hand or can be generated automatically. We have generated the weights by hand in the implementations that we have used.

3.2 Search

The basic premise of our ontology search is to use search terms from the users' request and proceed from the search terms to "near by" database terms. Weights can be combined with user interaction and define what is meant by "near by". The thresholds used to block the search are provided by users when they register with integration system.

For *synonym*, *antonym* and *is-closely-related* edges, we use 0 to represent identical and larger weight values for terms that are not as close.

To look at the search, we provide a set of basic rules used in the search.

1. When the user generates a request he/she is asked for a set of search terms inside the request. For a map request, the search terms might be a theme for the map and/or the symbolic terms referring geographic location. For a relation request, the requesting attributes are the search terms used to map to existing database terms. For the merged request, the theme and/or symbolic terms referring geographic location and requesting attributes are used to search the ontology.
2. Weights are used to block paths that are unlikely to provide useful results. For example, an *is-a* edge from a specific term to an abstract term is unlikely to yield a useful “near by” term.
3. For a typical positive search, the algorithm first locates the query node by using the search terms in the request and then starts from the query node by looking for *synonym* edges. If one is found the weight is tested against the *synonym* threshold. If the weight is larger, the search moves to the next node and continues. Whether more edges are followed from the individual nodes depends on whether we are looking for all “near by” terms or one. This can be decided by the nature of the request and what is known about the application (more details on what is known about an

application will be described in Chapter 4). If no *synonym* edge exists, then the *is-closely-related* edges are used and if no *is-closely-related* edge is found then *has-instance*, *is-part-of* and *contains* edges are used. If none of the edges mentioned above exists then the *is-a* edges are used as indicated in rule 2.

5. For a NOT search, the algorithm starts from the query node in the ontology and looks for an *antonym* edge associated with the term node. If one exists, its weight is tested against the *antonym* threshold. If an appropriate *antonym* edge is found, the search moves to the new term node and a positive search (rule 3) is initiated from that point.
6. In all cases if no “near by” term is found for a query term, the user application is notified of the request failure.
7. When all query terms have been processed, the search algorithm returns a set of references to data sources. For the relational request and merged request, not only the references to data sources is returned but also a set of references to the attributes that can be used to generate the required query(ies) are returned by the search algorithm.

3.3 Extended function ontology

Our ontology model has been extended to include an ontology of domain functions. The major purpose of this extension is to increase the richness of data available in the ontology model. A good ontology model should contain the richest information in the least space. Adding the function ontology has limited cost, while the information it contains is much richer. We will call the newly added ontology, *function ontology* and the weighted ontology, *ontology* in the remainder of this thesis.

The structure of the function ontology is similar to the weighted ontology structure except that the term nodes of the function ontology are terms referring to classes of functions or specific functions in the real world. In our previous works (Tsai et al. 2001, Tsai et al. 2003), we see the "leaves" of this ontology as being the set of implemented domain functions.

Definition 3.10:

A *function ontology* is defined as the tuple $\mathbf{F} = <\theta, T, F, R_0, R, S>$, where

θ is a weighted ontology,

T is a set of terms used as internal nodes,

F is a set of functions used as leaf nodes,

R is a set of relations that can be of the form t_1 relation t_2 or t relation f,

where t, t_1, $t_2 \in T$ and $f \in F$.

R_0 is a relation that connects a term from θ to either a term t $\in T$ or a function f $\in F$. The relations in R_0 support the types: *applies-to* and *define-as*.

S is a set of operations needed to create, maintain, and search the ontology structure defined by *T, F, R, R_0*.

Definition 3.11:

The relation type t_1 *has* t_2 is a relation between t_1 and t_2 such that t_1 is super type of t_2.

A simple example of *has* is "merge *has* vecOnRaster". The graph structure of the relation type is shown in Figure 3.13a and the example is given in Figure 3.13b.

$t_1 \xrightarrow{has} t_2$

Figure 3.13a: The *has* relation between t_1 and t_2.

merge $\xrightarrow{has}$ vecOnRaster

Figure 3.13b: The *has* relation between merge and vecOnRaster.

Definition 3.12:

The relation type t_1 *evolves* t_2 is a relation between t_1 and t_2 such that t_2 is newer version of function of t_1.

A simple example of *evolves* is "version1 *evolves* version2". The graph structure of the relation type is shown in Figure 3.14a and the example is given in Figure 3.14b.

Figure 3.14a: The *evolves* relation between t_1 and t_2.

Figure 3.14b: The *evolves* relation between version1 and version2.

Definition 3.13:

The relation type t_1 *implements* f is a relation between t_1 and f such that f is an implemented function of t_1.

A simple example of *implements* is "vecOnRaster *implements* vecRas-v1". The graph structure of the relation type is shown in Figure 3.15a and the example is given in Figure 3.15b.

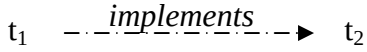


Figure 3.15a: The *implements* relation between t_1 and t_2.

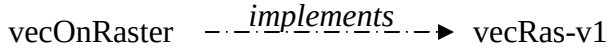


Figure 3.15b: The *implements* relation between vecOnRaster and vecRas-v1

Definition 3.14:

The relation type t *defines-as* f between t and f such that t defines f where t is a term □ *T* and f is a function □ *F*

A simple example of *defines-as* is "fusion *defines-as* merge". The graph structure of the relation type is shown in Figure 3.16a and the example is given in Figure 3.16b.

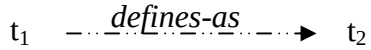


Figure 3.16a: The *defines-as* relation between t_1 and t_2.

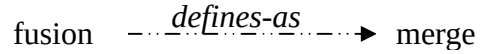


Figure 3.16b: The *defines-as* relation between fusion and merge

Definition 3.15:

The relation type f *applies-to* t between f and t such that f can be applied to t where t is a term $\in T$ and f is a function $\in F$

A simple example of *applies-to* is "merge *applies-to* roadMap". The graph structure of the relation type is shown in Figure 3.17a and the example is given in Figure 3.17b.

Figure 3.17a: The *applies-to* relation between t_1 and t_2.

Figure 3.17b: The *applies-to* relation between merge and roadMap

Figure 3.18 shows fragments of ontology and function ontology. The term "fusion" has relation *defines-as* with a term "merge" in the function ontology while "merge" has relation *applies-to* with the term "roadmap" in the ontology.

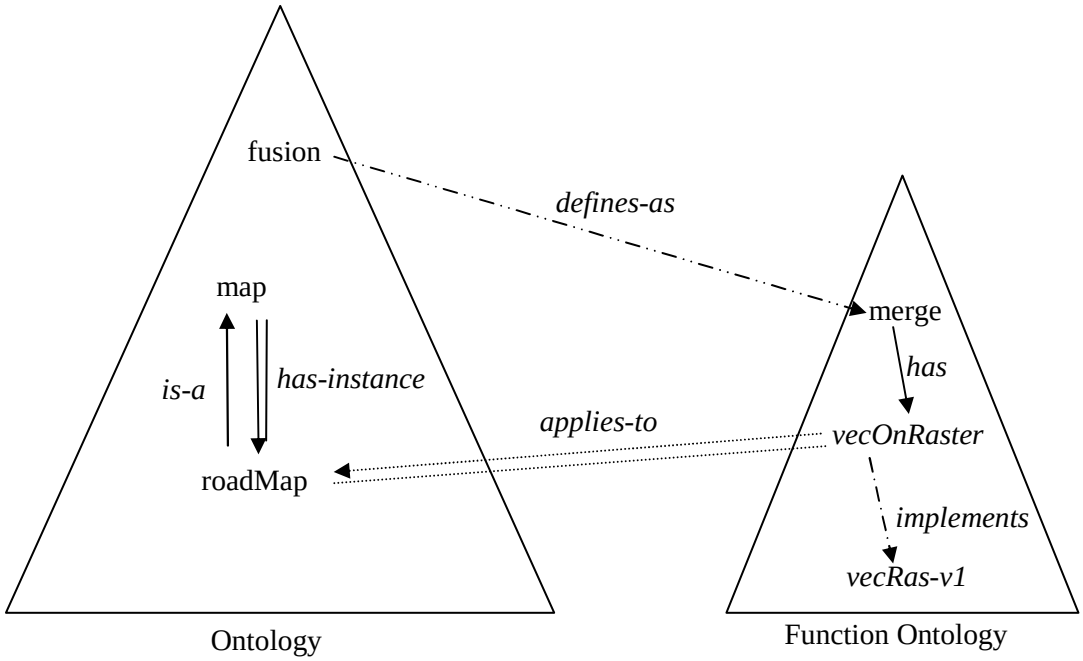


Figure 3.18: Fragments of ontology and function ontology.

The current version of our function ontology handles software versions by using a term to indicate the version number. Figure 3.19 shows a simple example

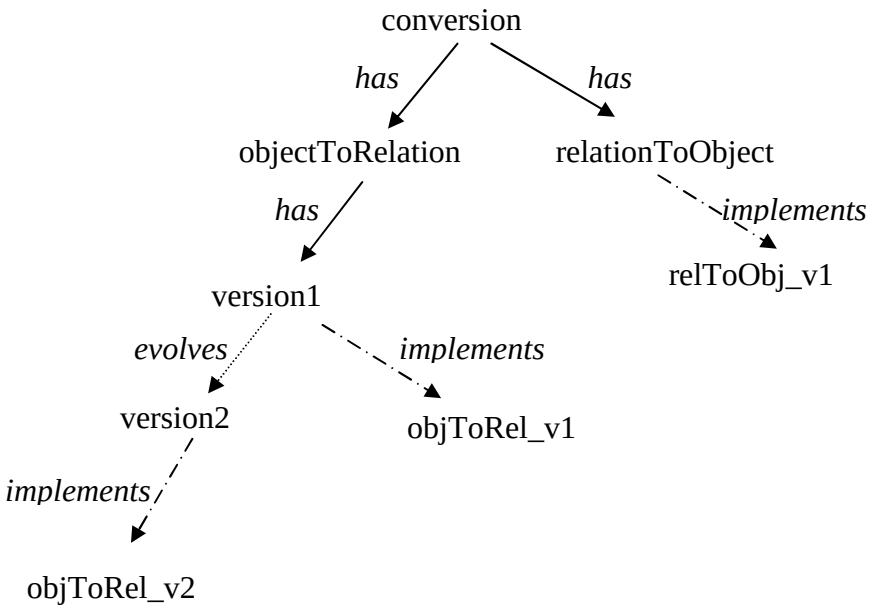


Figure 3.19: Fragment of the function portion of the function ontology showing how versions are maintained in the model.

3.3.1 Function search

To look at the search of function portion of the function ontology, we provide a set of basic rules used in the search.

1. The search inside the function ontology starts with a search term from the uers' request. Note that the search always starts in the weighted ontology by applying the search algorithm described in Section 3.2.

2. If the search follows a *defines-as* edges that point into the function portion of the ontology then search inside the function portion of the function ontology then starts.

3. The search follows either *evolves* edges that lead to a newer version of the function or *has* edges to a term in the function portion of function ontology with more specific meaning.

4. The search continues downwards to the terms in the lower level, the search follows *implements* edges and then halts a function at the leaf level is reached.

3.4 Semantic network model

In the spatial mediator, the Semantic Network Ontology (***SNO***) is used to search the data sources that can answer users' relational queries. We first briefly

introduce what a RLIV is here in order to describe the semantic network. A formal definition of RLIV is presented in Chapter 4. The spatial object is a view type object (program) that is used by the data sources to provide a mechanism to make the local data available to use. A RLIV is a spatial object. In our current model we have restricted them to represent relations. Due to our assumption that RLIVs represent relations, we use equijoin in the current ***SNO*** model definition for combining RLIVs. Note that this can be expanded to include other data type (e.g. objects) in the future.

Since semantic data model is a phrase that has seen a lot of overuse, we start by providing what we envision as a semantic data model.

Definition 3.16:

A *semantic data model* is a t-tuple $\vartheta = <R, T, A, L, O>$, where

R is a set of RLIVs,

T is the set of terms stored in the RLIVs,

A is the set of association (equijoins) used to show how RLIVs can be combined (joined),

L is the set of links that connect the RLIVs to the associations, and

O is the set of operations for operating on the graph created by connecting the RLIVs to the appropriate associations using the links.

Definition 3.17:

The *Semantic Network Ontology* is defined as a tuple **S** = <θ, ϑ , *R*, O>, where

θ is the weighted ontology,

ϑ is a semantic data model,

R is a set of relations that connect the terms in θ with terms in ϑ, and

O is the set of operations needed to create, maintain and search the *SNO* structure.

Two RLIVs are connected by an association node that describes how to combine the data from the two sources. For example, if the two RLIVs from two data sources are biology data, such as protein data that appear to the user as relations in the relational databases, the association node defines how the two RLIVs can be joined (Figure 3.20). Note that the example has been chosen to point out that the association nodes can support more than simply equijoin.

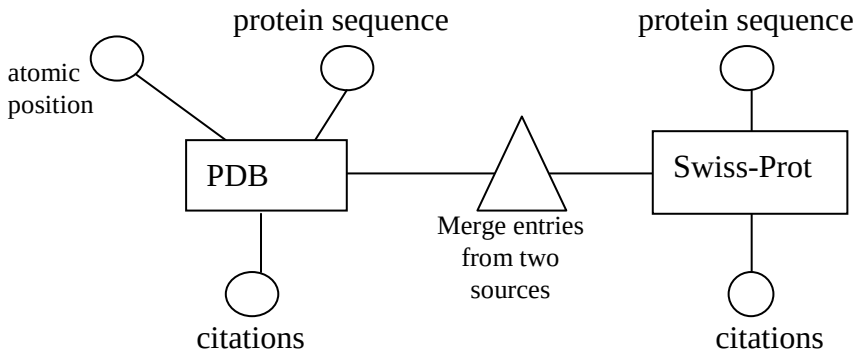
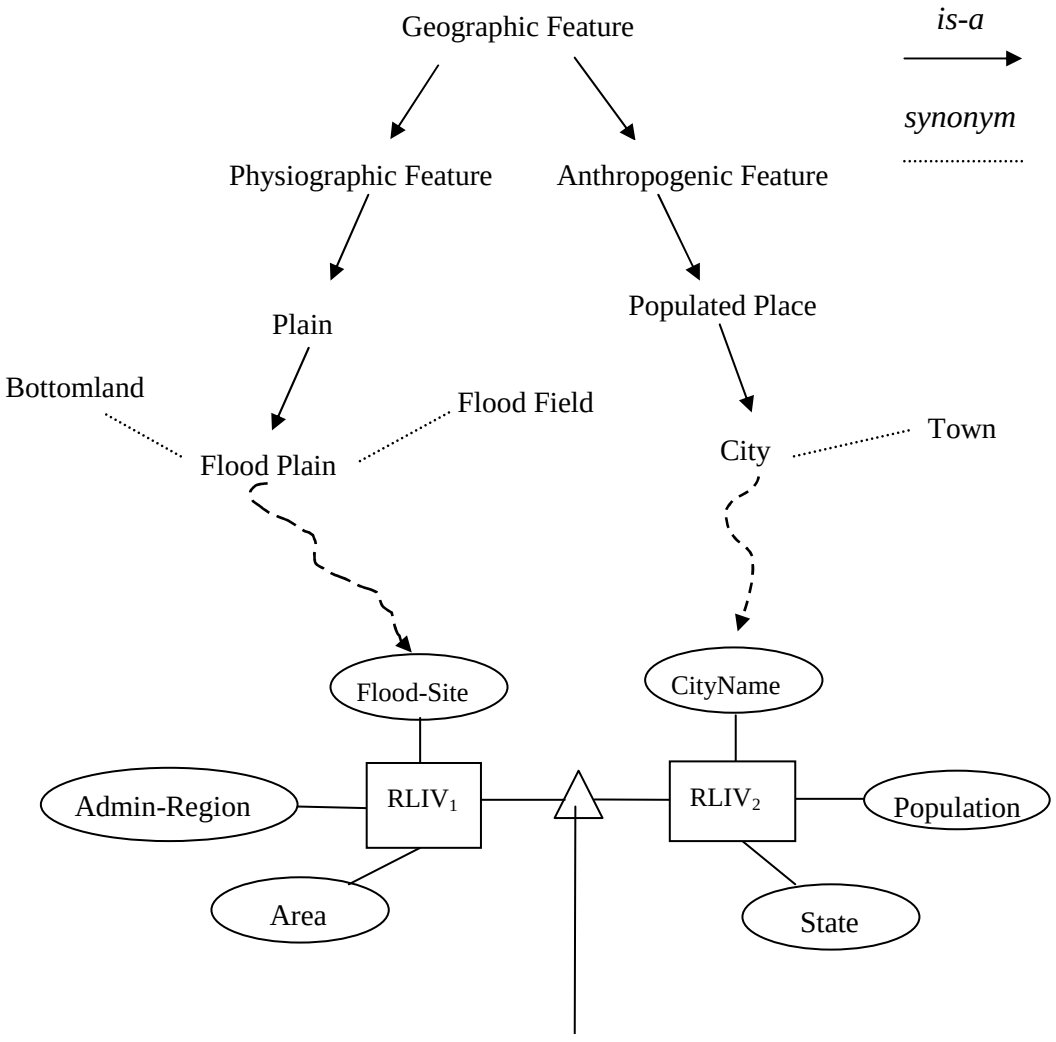


Figure 3.20. Example of two data sources in the semantic data model.

Figure 3.20 illustrates the entries in the semantic network for the RLIVs provided by PDB and SWISS-PROT databases. The resulting semantic network is connected to the ontology model by a process where each property node is attached to a node in the ontology.

Figure 3.21 shows a fragment of a *Semantic Network Ontology*.



($RLIV_1$, $RLIV_2$, $RLIV_1.Admin\text{-}region = RLIV_2.State$)

Figure 3.21: A fragment of Semantic Network Ontology.

The example in Figure 3.21 shows two RLIVs, namely $RLIV_1$ and $RLIV_2$, are connected through the semantic network. Two entity nodes each representing one RLIV have the properties associated with it. These properties are the attributes of the relations inside the RLIVs. For example, there are three attributes associated with $RLIV_1$ and they are “Admin-Region”, “Flood-Site”, “Area”. The join criteria specified by the triangle in the figure indicates that these two RLIVs can be joined by the attribute “Admin-Region” of $RLIV_1$ and the attribute “State” of $RLIV_2$. The application is interested in which towns are located in a flood-risk area. It uses terms: “FloodField” and “town” to request the information. The system uses these two terms to start the search algorithm mentioned in the previous section and search through ontology. Since “FloodField” and “town” are synonyms to “Flood Plain” and “City” respectively, the search algorithm locates $RLIV_1$ and $RLIV_2$ that contain required data. Note that the leaf node in the ontology has a reference to the corresponding attribute which is indicated by a dashed-line with an arrow in the figure.

The tools in ϑ for generating corresponding queries to query the data of the relations within the RLIVs are elaborated in detail in Chapter 4.

CHAPTER 4. SPATIAL MEDIATOR MODEL

In this chapter, we present the formal definition of the spatial mediator as well as the infrastructure in which the mediator is engaged. However, we must point out that our contribution of this study comes from the formal definition of the overall infrastructure, the registration process, and the model, algorithms and the evaluation of the spatial mediator. The infrastructure referred to as GeoGrid is the final outcome of a group project. Several components of GeoGrid have been implemented by members of the group. The spatial mediator server is the core component of GeoGrid.

The motivation and the overall structure of the GeoGrid infrastructure is introduced in Section 4.1. The role played by the spatial mediator in the infrastructure is also described. The overview of the spatial mediator model is presented in Section 4.2. The two major tasks performed by the spatial mediator, namely map generation and relation generation are described in detail in Sections 4.3 and 4.4, respectively. Section 4.5 looks at the Integration Script produced by the spatial mediator. The Integration Script defines the process that guides the creation of the final result requested by the user.

4.1 Motivation

As mentioned in the Introduction chapter, geographic data is very diverse and dynamic. The geographic information may be structured, semi-structured or unstructured and usually there is no regular schema to describe it. As the amount of geospatial data grows, the problem of interoperability between multiple geospatial data sources becomes the critical issue in developing distributed geospatial systems.

We use the following example as our motivating scenario. An application requests the tornado information of a particular place and time, for example, the state of Alabama in 2011, in a distributed data source environment. The application requests a map which indicates locations where tornados happened, it also asks for the detailed data regarding the tornados such as the scale of each tornado, date, time, etc. The purpose of the application is to look for regions that have been under attack by F4 scale tornados and study the path of tornados. In the following paragraphs the process required is described if the system is to respond to application' request.

In this example, two different forms of data are needed, namely a map and relational data. The system must decide which data sources contain related relational data or the maps needed to respond. It then needs to select the data sources if the data resides in more than one place. Finally, the system must have the capability to merge the map with relational data in order to respond to the application.

Many approaches have been proposed to provide solutions to the integration problem. Among them we see mediation, an information integration strategy, as one of the most appealing approaches. The mediator concept was introduced by (Wiederhold 1992). In Wiederhold (1992) mediators were defined as components occupy a layer between the users, applications and the data sources. Mediators provide intermediary services between these parties. A mediator is build to provide a uniform interface to a number of heterogeneous data sources. Given a user request, the mediator defines the process that decomposes the request into multiple local sub-queries, sends them to the appropriate data sources and merges the partial results and reports the final answer to the user. We first introduce the GeoGrid infrastructure and then the details of spatial mediator in the following sections. GeoGrid was developed to provide geographic information to a distributed mobile environment (Nusser et al. 2003, Miller et al. 2004, Miller et al. 2001).

4.1.1 GeoGrid infrastructure

GeoGrid is modeled as a directed graph G(N,E). N is a set of nodes with some processing power focused on supporting the GeoGrid infrastructure. The edges in the edge set E represent the communication links that tie the components of GeoGrid together.

The set of node types T = {*user, spatial mediator, computation server, data source, tool, registration*} represents the infrastructure component types in GeoGrid. A wrapper is associated with each node type to simplify integration of the components in this computing environment and to standardize communication requirements. Figure 4.1 provides a simple illustration of the directed graph formed by GeoGrid.

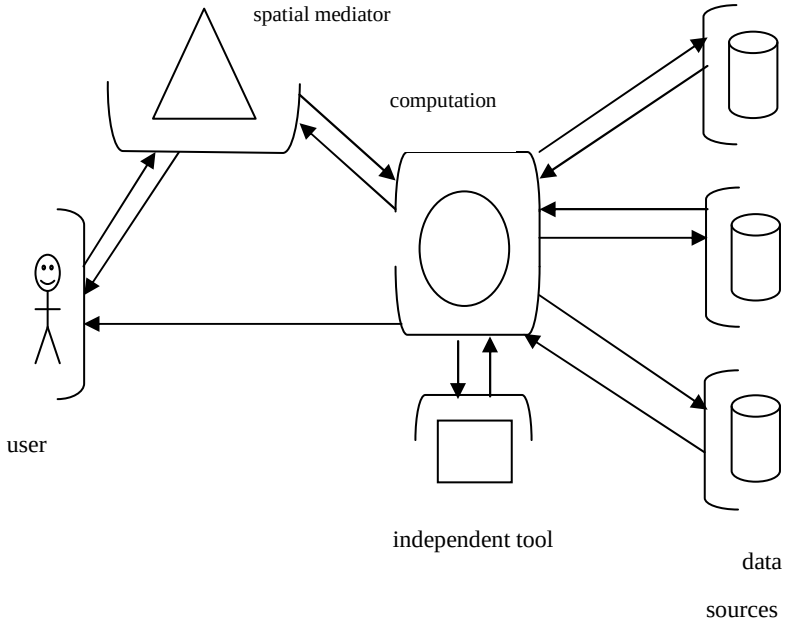


Figure 4.1: A simple example of a GeoGrid graph showing the nodes and the data flow for a mediated data request.

4.1.1.1 User

The user node represents the user application that generates the initial request for the geographic data that has to be up/down loaded. The user's device can either be stationary or mobile. The user application and the device that the application is

running on are wrapped by a user wrapper (Nusser et al. 2003; Miller et al. 2004, Miller et al. 2001). The user node (primarily through the wrapper) is responsible for determining the next node (usually the mediator) required to complete the request, formulating the request in the format expected by the next node's wrapper, initiating moving the request object to the next node, and preparing to receive the data requested. The details of how user nodes were implemented are given in (Zou 2004).

4.1.1.2 Registration

The registration node supported by GeoGrid provides a window into the spatial mediator for potential users and data suppliers. Independent tools (i.e., tools available outside of the computation server) have to be registered as well. Due to its importance in the mediation process, the details of registration process are presented in Section 4.2.

4.1.1.3 Data sources

The basic structure of a data source is given in Figure 4.2. The local interface view (LIV) (Yen et al. 1994, 1995, 1998) is designed to export data from the data source into the GeoGrid environment. The number and type of LIVs is a local decision dependent on how the local information manager wants to share the available

data within GeoGrid. The details on how the data sources were implemented is given in (Qu 2003).

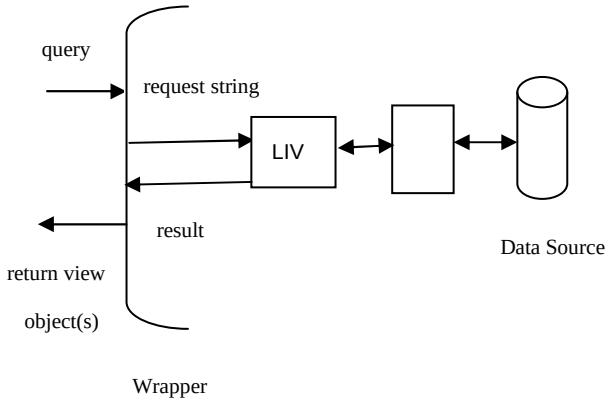


Figure 4.2: Data source node layout and request/data flow for retrieval.

4.1.1.4 Independent tools

A similar structure has been used for independent tools. The tool interface converts the incoming data to the format expected by the tool and converts the results to the object format expected by the wrapper. The registration process for a tool node defines the tool type (i.e., its functionality) and the local interface views used to move data to and from the tool.

4.1.1.5 Computation server

The major task performed by the computation server is to provide the facility to execute the integration script it receives from the mediator and store the results of the subqueries in the integration script. It also provides tools to operate on the results of the individual queries. Once the integration script is received from the spatial

mediator it is parsed into the individual tools and query components. The components are then used to initiate and send the subqueries to the individual data sources. Upon the completion of the request the computation server sends back the result to the user and a message indicating the successful completion the request is also sent to the mediator. The computation server is described in (Ming 2006).

4.1.2 Interaction between spatial mediator and infrastructure

The spatial mediator is connected to three types of components in a GeoGrid environment. The first is the connection between the spatial mediator and the user nodes. User nodes generate requests that go to the spatial mediator and the spatial mediator sends acknowledgement and messages back to the user's wrapper. The second edge type links the computation server to the mediator and is used as the communication link over which the integration script is sent to the computation server from mediator and computation server sends acknowledgement and messages (the most important message is that the data indicated in the integration script is not available) back to mediator. The last edge type is used to communicate with the registration node(s). The mediator receives registration data from a registration node and sends acknowledgements and messages back to the registration node. The mediator populates the Fact Database and the rule sets with the registration data received from the registration node(s).

4.2 The spatial mediator model

4.2.1 Local Interface Views (LIV)

Before we present the spatial mediator model we first introduce the Local Interface View (LIV). The basic unit of communication between the spatial mediator and other components in GeoGrid is an object view. The local interface view is a view type object that is used by the local data administrator to provide a mechanism to make the local data accessible to the GeoGrid infrastructure. The object view type is defined as being an extension of the object model (EOM). The use of views in this model is an extension of the work on the Zeus View Mechanism given in (Yen et at. 1994, 1995, 1998). The views have a traditional object structure (attribute and methods) with the restriction that they support a derivation method. The derivations method is used to generate the public and private attributes of each object instance created through a view. The individual data sources are expected to have local control. The local interface allows distribution transparency and representation transparency, while hiding or converting (mapping) some of the data from the data source. The local interface view belongs to the local data source. It interacts directly with the data source and passes the result to the wrapper which controls communication between the GeoGrid components. A given data source and its wrapper can support multiple local interface views in order to present its data to different applications or users.

The views are developed by the owners of the data source and are registered for use to the GeoGrid infrastructure.

There are two types of Local Interface Views used inside the GeoGrid infrastructure. One is the map type, MLIV. MLIV is defined as a view type object that provides access to a map object from a data source in GeoGrid. Each data source that provides maps contributes one or more MLIVs, where each MLIV can be used to generate a map. The other type of spatial object used in GeoGrid is the RLIV. For this thesis, we have restricted RLIVs to operate on sets of relations. The RLIVs is defined as view type object that provides the access to data inside the data source and have the capability to transform the data into a relation. Examples of MLIVs and RLIVs are given in the description of the mediation process.

4.2.2 Registration process

Every participant providing data, tool or user applications in GeoGrid is required to register with the infrastructure. The registration process provides two kinds of information to the spatial mediator. First, it provides the information necessary to link new nodes into the GeoGrid infrastructure. In addition the registration process gathers the facts and rules about new nodes. This information is stored in the Fact Database and/or Rule Set. It's needed in the mediation process to make it possible to reliably use the new nodes. The required registration data varies

depending on the type of node the participants are introducing. We explain several important registration data that support the mediation process of the spatial mediator. The details of registration data are given in Appendix A.

Designers of new user applications are required to register their applications and the device types that the applications will use in the field. The capability of device display capability is also required when registered with the infrastructure. An integer variable named “screen code” is used to specify the capabilities of an application to display result and the return data type requested by the application. For example, the mediator interprets the value 1 to indicate the requested type from the application is a map and the device can only display one map at a time without panning function. When the mediator receives the request from the application it checks the Fact Database and learns the screen code associate with the requesting application has the value 1. Information on the application and devices are used in the mediation process to help guide the generation of the integration script.

Data sources can be made available to GeoGrid by registering the data sources and the local interface view(s) that will provide the mediator a view of the data that is being made available from the data source. As we mentioned in the previous section, the communication unit in the GeoGrid are the LIV objects and that there are two

types of LIVs used in GeoGrid, namely the MLIV and RLIV. If a data source can provide multiple LIVs then it must register each LIV separately.

To register a MLIV the person registering it must provide the following information:

- A set of geographic coordinates specifying the point or bounding box covered by the MLIV. In our implementation, the decimal degree latitude/longitude is chosen for the ease of use in the algorithms.
- The theme(s) of the MLIV specifying the geographic features supported for the map. For example, a MLIV with a theme: “lakes” indicates it can provide maps with lakes.
- Symbolic terms representing geographic location covered by the MLIV. For example, a MLIV with a symbolic term: “Midwest” indicates it can provide the maps of Midwest region of U.S.
- Values for quality attributes such as completeness, positional accuracy, accessibility, reliability, resolution and file type. The quality attributes required are based on the metadata suggested in the *Content Standard for Digital Geospatial Metadata Workbook* (FGDC 2000) published by FGDC (Federal Geographic Data Committee).

For each RLIV that operates on a set of relations, the person registering it needs to register the following information:

- Every relation schema made available to the infrastructure. This includes name of relations, the attributes and attribute data types.
- Key attributes are required. This information is made to support mediation process of the spatial mediator.
- The set of functional dependencies describing the semantics of the data covered by the RLIV.
- Whether the relation schemes defined by the RLIV supports the universal relation property (i.e. is the join of the relations defined by the RLIV lossless).

Independent tool nodes also need to register with the infrastructure. Registration information includes tool type, tool name and parameters for the tool. Tool types supported by GeoGrid include *combine, crop, convert, scale, merge* in our current implementation. Our spatial mediator can easily be extended to support additional types by either adding independent tools or by adding more tools to the computation server.

4.2.3 Overall mediation process

The mediation process starts with a user application in the field sending a request to the spatial mediator. The spatial mediator then generates an integration script defining the tools and the data sources needed to generate the requested spatial object(s). The integration script is passed to the computation server where it is used

to execute the process of obtaining the data and creating the spatial object(s). The resulting spatial object(s) which are either a map, a set of tuples or a map merged with some spatial data is then passed back to the user application.

There are three types of requests handled by the mediator, namely map requests, relational requests and merged requests. A map request looks for data that can generate a map that satisfies the request requirements. A relational request asks for the spatial data in the form of a set of relational tuples. The third type of request, the merged request, builds on the previous two. That is, the spatial data is displayed on top of the map in the form of icons. Different types of requests contain different requested properties. We explain each type of request in detail in the following paragraphs.

(1) The map request:

A map request contains location requirements, property requirements and theme requirements. The location requirements are either in geographic coordinate data format or a symbolic term referring a geographic location. The geographic coordinate data has three forms: (1) a point specified by a latitude and longitude pair (2) a point and the radius (3) a set of latitude and longitude pairs specifying the lower left and upper right corners of a bounding box. Property requirements of the map request are preferred values for the quality attributes namely, completeness, positional

accuracy, accessibility, reliability, resolution and file type. The theme is a term specifying feature of the map required, for example: river, soil, etc. The following are some examples of map requests.

Example 1: The map request with request id *r1001* is asking for a *soil* map with a bounding box of *<41.5233, 93.1402>, <42.1341, 94.1435>* and has the following quality request: a *100%* (complete) coverage(encoded as 1.0), a *SHAPE* map file type, *0.03* m for the positional accuracy, a *90%* reliability of the data, a *100* m resolution and accessibility is *25* seconds.

< r1001,(<41.52, 93.14>,<42.13, 94.14>), (<1.0, SHP, 0.03, 0.9, 100, 25>, soil) >

Example 2: The map request with request id *r1002* is asking for a *wetland* map with a center point at *<41.5211, 93.1463>* and a *50 m* radius and has the following quality request: a *90%* coverage, a *TIFF* map file type, *0.01* m for the positional accuracy, a *100%* reliability of the data, a *10 m* resolution and accessibility is *5* seconds.

< r1002, (<41.5211, 93.1463>, 50m), (<0.9, TIFF, 0.01, 1.0, 10, 5>, wetland) >

Example 3: The map request with request id *r1003* is asking for a *prairie* map of *Midwest* and has the following quality request: a *80%* coverage, a *JPG* map file type, *0.02* m for the positional accuracy, a *90%* reliability of the data, a *25* m resolution and accessibility is *45* seconds.

< r1003, (Midwest), (<0.8, JPG, 0.02, 0.9, 25, 45>, prairie) >

(2) The relational request:

Relation request contains request id and the requesting attributes and/or conditions. The following is an example of relation request.

Example 4: The map request with request id *r1004* is asking for the data (owner name and owner address) of the property in the state of Iowa whose area is larger than or equal to 2000 sqft.

< r1004, (ownerName, ownerAddress), (ownerState = "Iowa" and sqft >= 2000) >

(3) The merged request:

A merged request contains request id, location requirement, property requirement, theme and requesting attributes and/or conditions. The following is an example of a merged request.

Example 5: The map request with request id *r1005* is asking for a map exactly the same as example 4 but also requests the data (owner name and owner address) of the property in the state of Iowa whose area is larger than or equal to 2000 sqft. The application that generates this request is looking for the topology relationship between big houses and prairie in the Iowa. The final result for this request is a tabular form of some spatial information along with a map with some icons indicating requested houses on it.

< r1003, (Midwest), (<0.8, JPG, 0.02, 0.9, 25, 45>, prairie), (ownerName, ownerAddress), (ownerState = "Iowa" and sqft >= 2000)>

Upon receiving the request, the mediator first identifies the request type and then starts the corresponding process. The spatial mediator initiates the generating map script process for a map request (if one exists) and starts the generating relational script process if one exists. As to the merged request, the mediator first starts both processes and then generates an integration script specifying a merge type tool is required based on the map and relation generation scripts.

In general, the mediator has two equally important tasks. First, it has to be able to locate the data given a request. Second, once the data is located it must be able to bridge the semantic gap between the user's request and the existing data, perform

data manipulation operations needed to query the data sources and integrate the results from the individual such queries.

If users request a map with symbolic terms then the spatial mediator uses the symbolic terms to search Geographic Symbolic Ontology to locate potential MLIVs that can respond the request. If users provide geographic coordinate data then the mediator uses the R_Tree structure to find the relevant MLIVs. A rule Set and Fact Database are also used in the process to generate the map script. The generation of the map script is described in detail in Section 4.3.

As to the relational request, users are required to provide information on requested attributes. The spatial mediator first makes use the Semantic Network Ontology to search for RLIVs and then uses the Semantic Network Ontology and Fact Database to generate a relational script. Details of the relational process are given in Section 4.4.

Figure 4.3 shows a block diagram of the process of the spatial mediator. Both generating map script process and the generating relational script process displayed in the bold outlined rectangles will be presented in detail in Sections 4.3 and 4.4, respectively.

The detailed descriptions of each component of model are presented in the next subsection.

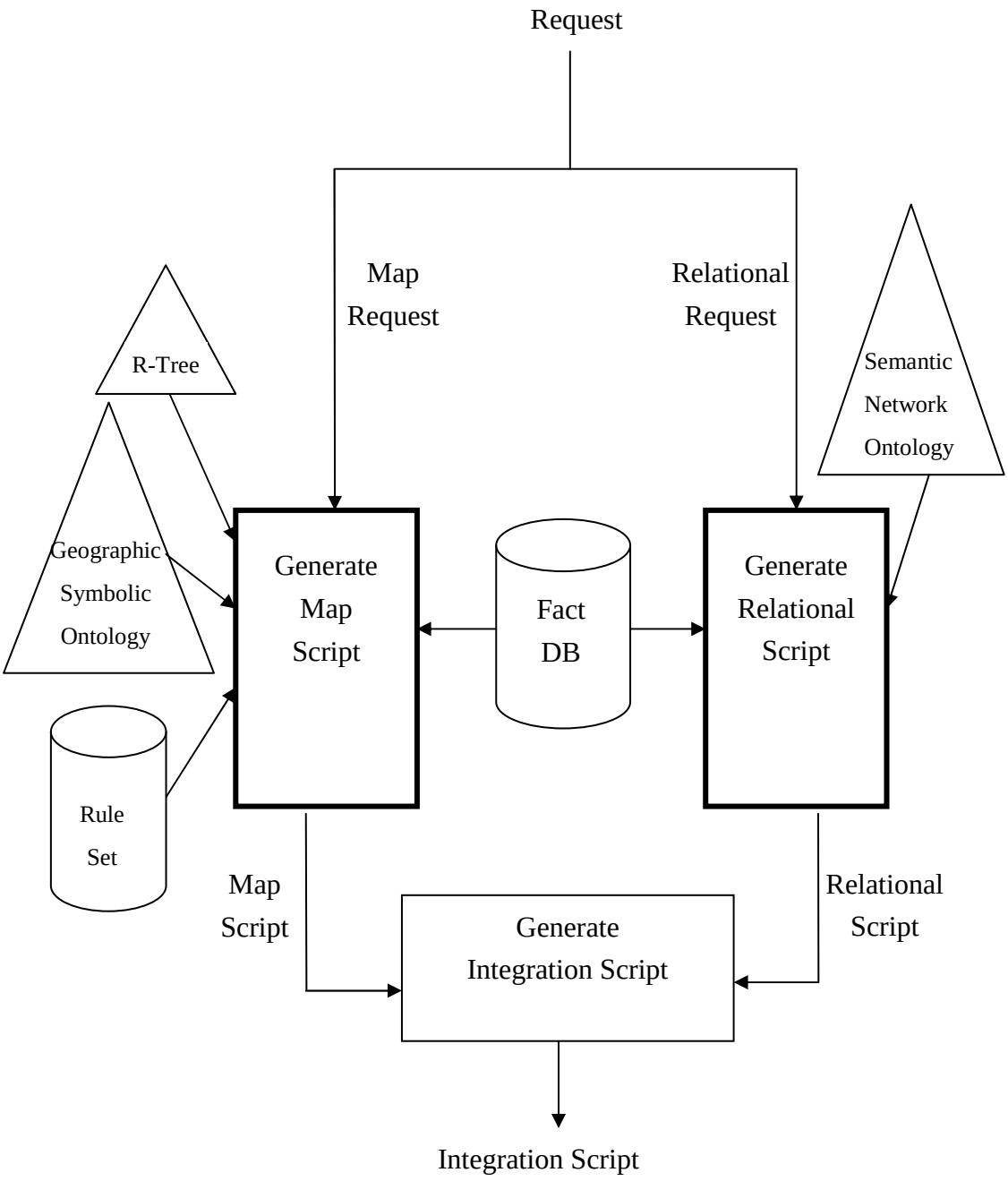


Figure 4.3: Overall process of the spatial mediator.

4.2.4 Major Components

The spatial mediator model is defined as 7-tuple $\boldsymbol{M} = <\boldsymbol{\Psi}, \boldsymbol{RT}, \boldsymbol{RS}, \boldsymbol{GSO}, \boldsymbol{SNO}, \boldsymbol{FD}, \boldsymbol{SM}>$ where $\boldsymbol{\Psi}$ is the mediation process that was overviewed in the previous section and the rest of components are defined in the following paragraphs.

4.2.4.1 R_Tree: *RT*

The component ***RT*** ={η, ***rt***} where η is a set of operations and ***rt*** is a spatial index that utilizes the R_Tree structure. The ***rt*** is adopted from R-Trees proposed by Guttman (1984). It is an index structure used for spatial objects retrieval. The data structure splits space with a hierarchically nested, and possibly overlapping, minimum bounding rectangles (MBRs, also known as bounding boxes). η consists of the search operation that searches through the R_Tree index structure to identify the MLIVs that match the search conditions. The search operation makes use of the following detecting functions to determine topological relationships between MLIVs and the geographic coordinate data of the incoming map request. For the component ***RT*** to identify a MLIV there exists at least one of these functions that returns TRUE.

- *Include(Point, Polygon)* is a function that determines the spatial coincidence of points and a polygon. It can be used to identify the bounding box that contains the requested point. It returns TRUE if the point meets the following two conditions: (a) the point is located inside the polygon and (b) the point doesn't touch the borders of the polygon

- *Overlap(Polygon, Circle)* is a function that determines whether the polygons is overlapped with the circle region. It returns TRUE if the circle and the polygon overlapped. To overlap, the circle and polygon must include at least one point.
- *Overlap(Polygon, Polygon)* is a function that determines whether two polygons overlap. It returns TRUE if two polygons include at least one point.

4.2.4.2 Rule Set: *RS*

RS is the rule set defined as follows:

RS = {*r*}, where *r* is a rule that contains three clauses: *IF clause, THEN clause and ELSE clause. IF clause* contains Boolean expression. *IF clause* and *THEN clause* are mandatory while *ELSE clause* is optional.

In the process to generate a map script the spatial mediator first locates MLIVs and then uses a ranking mechanism to rank the MLIVs. It makes use of rules from the rule set ***RS*** to generate the value of the parameters used in the ranking mechanism. The ranking mechanism is introduced in Section 4.3.3. In our current model, the rules remain static while the mediator is running. Contents of *RS* are included in the Appendix B.

4.2.4.3 Geographic Symbolic Ontology: *GSO*

GSO is the geographic symbolic ontology which is based on our ontology work described and defined in Chapter 3. An example is of a fragment of ***GSO*** is illustrated in Figure 4.4.

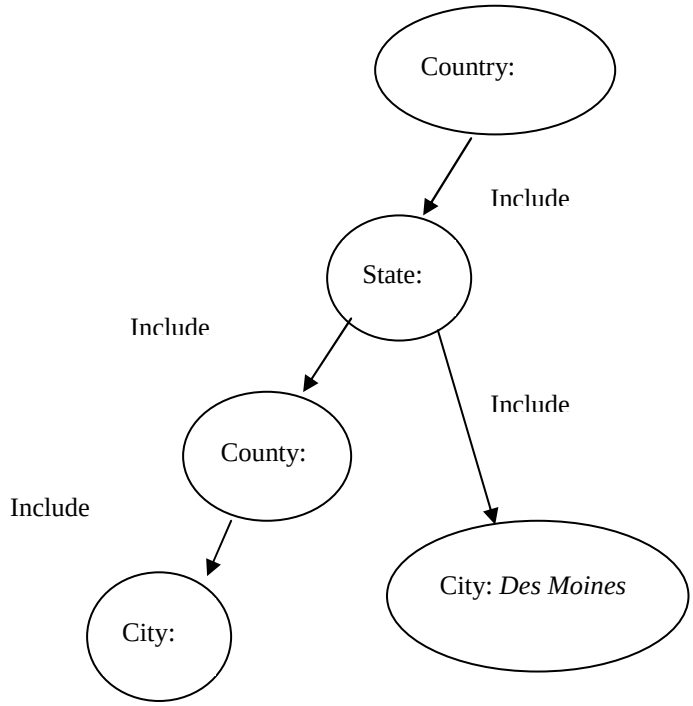


Figure 4.4: A fragment of Geographic Symbolic Ontology.

4.2.4.4 Semantic Network Ontology: *SNO*

SNO is the semantic network ontology defined in Chapter 3. The search terms from the relation request are used to search the ontology to locate the RLIVs that contain the attributes necessary to respond to the user request. The join criteria in the semantic network connecting two RLIVs is used to combine them into one relation. A fragment of a sample ***SNO*** is shown in Figure 4.5.

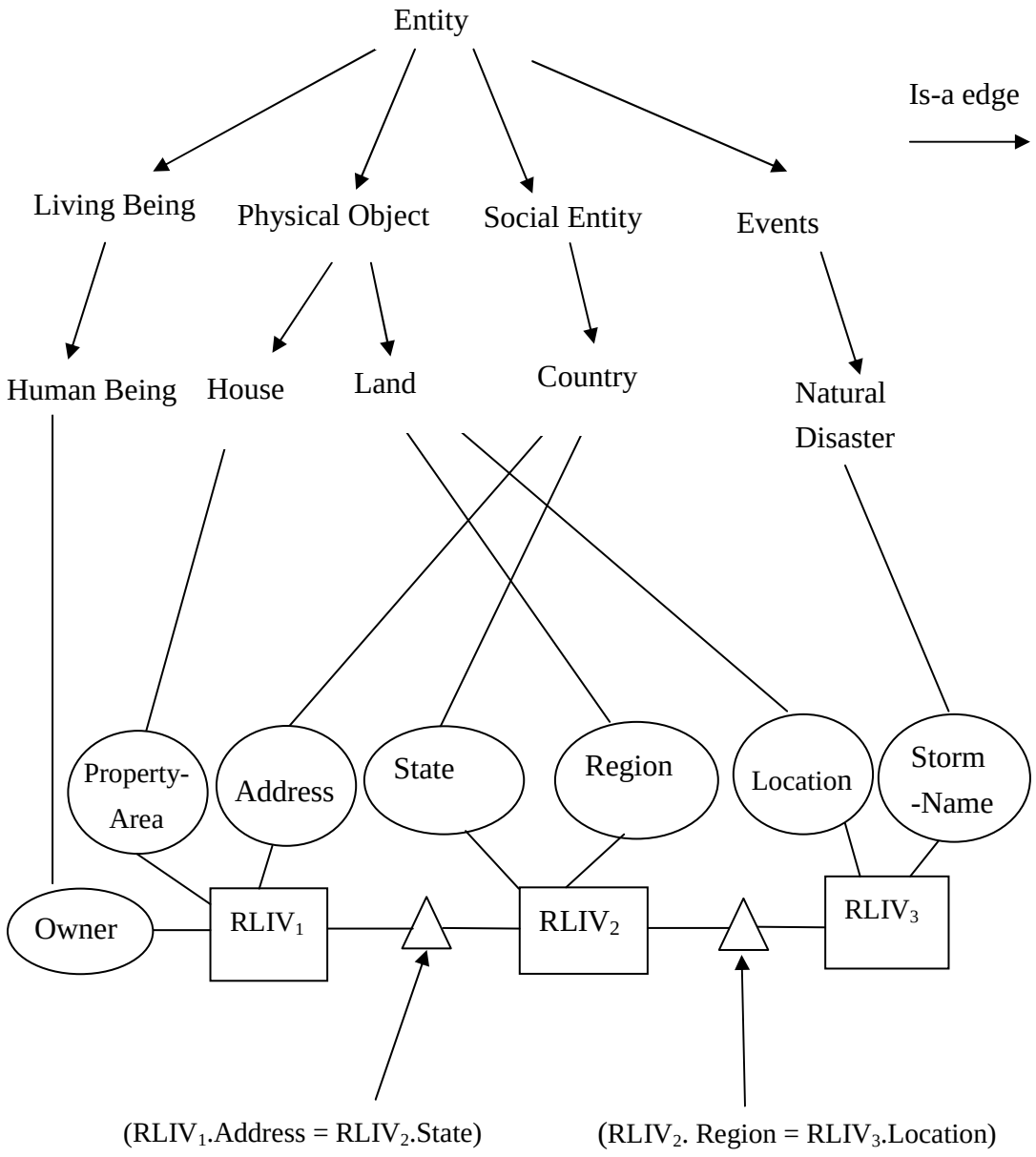


Figure 4.5: A fragment of Semantic Network Ontology.

4.2.4.5 Fact Database: *FD*

FD is a database based on the relational data model. It stores the metadata of the applications, LIV objects from the data sources and either computation server tools or independent Tool Nodes in the GeoGrid infrastructure.

The metadata suggested by FGDC (Federal Geographic Data Committee) (FGDC 2000), in particular, “Content Standard for Digital Geospatial Metadata” includes seven categories: identification, data quality, spatial data organization, spatial reference, entity and attribute, distribution and metadata reference. Among them, several attributes in data quality associated with MLIVs which provide the spatial objects in a map form in our infrastructure are stored in the ***FD***. These data are recorded into ***FD*** when data sources registered with the infrastructure. Besides the spatial characteristic metadata, the spatial mediator also document related characteristics, such as reliability of a data source which can be obtained from a statistic data maintained by mediator.

For the RLIVs that provide spatial data in a relational form, the metadata of relations within each RLIV are stored in the ***FD***. The metadata includes relational schemes, data type of the attributes within the relations and functional dependency within each relation. The spatial mediator makes use of these metadata to generate the integration script.

When a user node is registered within the infrastructure, the characteristics and functionality of their display devices are also recorded in the ***FD***. These information help the spatial mediator decide what type of data needs that the application is requesting. The geographic quality requirements for applications used in user nodes

are also stored in the ***FD***. A preference selection on geographic quality attributes indicating which attributes are more important than other attributes is completed during registration process. This preference is stored in the ***FD***. The spatial mediator makes use of this information to find MLIV that can generate the requested map(s).

4.2.4.6 System Manager: *SM*

SM is the system manager of the spatial mediator. It is a collection of programs and performs the mediation process. The ***SM*** is further divided into the following modules:

1. Administrator:

Functions:

- Evaluates the request after receiving a request from the wrapper of the spatial mediator to decide the type of the request. It then invokes the corresponding mediation process
- Coordinates the operation flow between modules inside the mediator manager ***SM***
- Monitors the meditation process and records status data for every request.
- Manages information received from the registration node which receives registration information from participants of the GeoGrid.

2. Ranker:

Functions:

- This component is activated by ***SM*** and perform the following function if the incoming request is a map request or a merged request. It doesn't perform any task for a relation request.
  i. Perform the ranking mechanism to identify the data source that can provide the best answer in term of map quality.

3. Script Generator

Functions:

- For the map request, it performs the following function:
  i. Enforce the map group algorithm to generate map groupings
  ii. Replace the tool type in a map grouping to generate template skeleton and then transform the template into the map integration script
- For the relational request, it performs the following functions:
  i. Generate the relation framework query
  ii. Create the subquery for each MLIV and generate the relation integration script
- Generate the final integration script where the returned type maybe "map", "relation" or "merged"

4. Curator:

Functions:

- Maintains metadata and rule sets inside *FD* and *RS*, its job includes updating data, periodic back up.

We elaborate the map generation corresponding map request in the following section. The relation generation is discussed in detail in Section 4.4.

4.3 Map script generation

After the spatial mediator identifies that the request includes a map request it starts the map script generation process. A block diagram of this process is depicted in Figure 4.6. We elaborate the process in the following sections.

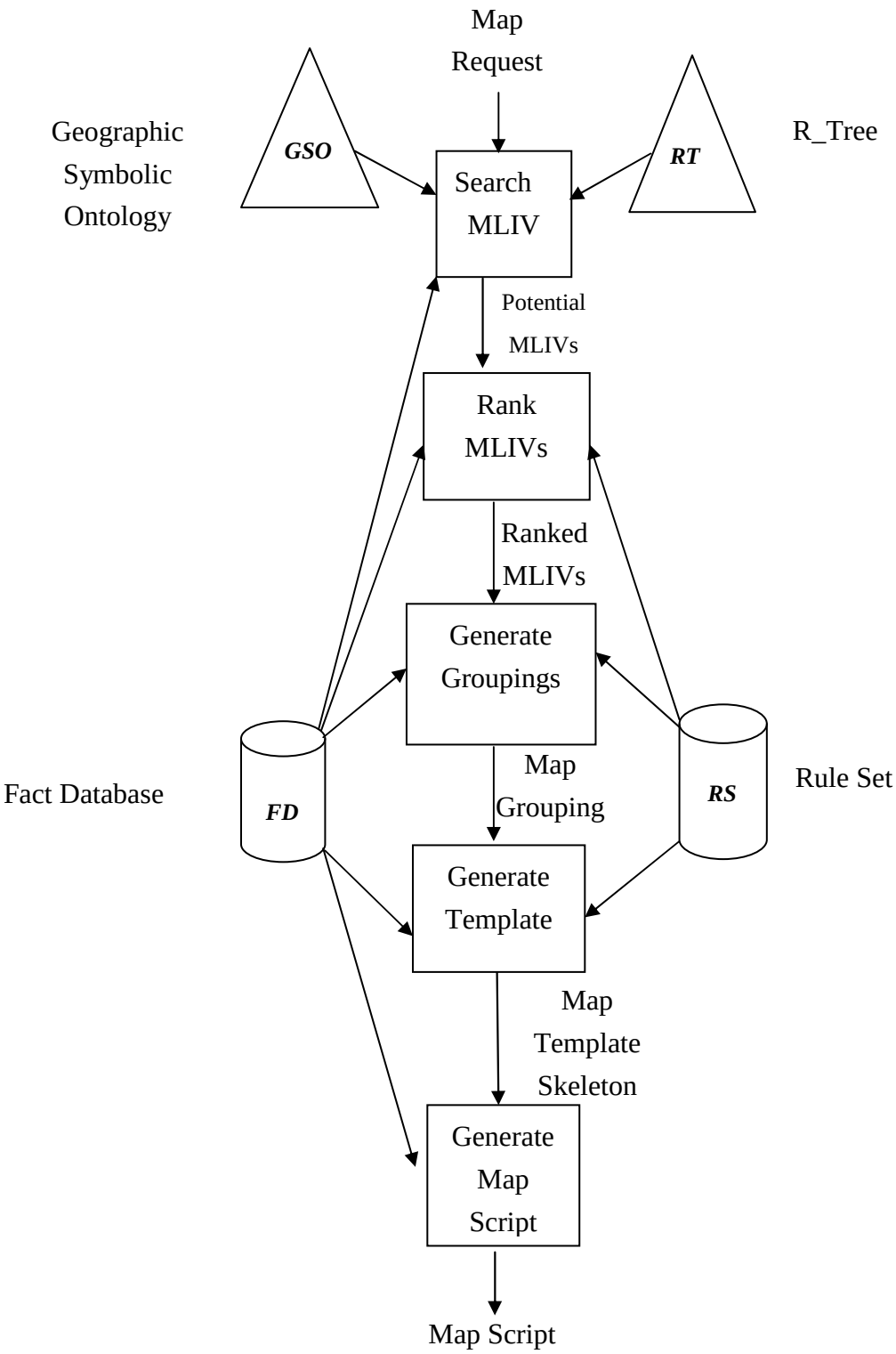


Figure 4.6 A block diagram of the mediator components that generates map scripts.

4.3.1 An example of MLIV

MLIV is defined as a view type object that provides the access to a map object from a data source in GeoGrid. Each data source that provides maps contributes one or more MLIVs, where each MLIV can be used to generate a map. The following is an example of a MLIV which can generates a map as shown in Figure 4.7a. The MLIV provides the access of the following map. An example of the object structure of the MLIV is shown in Figure 4.7b. Note that the data source providing this MLIV needs to register with GeoGrid the following information shown in Table 4.1. To simply the example we have limited the code in Figure 4.7b to show only how the data is accessed and have ignored the code required to restrict the access to one user at a time. Figure 4.7c is another example of MLIV which stores geographic data in a vector data model.

Figure 4.7a: An example of data accessible by a specific MLIV

```
public class MapLiv {

   public    jpgMap getMap (int height, int width, double dpi, double minlat, double minlong,

   double maxlat, double maxlong) {

        string endpoint = "http:// rasterImageServer /arcgis/services/satelliteMaps/MapsServer ";
         ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy mapserver =
         new ESRI.ArcGIS.ADF.ArcGISServer.MapServerProxy(endpoint);

         MapServerInfo mapinfo = mapserver.GetServerInfo(mapserver.GetDefaultMapName());
         MapDescription mapdesc = mapinfo.DefaultMapDescription;

         ImageType imgtype = new ImageType();
         imgtype.ImageFormat = esriImageFormat.esriImageJPG;
         imgtype.ImageReturnType = esriImageReturnType.esriImageReturnURL;

         ImageDisplay imgdisp = new ImageDisplay();
         imgdisp.ImageHeight = height;
         imgdisp.ImageWidth = width;
         imgdisp.ImageDPI = dpi;

         ImageDescription imgdesc = new ImageDescription();
         imgdesc.ImageDisplay = imgdisp;
         imgdesc.ImageType = imgtype;
         MapImage mapimg = mapserver.ExportMapImage(mapdesc, imgdesc);
         jpgImage = clip(mapimg, minlat, minlong, maxlat, maxlong);

         return jpgImage;
        }
}
```

Figure 4.7b An example illustrates an MLIV that provides access to ArcGIS.

```
package mapLIV;
import java.sql.*;
import java.io.*;
import oracle.spatial.geometry.*;
import java.lang.Object;

public class QuerySpatialdb {
    public geoMap derive (double minlat, double minlon, double maxlat, double maxlon) {
        private GeoMap theMap = new GeoMap();

            Class.forName("com.oracle.jdbc.Driver");
            String url = "jdbc:oracle://spatiallocalhost/geogrid";
            Connection connection = DriverManager.getConnection(url);

            Statement stmt = connection.createStatement();

            String query = "";
            query = "select theGeometry from defaultTable where sdo_filter(theGeometry, ";
            query = query + "SDO_geometry(2003, 8307, null,SDO_elem_info_array(1, 1003,3),";
            query = query + " SDO_ordinate_array(minlon, minlat,maxlon,maxlat) = \"TRUE\"";
            ResultSet rs = stmt.executeQuery(query);


            STRUCT st = (oracle.sql.STRUCT) rs.getObject(1);
            //convert STRUCT into geometry
            JGeometry j_geom = JGeometry.load(st);
            theMap.addOneGeometry(j_geom);
        return theMap;
    }
}
```

Figure 4.7c: An example illustrates an MLIV that provides access to Oracle spatial.

Table 4.1. Registration data for the MLIV shown in Figure 4.7.

OwnerID	**Bounding Box**	**Theme**	**Geographic Quality**					
			Complete-ness	**Map type**	**Positional Accuracy**	**Reliability**	**Resolu-tion**	**accessibility**
MLIV-55	”42.0597047,-94.165246”, “42.210095, -93.8802273”	Satellite	1	**JPG**	**0.01**	**0.9**	**25**	**5**

4.3.2 Search MLIVs

The map mediation process starts with the spatial mediator determining the MLIVs that are capable of responding to all or part of the incoming request. To do this, the map mediator makes use of the MLIV registration data. The Fact Database ***FD*** is used to identify the MLIVs that satisfy theme and property requirements in the map request. Location requirements can take either symbolic (e.g., a city name) value or radius/point/bounding box values. Location requirements based on geographic coordinates (e.g. a point/point and radius/bounding boxes) are resolved using the R-Tree (***RT***). For symbolic terms that identify location (e.g. “Midwest”), the Geographic Symbolic Ontology (***GSO***) is used. Any terms that indicate locality point to the MLIVs that include the term value and its surrounding area. For example, terms like USA, North Central, Iowa, Story and Ames define locality, while terms like country, state, region, county and city are terms that assist the search. A fragment of the ***GSO*** is illustrated in Figure 4.8.

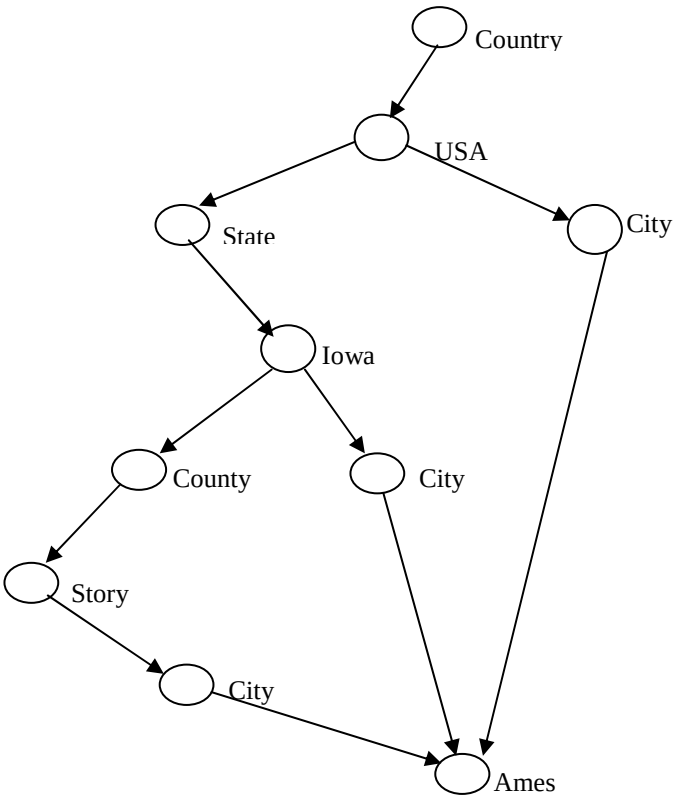


Figure 4.8: A fragment of the symbolic search.

If the request only provides a symbolic term then the mediator first uses Geographic Symbolic Ontology (***GSO)*** to locate the MLIVs that match the search criteria. The mediator then searches the Fact Database (***FD***) to find the bounding boxes of the located MLIVs.

4.3.3 Rank MLIVs

Once the MLIVs that satisfy the Fact Database search and the ***RT*** and/or ***GSO*** are gathered into a list, it is necessary to rank the MLIVs on the basis of their value responding to the request.

The spatial mediator partitions the MLIVs into two lists: *FullCoverageList* and *PartialCoverageList*. The *FullCoverageList* is defined as the set of MLIVs whose bounding boxes fully cover the requested area of the request. The *PartialCoverageList* is defined as the set of MLIVs whose bounding boxes overlap part of the requested area of the request. The motivation for the two lists is to allow our algorithms to first examine the quality of MLIV on the *FullCoverageList* (if they exist) and only go to the process of using MLIVs from the *PartialCoverageList* when they are required. Figure 4.9 shows several examples from these two lists. The request bounding box is indicated by the solid shaded box and the diagonally shaded box represents the MLIV bounding box. Examples A and B are from *FullCoverageList* and examples C and D are from *PartialCoverageList.* Example A shows the bounding box of the MLIV is bigger than the one of request and so it is a complete cover of the bounding box of the request. Example B is the case where bounding box of MLIV is exactly the same as the bounding box of the request. Example C and D are cases where the bounding box of a MLIV overlaps part of bounding box of the request.

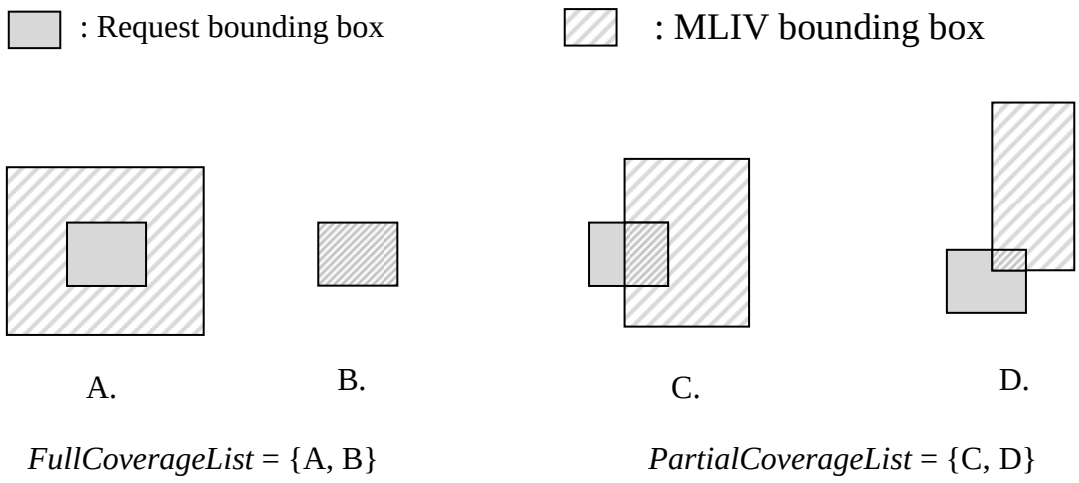


Figure 4.9. Examples of *FullCoverageList* and *PartialCoverageList*.

To investigate the two lists of MLIVs requires the mediator to evaluate the potential contribution of each MLIV to the generation of a useful map. The mediator makes use of a ranking mechanism to rank these potential MLIVs based on their likelihood of generating high quality spatial object in respond to the request. MLIVs on these two lists on are ranked on decreasing order of their ranking values. We put lots of effort to investigating ranking methodologies. The ranking mechanisms tested are described in more detail in Section 4.3.5

4.3.4 Map grouping

Before we describe our *map grouping algorithm* we first introduce a set of definitions that provide the basic concepts needed in the algorithm.

Definition 4.1:

A *grouping* is recursively defined as consisting of a tool type and a list of objects such that each object is either a MLIV or a grouping.

Definition 4.2:

A *well-formed grouping* is a grouping that only consists of tool types and MLIVs.

Definition 4.3:

A *map grouping* is a well-formed grouping that generates an instance of the requested map.

Definition 4.4:

The *ranking value of a grouping* is set by using the value of the smallest ranking value of any MLIV found in the grouping.

The task of the mediator is to generate at least one map grouping that has a ranking value greater than or equal to the system threshold. In our current model the system threshold is set when the mediator is initially installed. The *map-grouping algorithm* starts by examining the *FullCoverageList* in order of decreasing ranking values. If an MLIV in this list has a quality measure above the system threshold, it is formulated as a map grouping and passed to the next level.

If no complete cover MLIVs can generate a map grouping with a ranking values above the threshold, the algorithm switches to the *PartialCoverageList*. As in the previous case, the MLIVs are processed in order of decreasing quality measures. The algorithm maintains a collection of bounding boxes that represents the uncovered portions of the request map bounding box. We use an example to show how the *map grouping algorithm* works (Figure 4.10). Assume that MLIV α and MLIV β are the first and second top ranked MLIVs on the *PartialCoverageList* used, respectively. The algorithm first uses the bounding box of the MLIV α against the request bounding box and partitions the request bounding box into a collection of three fragments {1,2,R1} of the original request bounding box (Figure 4.10a). The algorithm then uses MLIV β to partition remaining bounding box R1 into bounding box 3 and 4. The collection of bounding boxes that remain uncovered contains four fragments (1,2,3,4) of the original request bounding box. Note the algorithm discards any MLIVs that come before MLIV β on *PartialCoverageList* but don't overlap with any fragments {1,2,R1}. Figure 4.10b shows the example after two MLIVs (α,β) have been processed.

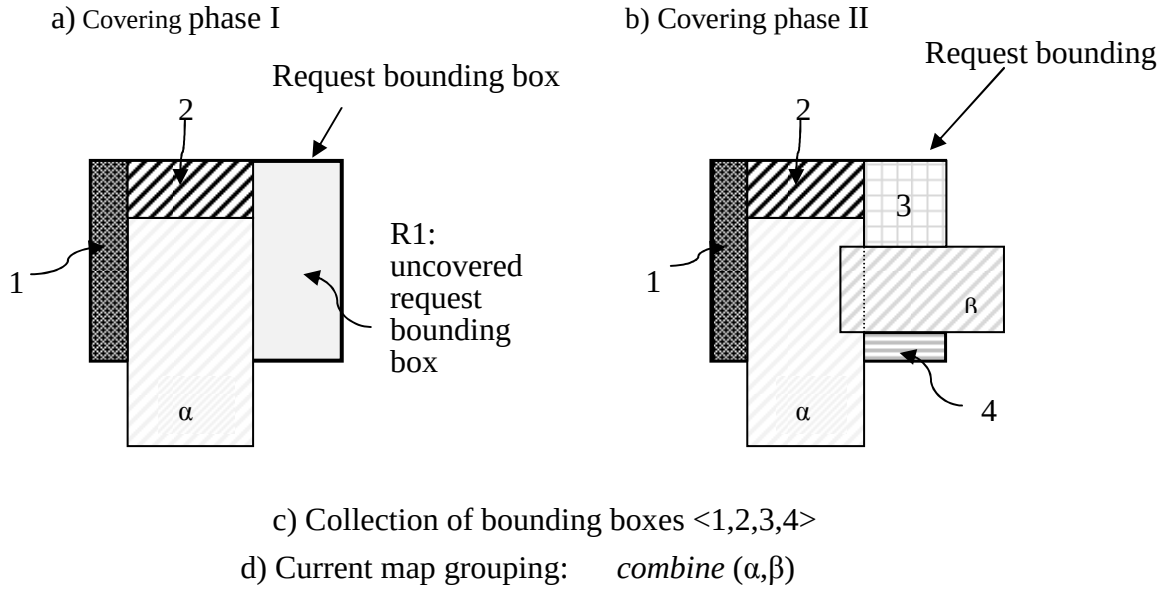


Figure 4.10. An example of the data structures after two MLIVs (α,β) have been processed.

The map grouping *combine* (α, β) (Figure 4.10d) formed after processing α and β indicates that a tool of tool type *combine* will be necessary to generate the map area covered by α and β.

The algorithm will continue to process the list of bounding boxes (1,2,3,4) (Figure 4.10c) in the example that have not been covered by either α or β. The algorithm continues until either the collection of uncovered bounding boxes (Figure 4.10c) is empty or the remaining MLIVs in the list have quality measure values below the acceptable threshold level. If the collection is empty, it means the request bounding box can be covered by the combination of areas of bounding boxes of the MLIVs in the current map grouping. The map grouping is returned. Otherwise, an exception indicating failure to find an appropriate map grouping is raised and a

message is sent back to the user that the request as written cannot be processed. Figure 4.11 shows the continuing process from the previous example shown in Figure 4.10.

Assume MLIVγ, MLIVδ and MLIVε are ranked after MLIVα and MLIVβ on the *PartialCoverageList*, respectively. The algorithm tests the bounding box of MLIVγ against the request bounding box. Since the bounding box of MLIVγ covers the bounding box 1 the collection of uncovered bounding boxes becomes {2,3,4} (shown in covering phase *a*). In covering phase *b* the algorithm then uses MLIVδ to cover the bounding box 4 and the set of uncovered bounding boxes becomes {2,3}. In the covering phase c, the algorithm uses MLIVε to cover bounding boxes 2 and 3 and leave the set of bounding boxes empty. This is one of the halting conditions of the algorithm and the algorithm stops. We present the *map grouping algorithm* in Figure 4.12.

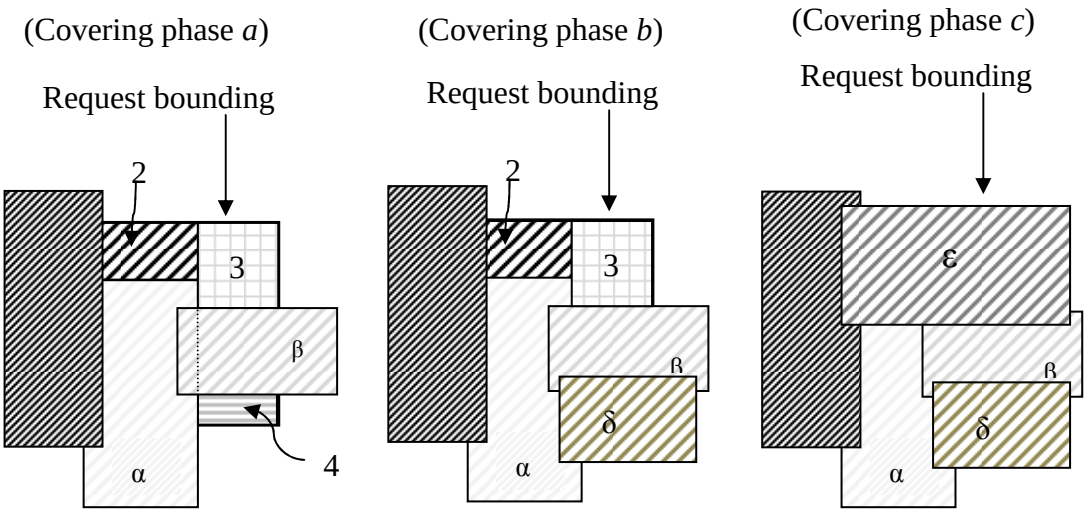


Figure 4.11. A continuing example from Figure 4.10.

```
Input:
        1.      Request bounding box:    BBX_R
        2.      The fullCoverageList:    List_C
        3.      The partialCoverageList:    List_PC
Output:     A map grouping: MG
{
   // The MLIVs on both list are arranged according to their ranking values.    The first MLIV has the highest ranking
   //value and the last MLIV has the lowest ranking value.

  process_done = false;
  while ( List_C.notEmpty() and process_done = false)
     {
            MapG ← (List_C.firstElement());              // a map grouping is formed
            List_C.removeFirst();                        // remove the first MLIV from the list
            MapG.rankingValue = MLIV.rankingValue;
            if MapG.rankingValue >= system.threshold
                    then process_done = true;
      }

   // Array_BBX is a collection of BBXs that are not covered by any MLIVs. Inclusion of bounding box of MLIV breaks
   //the original BBX_R into several smaller bounding boxes which are put into the Array_BBX. The request bounding
   //box is put into the array for the first round.

   Array_BBX ← BBX_R;
   While (List_PC.notEmpty() and process_done = false)
        {
           MLIV =    List_PC.firstElement();     //always takes the first MLIV on the list
           BBX_MLIV = MLIV.boundingBox;         //obtain the bounding box of MLIV
           found = false;
           i = 0;
           while ((found = false) and ( i < Array_BBX.size()))
                {
                    A_BBX = Array_BBX.elementAt(i); // check the element in the array
                    If (BBX_MLIV partially cover A_BBX ) then
                        {
                                MapG ← MLIV;    // add MLIV into map grouping
                                List_PC.removeFirst();      //remove the MLIV from the list
                                Array_BBX.removeElementAt(i); //remove the bbx from array
                                Array_BBX ← break A_BBX by BBX_MLIV; //add smaller bbxs
                                found = true;
                        }
                     else    // this MLIV might cover more than one bounding box in the array
                        {
                                Array_BBX.removeElementAt(i); //remove the bbx from array
                                i = i + 1;    // evaluate the next uncovered bbx in the array
                                if (Array_BBX.empty())
                                      {
                                          MapG ← MLIV;
                                          List_PC.removeFirst(); //remove the MLIV from the list
                                      }
                        }
                }

           If (List_PC.empty() or Array_BBX .empty()) then
                    Process_done = true;
        }

      If (Array_BBX.notEmpty() or MapG.rankingValue < system.threshold) then
              Error message = "Fail to find any MLIV that covers request bounding box";
      else    output MapG;              //process is done successfully.
}
```

Figure 4.12. Pseudo code for map grouping algorithm.

4.3.5 Map Script Generation

Once an acceptable map grouping has been generated, it needs to be converted into syntax capable of guiding the computation server. The tool types must be replaced with actual tool names available in the GeoGrid infrastructure. It is also useful to use an optimization step to improve tool usage in the computation server. We call the converted form, the *template skeleton* of the integration script. The definitions of template skeleton and resulting *map script* are given as follows:

Definition 4.5:

A *template skeleton* is recursively defined as consisting of a tool name and the parameters required by the tool, where each parameter is a string that identifies an MLIV or a *template skeleton*.

Definition 4.6:

The *map script* is the *template skeleton* replaced the tool name and parameter with the specific format and actual data. A *map script* has the form:

map(O_α(□,□ O_β(□,□,□,□ where

$\mathbf{S}_n$ is in the form of: (MLIV_location, MLIV_query, MLIV_boundingBox)

$\mathbf{O}_m$ is either in form of: (tool_name, tool_location,

tool_category) or NoOp

Where NoOp indicates no tool is included

The MLIV tokens in the *template skeleton* are replaced with the viable queries for the MLIVs and any associated information required by the computation server to gain access to the data wrapper supporting the individual MLIVs. The MLIV could either be actual queries or vectors of query parameters.

To deal with the large variety of geographic data sources (e.g., Oracle spatial, Arc View, etc.) that exist today, we have found it more practical that each MLIV accepts a vector of values that the MLIV substitutes into the appropriate query language to form the query that the MLIV executes against the local geographic data source(Figure 4.7b). The detail of the vector format are determined during the MLIV registration and made available to the spatial mediator through the Fact Database.

The associated information mentioned above depends to some extend on the type of communications protocol being used to connect the computation server wrapper to the data source wrappers. In general we have found two types of information to be valuable, namely, the address of the data source site (e.g, IP address, url), and the layout of the spatial object(s) generated by the data source.

Once the queries and associated information have been generated and used to replace the MLIV tokens, the resulting map script is passed to the computation server for processing. To allow construction of a new map script in case of failure, the mediator stores the request, the MLIV lists, map grouping, template skeleton, and map script in temporary storage until the map has been created and returned to the user application.

Example 6: We use an example to illustrate the map mediation process. Assumes that the bounding box of an incoming request has the following latitude/longitude coordinates: (“41.86301,-94.165246","42.210095, -93.698127”).

The first set of coordinates is the lower left corner of the bounding box and the second one indicates the upper right corner. To give a clear demonstration of the map grouping process, we further assume the spatial mediator search R_Tree and locates four MLIVs that each partly overlap with bounding box of the request. *FullCoverageList* is empty and the mediator switch to *PartialCoverageList*. Four MLIVs on the *PartialCoverageList* are shown in the Figure 4.13. Assume the ranking values of MLIVs are ordered in accordance with their subscriptions, that is, $MLIV_1$ is ranked higher than $MLIV_2$ and $MLIV_2$ ranked higher than $MLIV_3$, etc. The spatial mediator generates the following *map grouping* based on the *map grouping algorithm*.

Map grouping:

combine(*combine*($MLIV_1$,$MLIV_2$), *combine*($MLIV_3$, $MLIV_4$))

The *map grouping* is then converted into *template skeleton* and then *map script* shown below.

Template skeleton:

clip(mosaic($MLIV_1$, $MLIV_2$, $MLIV_3$,$MLIV_4$), “41.863010, -94.165246” ,”42.210095, -93.698127”)

The four MLIVs are combined into one map by the mosaic tool in the computation server and then clipped to fit the indicated bounding box (“41.863010, -94.165246”, ”42.210095, -93.698127”). Replacing the MLIV tokens ($MLIV_1$, $MLIV_2$, $MLIV_3$, $MLIV_4$) with location and query vector information generates the following map script.

Map script:

map(<clip, IP of clipTool, cropTypeTool>(<mosaic, IP of mosaicTool, combineTypeTool>(IP of $MLIV_1$, <” MapServer”, “4,4,72” >, (”42.0597047,-94.165246”, “42.210095, -93.8802273”)>),(IP of $MLIV_{2,}$ <” MapServer”, “4,4,72” >, (”42.0597047, -93.8802273”, ”42.210095, -93.698127”)),(IP of $MLIV_{3,}$, <” MapServer”, “4,4,72” >, (” 41.863010, -93.8802273”, “42.0597047, -93.698127)), (IP of $MLIV_{4,}$, <” MapServer”, “4,4,72” >, (” 41.863010, -94.165246”, “42.0597047, -93.8802273))))

$MLIV_1$

$MLIV_2$

$MLIV_4$

$MLIV_3$

Figure 4.13. Maps from the MLIVs that have been clipped to the part of requesting bounding box that they cover.

The map fragments shown in Figure 4.13 illustrate the results of the queries generated the map for the four MLIV defined in Example 6.

The map displayed in the Figure 4.14 is the map results in the execution of the map script (from Example 6) in the computation server, which in turn will be sent to the user application in response to the request. The accuracy of the final map will depend on the quality of tools that are available within GeoGrid.

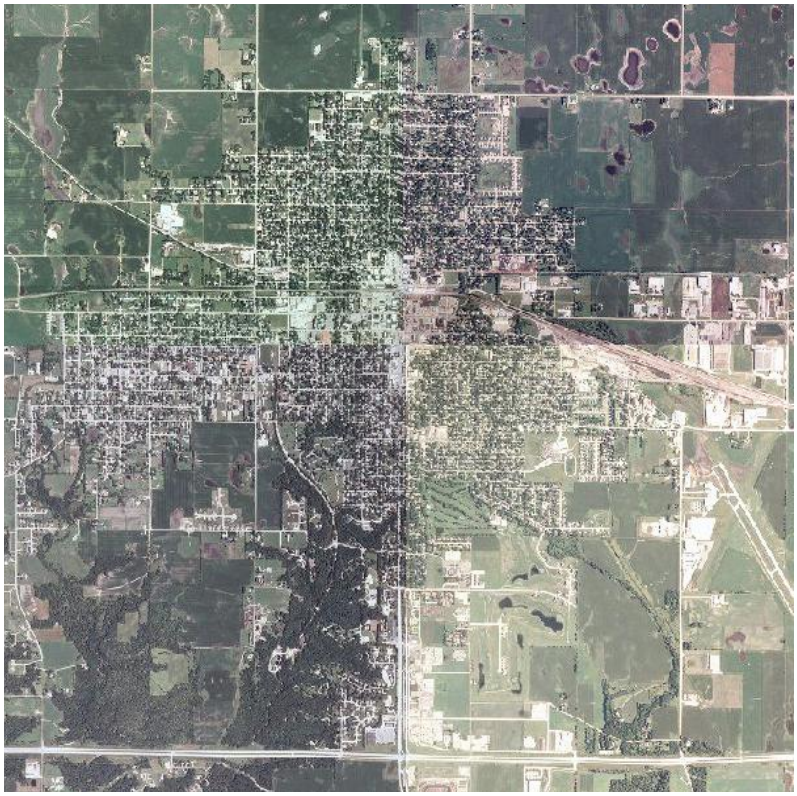


Figure 4.14. Result map after integration.

4.3.6 Map MLIV ranking strategy

In this section, we present the ranking strategy used to support the generation of the *map script* in the spatial mediator. We first describe our motive to employ a ranking mechanism into the spatial mediator followed by the definitions of quality attributes used in the ranking mechanism and then introduce the mechanisms themselves.

The popularity of geographic information systems results in the availability of a large number of geospatial data sources with different types of data of varying qualities. Ranking of data providers plays an even more important role in the multiple

data sources environment than before. Generally, frameworks in the information integration environment have not addressed explicitly the ranking issues. However, we argue the ranking will become mandatory as well as locating relevant data when it is necessary to perform integration of information from multiple data sources. One of our goals in GeoGrid is to look into all aspects of data integration including ranking. We propose several approaches to address ranking in the context of geographic data integration.

One of the challenges of ranking is to push ranking computation into the pre-query processing phase to make it efficient and ease the overall operation. We apply the ranking metric before querying every individual data source and generating the *map script*. We believe these ranking approaches will result in the minimal computation cost in the overall integration process.

The most effective ranking approach is to make use of characteristics of data sources. This information is available in the Fact Database and Rule Set in the form of metadata and rules, respectively. The ultimate goal of the spatial mediator is to provide a high quality map in respond to the map request. This motivates our selection of quality based parameters in the ranking and scoring metrics discussed in the Section 4.3.6.2 and Section 4.3.6.3, respectively. The consideration behind the selection is based on geographic data quality. There are six attributes associated

with our ranking mechanism, namely *completeness, file, positional accuracy, reliability, resolution* and *accessibility*. We introduce them in the next section.

The values of each attribute of an individual MLIV will be loaded into our Fact Databases when the MLIV is registered in GeoGrid. To determine the corresponding parameter value in the quality measure, the spatial mediator makes use of this value from the Facts Databases ***FD*** and the corresponding attribute value in the request to fire rules in the Rule Set ***RS***. More detail on how this is done is given in Subsection 4.3.6.2.

4.3.6.1 Quality attributes of geographic data

While participants of the geographic community agree on the importance of spatial data quality, their definitions of quality varies greatly. Many efforts are made towards gaining a consensus on a single definition in the past years. Devillers et al.(2007) present the concept of spatial data quality as “the closeness of the agreement between data characteristics and the explicit and/or implicit needs of a user for a given application in a given area.” (Devillers et al. 2007, p.264).

4.3.6.1.1 Geographic data quality standard

Attributes of data quality recommended by ISO and well recognized by the GIS community are commonly identified as the “famous five”: completeness, logical

consistency, positional accuracy, temporal accuracy and thematic accuracy. Among them, logical accuracy refers to all logical rules that govern the structures and attributes of geographic data. Based on our observation it is reasonable to assume all spatial objects provided by MLIVs follow the topological and geometric integrity (for example, the contour of a polygon is properly closed in the dataset). So, we don't consider the logical consistency as an attribute in our quality measure.

Thematic accuracy is another attribute not included in the measure. Thematic accuracy sometimes refers as the "attribute accuracy" and is defined as the accuracy of attributes and of the classification of features and their relationship (Devillers et al.2007). In our model, we have used "theme" as a filter criteria to locate the MLIVs. Therefore the thematic accuracy is not included in our quality criteria. Temporal accuracy is not included in our quality measure for the same reason.

4.3.6.1.2 Fitness for use

The concept of "fitness for use" proposed by Juran (Juran et al. 1974) has often recognized as a definition of quality in the largest sense and sometimes refers as the external quality. It corresponds to the level of concordance that exists between a product and user needs, or expectations (Devillers et al. 2007). Several researchers from the area of information management have adopted the concept of "fitness for use" and identified some attributes to define business data quality (Wang and Strong 1996,

Lee et al. 2002). Some similarity between these attributes and ones for the geographic data quality emerges after our further investigation. And yet there are some attributes not identified by geographic participants but worthy of being considered as criteria to evaluate the quality of geographic data in the mobile environment that GeoGrid is designed to operate in.

Among them are size of data, accessibility and reliability. The size by itself cannot represent the usefulness of a data. The size of data is a function of resolution and compression and region covered in the context of geographic data. For example, a map of very small size (that means its resolution is generally coarse) that display the soil type of a region and doesn't provide useful information when compared to a map with a bigger size (with a fine resolution) that conveyies more meaningful information is not a reasonable choice. We include accessibility and reliability in our ranking criteria.

4.3.6.1.3 Data conversion

Not only the original quality characteristic associated with maps but also the quality change due to a conversion process is considered in our quality measure. The common conversion process deals with the alteration of the underlying data model, for example, convert raster data to vector data. Changing the resolution is another type of conversion. When the request asks for the particular resolution or a

map type the spatial mediator needs to be able to consider the availability of tools and evaluate the potential quality loss due to a conversion of resolution or data model in order to rank the MLIV. Due to the consideration of possible conversion we include resolution and map type in our ranking criteria.

4.3.6.1.4 Completeness

Completeness is defined as the difference between an actual object and its specification in the document (CEN/TC287/WG02 1995). It is used to detect errors of omission (abnormal absence) or commission (abnormal presence) of features, their attributes and relationship (Devillers et al. 2007). To quantify completeness, three possible measures are suggested by (CEN/TC287/WG02 1995). We adopt the "coverage ratio" as the measure which is the percentage of data present relative to specification. For example, if there are 250 roads in a geographical area and 2 of them are missing, then the dataset is 99% covers the features that is, roads. Or if the map only includes 225 roads then the map is 90% complete. We observe that most geographic data providers use a text description to specify the *completeness* of their data. The text description of *completeness* report in the metadata of the geographic data can be quantified by computing the ratio between coverage presented and the actual area on the ground. This job is done by data source administrator. When data source registers a MLIV with GeoGrid it needs to provide a quantity value for the

completeness attribute of the registered MLIV. For example, the text “The following areas are missing, with no known data source: Essex County except for Newark” is used in the metadata of the State of New Jersey Composite of Parcels Data provided by New Jersey Office of Information Technology (NJOIT) and Office of Geographic Information Systems (OGIS) (JNOIT 2010). A numerical value for the attribute *completeness* should be assigned by the data source which provides this map. We adopt these definitions and define the completeness attribute associated with the ranking measure as follows:

Definition 4.7:

Completeness is a measurable coverage ratio between data content and its specification.

4.3.6.1.5 Positional accuracy

Positional accuracy sometimes refers as the “spatial accuracy” is a measurement of how close map features are to their true position on the Earth (Devillers et al. 2007). The common measures are horizontal error and vertical error. We adopt the definition of “Quantitative_Horizontal_Positional_Accuracy_Assessment” suggested by SDTS (Spatial Data Transfer Standard) and define positional accuracy as follows:

Definition 4.8:

Positional accuracy is the degree of the deviation between data content and its ground true position.

4.3.6.1.6 Accessibility

Accessibility is defined as "the extent to which data is available or easily and quickly retrievable" in (Pipino et al. 2002). The measurement of easy of data retrieval is beyond our scope of research. We define the *accessibility* as the time that the MLIV takes to make the generated map available for use plus the time it takes to download the data. It considers not only the data size but also the connection speed of data sources and is defined as follows:

Definition 4.9:

Accessibility is the measure of availability for the data in terms of time.

4.3.6.1.7 Reliability

The *reliability* is defined as "the extent to which data is available and regarded as true and credible" in (Devillers et al. 2007). In other words, it is defined as the level of confidence a data source has that the data is correct. We define the *reliability* as follows:

Definition 4.10:

Reliability is the confidence level of correctness and credibility of the data.

4.3.6.1.8 Resolution

Resolution refers to the "the small size of feature can be mapped or measured" (Burrough and McDonnell 1998). For example, if the size of each individual cell of an imagery type map is 30 meter x 30 meter then it is having a resolution of 30 m. We adopt this definition for the attribute *resolution* which is defined in the following paragraph.

Definition 4.11:

Resolution is defined as the size of the smallest recording unit of the map.

4.3.6.1.9 Map type

There are two types of logical structure for the maps considered in our work, namely raster data model and vector data model. A vector data model uses two-dimensional Cartesian (x,y) co-ordinates to store the shape of a spatial entity (Heywood 2006). In the vector model the point is the basic block from which other spatial features (line, polygon) are constructed. The raster data model is described as tessellations. Each individual cell is used as the building block for creating images of point, line, polygon (Heywood 2006). The attribute *mapType* is defined as follows:

Definition 4.12:

MapType is defined as the logical structure used to encode the geographic data.

4.3.6.2 Quality ranking measure model

The spatial mediator utilizes a quality measure to evaluate the potential contribution of each MLIV to the generation of a useful map. A MLIV ranking value v is defined as follows:

$v = w_1 * com + w_2 * file + w_3 * pos + w_4 * rel + w_5 * res + w_6 * access$, *where*

- *com* indicates the *completeness* of the geographic data,
- *file* estimates the cost in terms of quality of converting the MLIV data to the map type required by the requesting application,
- *pos* indicates the *positional accuracy* of the MLIV data,
- *rel* is the *reliability* of the data source supporting the MLIV,
- *res* indicates the degree to which the resolution of the map generated by the MLIV matches the requested resolution,
- *access* is an estimate of the *accessibility* of the MLIV data given, and the size of the data and the available connection speed and bandwidth, and
- w_i is the weight associated with *ith* parameter

The motivation for the ranking value comes from the work used to determine image similarity in image retrieval systems (Lim et al. 2001, Mountrakis et al. 2004). In particular the linear combination of the weighted terms is a common approach in such systems. Our contribution comes from ways in which we determine the parameters and the weights.

There are some questions that seem difficult to address using the traditional approaches like the ones in (Mountrakis 2004 & 2005, Lim 2001) to generate weights and parameter values. Their approaches require users' interaction with systems in deciding weight and parameter values. It is impossible for a large system like GeoGrid to use such an approach. To generate parameter values and weight values we propose several models which are described in Section 4.3.6.2.2 and Section 4.3.6.2.3, respectively.

Determination of the individual parameter value makes use of the Facts Database ***FD*** and/or rules from Rule Set ***RS*** . Models that generate their values are described in detail in the following sections.

The approach taken in the proposed system is to use the MLIV data to generate a set of rules for inclusion in the rule set. Each generated rule matches the request and MLIV values in the *if condition* and provides the value of the parameter in the *then clause*. Examples of the if/then rules generated for the *file* parameter are

shown in Figure 4.15(a). request.*mapType* in the sample rules shown in the Figure 4.15(a) represent the *mapType* attribute value associated with the request. MLIV.*mapType* in the sample rules stands for the *mapType* attribute value associated with the MLIV. In our implementation the rules are generated before the system is activated and remain static while the spatial mediator is running.

The motivation for using rules rather than functions to generate the individual values of the parameters comes from the fact that the complete process of determining the individual parameter values can require additional rules in the rule set (e.g., Figure 4.15 (b)). We found it more practical to expand the rule set rather than combine the use of rules and functions. The use of the rules also allowed us to simplify the run time requirements.

(a) Map Type Rules:

if request.*mapType* = vector and MLIV.*mapType* = raster
then *file* = 0.55
if request.*mapType* = raster and MLIV.*mapType* = vector
then *file* = 0.65
if request.*mapType* = MLIV.*mapType*
then *file* = 1.0

(b) Samples of rules for converting file types to map types:

if fileType = JPG
then *mapType* = raster
if fileType = GEOTIFF
then *mapType* = raster

Figure 4.15. Sample rules for dealing with file and map types.

In the next section, we look at parameter values for these rules. We start with the values of the attributes used in the *if clause* of the rules and then we present approaches to generate parameter values used in the *then clause* of the rules.

4.3.6.2.1 Attributes Values Generation Approaches

Due to the growing amount of geographic data available different values exist in the quality attributes of geographic data. After long investigating on the geographic data quality our research team found most commonly used values for the attribute which is in the *if clause* of the rules generating parameter values of the ranking metric in our Quality Ranking Measure Model. One exception is the attribute *reliability*. The value for attribute *reliability* is calculated by the spatial mediator. The attribute values are listed in Table 4.2 along with citations.

4.3.6.2.1.1 Attribute *reliability* value

In our model, the value of attribute *reliability* provided by the data source is used as the initial value and the mediator uses a moving average window method to calculate the attribute *reliability* value for the $MLIV_i$, $MLIV_i^{reliability}$, and is defined by the following function:

$$MLIV_i^{reliability} = (fs) / (rs)$$

where fs is the number of failed responses sent from computation server indicating the MLIV fails to respond

rs is the last n request made to MLIVi, the value, the value n is set through a configuration file of the system.

Based on the practical consideration, the values for attribute *reliability* for registered MLIVs in GeoGrid infrastructure is limited within 80% and 100%. A MLIV with a poorer reliability value than 80% is restricted from providing any maps unless the request specific indicates a willingness to use maps of lower reliability.

Table 4.2 Attribute values in the rules

Attribute	Commonly used values	Related citations
completeness	1, 0.9, 0.8	(ISO 2002),(JNOIT 2010)
mapType	SHAPE, VPE, DLG, DEM, GEOTIFF, TIFF,JPG	(Clarke 2001), (Burrough 1998), (Heywood et al.2006)
Positional Accuracy	0.01, 0.02, 0.03	(NSSDA 1998),(SEDAC 2008)
Reliability	0.8, 0.9, 1.0	Decided by the spatial mediator
Resolution	1 m, 5 m, 10 m, 25 m, 45m	(UNBC GIS 2006),(Davis 2001),(Heywood et al.2006)
Accessibility	5 sec, 10 sec, 25 sec, 45 sec	(Moussaoui 2006)

4.3.6.2.2 Parameter Values Generation Approaches

The values for parameters used in the quality metric introduced in Section 4.3.5.2 are generated by use of Fact Databases ***FD*** and rules in the Rule Set ***RS***. The spatial mediator uses attribute values of incoming requests and of MLIVs from ***FD*** to generate a set of rules for inclusion in the ***RS***. Each generated rule matches the

request and MLIV values in the *if* condition and provides the value of the parameter in the *then* clause.

The approach proposed in (Mountrakis 2004) requires users to input their preference percentage on the individual dimension of their aggregation function. As we mentioned in the previous section, since our GeoGrid is such a large and dynamic system it is not a possible way to ask users to input their preference on each data object in our infrastructure. The models we propose require no users' interaction with the system. Our focus is to identify the parameter values based on their geographic interpretation and the goals of the application designs. We must point out difficulties exist in generating the parameter values due to the lack of an appropriate space to compute a numerical values for some of these parameters. We develop a model, Parameter Resolution Value Generation Model, to generate the parameter *res* because of its geographic characteristics. For the rest of the parameters namely, *com, file, pos, rel* and *access*, the values used in our testing have been determined by our team members based on our knowledge of geographic data and values available in the literature (Table 4.2). This approach is elaborated in the Expert Model section.

The generation of these parameter values is based on the geographic interpretation. The higher the value indicates a better geographic quality than the

one with a lower value with respect to a parameter. The example shown in the Figure 4.16 indicates the MLIV with a vector type will receive a reasonable high value for the parameter *file* (which is 0.65 in the example) when a request asks for a raster file type. The reason is that a conversion from vector to a raster will maintain a reasonable geographic quality. While the MLIV with a raster type will generate a lower value for the parameter *file* (which is 0.55 in the above example) when the request is asking a vector type indicates that the map will have a poor quality after a conversion from a raster to a vector type. The approach to generate values for the parameter *file*, i.e. 0.65 and 0.55 in this case, is presented in Section 4.3.6.2.2.2.1. The highest value for a parameter is 1 indicating the underlying MLIV generates a map that meets or exceeds the incoming request's requirement with respect to that parameter, in another words, this MLIV generates the best map with respect to that parameter. For example, a MLIV with a JPG file type will receive the value 1 for the parameter *file* if the incoming request asks for a map of JPG type.

4.3.6.2.2.1 Expert Model

There are some differences between integrating a disparate set of geographic data sources and the integration of traditional SQL_based databases. One of them is that the integration of geographic data requires more human participation. We consider that experts familiar with geographic data quality issues should get involved

in the process of data integration. Similar scenarios have already been identified in various contexts (Devillers 2007, Gervais et al. 2007, Combra 2009). Since our research team has conducted a long term investigation with deep exploration on geographic data quality we put forward some values for the parameters in the ranking metric used in our testing. In the expert model, the values of parameters *com, file, pos, rel, access* are identified by our team members and some GIS professionals. The available literature was an important source of parameter values as well (Table 4.2). We use the following section to demonstrate the rational behind the decision on value for the parameter *file*.

4.3.6.2.2.1.1 Data Model Conversion

Raster and vector are the two basic data structures for storing and manipulating geographic data on a computer. Raster model uses the grid form to store data. Each pixel or cell contains either a data value for an attribute, or a reference number pointing to an attribute in the database (Clark 2001). Because a raster image map has to have cells for all spatial locations, it is strictly limited by how big a spatial area it can represent. Vector data is represented as a collection of simple geometric objects such as points, lines, polygons, etc. All of the major GIS available today are primarily based on one of the two structures, either raster based or vector based.

Raster format data are often output from optical scanner or other raster imaging devices. Vector data acquisition is often more difficult than raster image acquisition, because of its abstract data structure, topology between objects and attributes associated (Heywood et al. 2006). It is possible to perform the raster-to-vector or a vector-to-raster conversion. And it is clear that going from vector to raster, filling in grid cells as lines cross them or as polygons include them, is relatively simple. The opposite is quite complex (Clark 2001). Although recent development in automated conversion technology has make this conversion in a matter of minutes or even seconds, the quality loss due to the conversion is unavoidable.

The following figures show the results of conversion. The tools being used to convert are Vextractor (http://www.vextrasoft.com/vextractor.htm), R2V (http://www.ablesw.com/r2v/) and ArcView (http://www.esri.com/software/arcview/index.html).

4.3.6.2.2.1.1.1 Raster to vector conversion

Figure 4.16 shows the conversion from the JPG file type to the SHAPE file type. The SHAPE file type is a popular geospatial vector data format. It is developed and regulated by ESRI (Environmental Systems Research Institute).

The original file is a colored relief map of County of Boulder, Colorado and is in the JPG file type (Figure 4.16a).The 3-dimentional representation of terrain is displayed by the shades of color. There are blue lines and polygons to depict as rivers. There is some obvious loss after the conversion. Not only the color is lost, but also some features (lines and polygons) are missing in the converted file (Figure 4.16b). Same scenario exists in the Figure 4.17a, 4.17b.

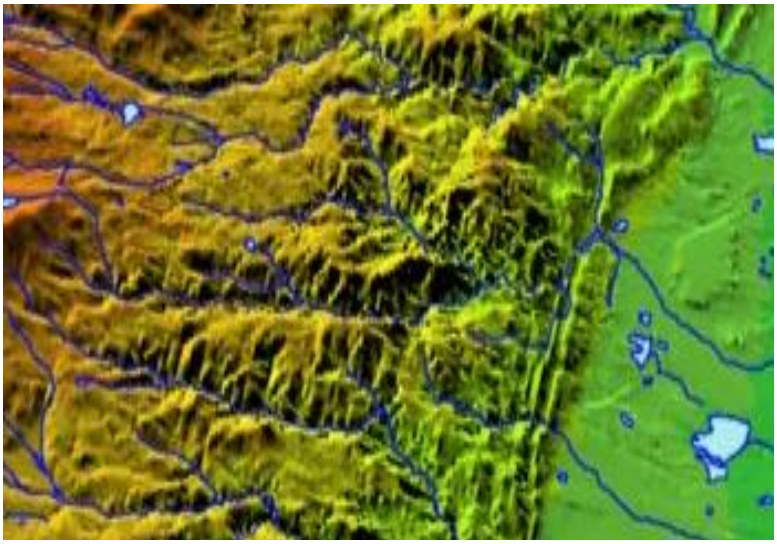


Figure 4.16a. The original raster (JPG type) map.
(Source: http://www.bouldercounty.org/lu/gis/images/reliefsd_bc.jpg)

Figure 4.16b. The converted vector (SHAPE type) map.

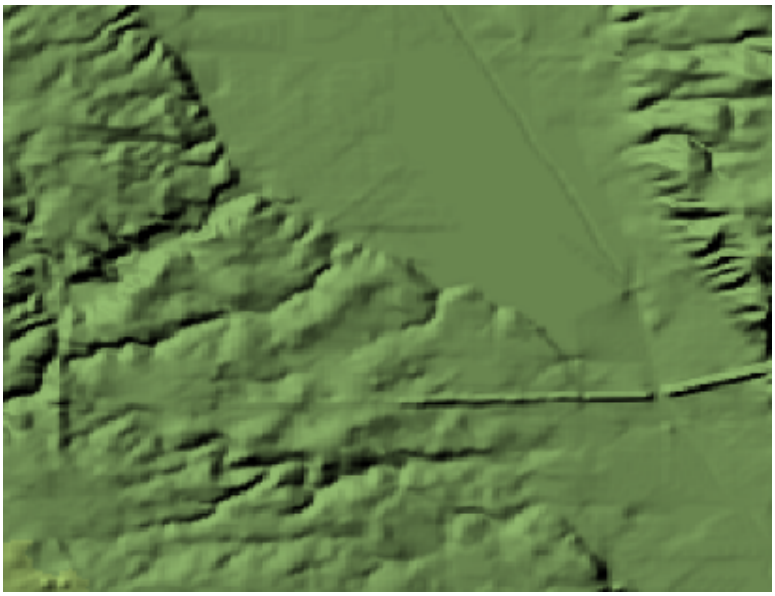


Figure 4.17a. The original raster (TIFF type) map.

(Source: http://ortho.gis.iastate.edu/)

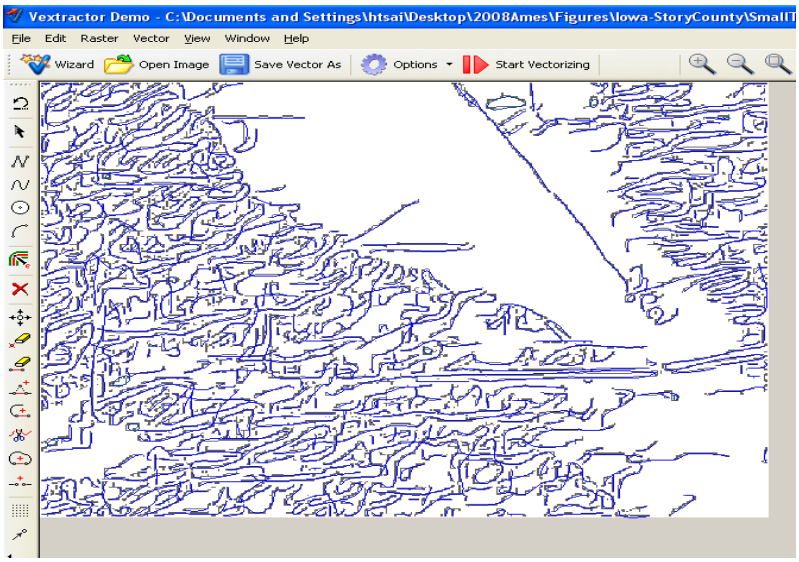


Figure 4.17b. The converted vector (SHAPE type) map.

In our next conversion example, an image with a JPG file type illustrating some portion of the proposed expansion in the Syracuse Metropolitan Area (Figure 4.18a). The diagonal line shaded area represents the area after the expansion. Names of counties are shown in this image. The shaded area is easy to identify after the

conversion into the SHAPE file type (Figure 4.18b), but counties names are distorted and hard to recognize.

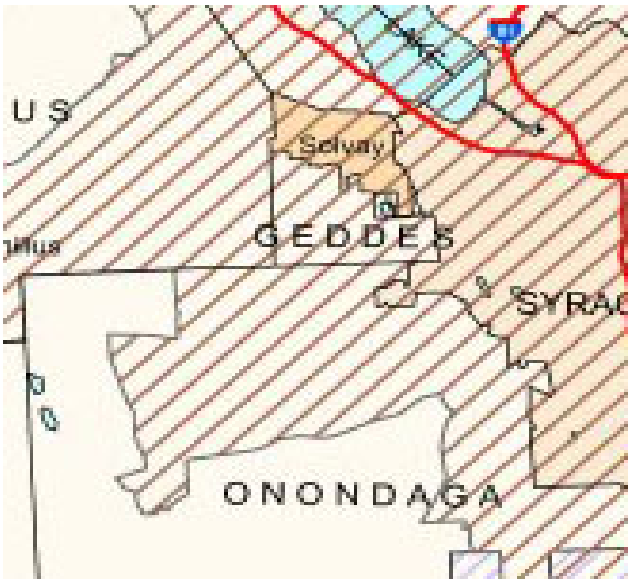


Figure 4.18a. The original raster (JPG type) map.
(Source: http://www.smtcmpo.org/docs/maps/smtcmpa.jpg)

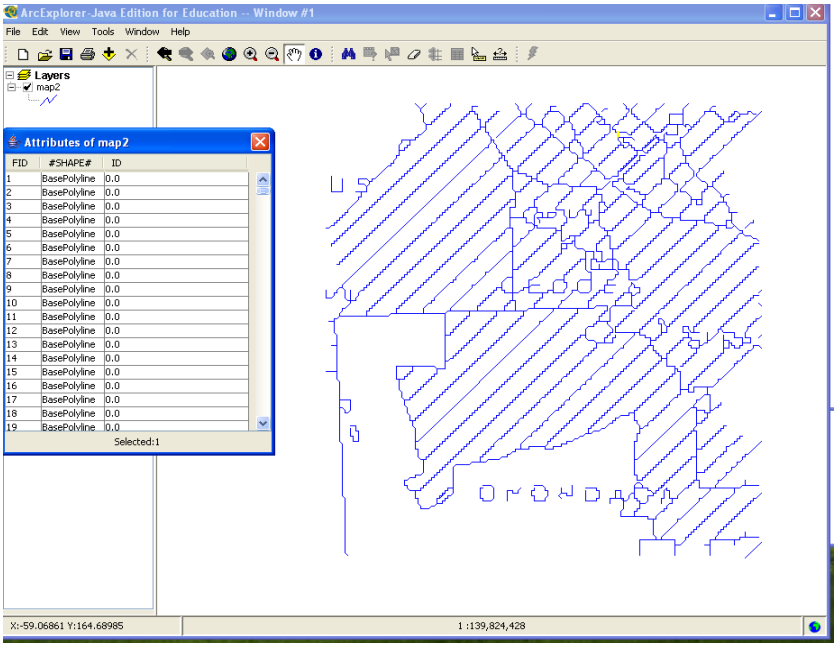


Figure 4.18b. The converted vector (SHAPE type) map.

4.3.6.2.2.1.1.2 Vector to raster conversion

In this example, a map with SHAPE file type which is the county of State Iowa (Figure 4.19a) is converted into a raster file type by the use of GIS tool ArcView.

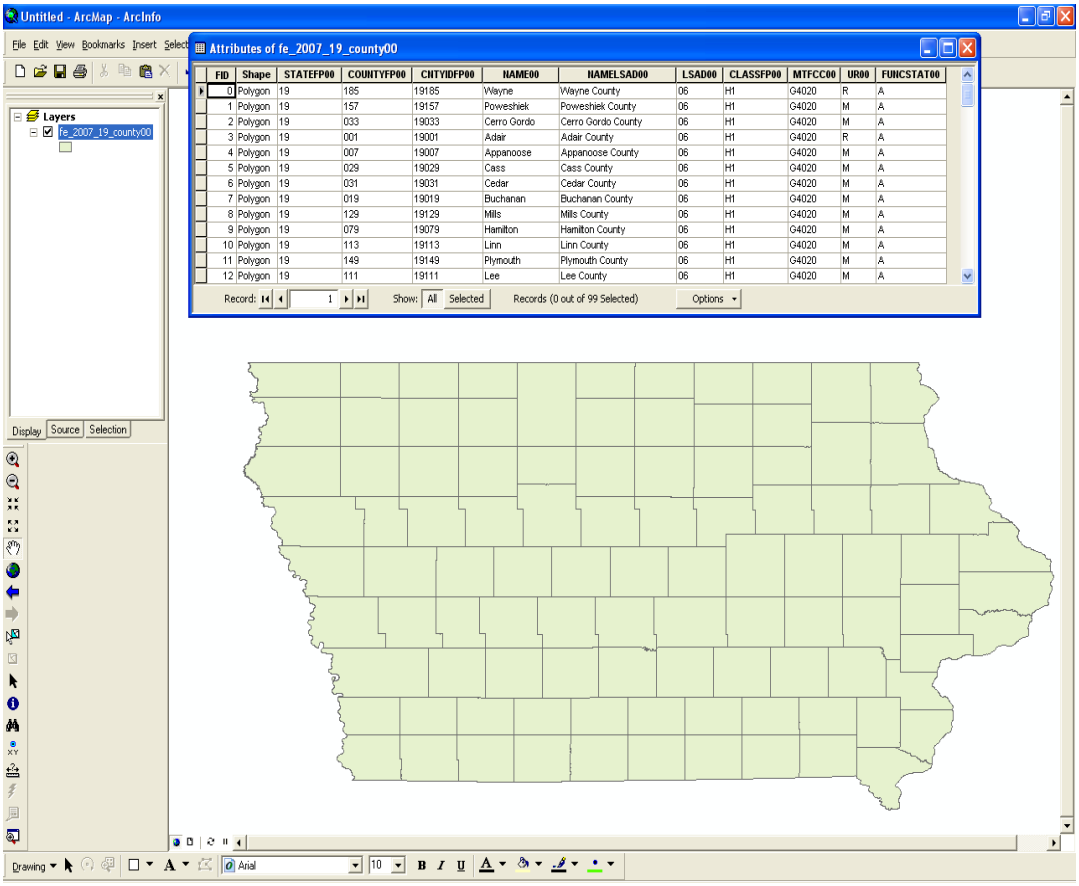


Figure 4.19a: the original vector (SHAPE type) map

(Source: http://ortho.gis.iastate.edu/)

This SHAPE file is converted into JPG and TIFF shown in Figure 4.19b and Figure 4.19c, respectively. When applications ask for a raster file type, they expect to obtain some information when they look at the image type map like the one shown in the Figure 4.19d. While the vector file type only stores geographic features and

other information (like the one in the table shown in Figure 4.19a) the raster file carries along the display information with geographic data, like color, legend or meaningful tags or names on the image itself. In another words, the display information of the raster file type conveys some information. It is obvious that some information is lost due to the conversion when we compare Figure 4.19a to the original raster map in Figure 4.19d.

Figures 4.19b and 4.19c, respectively, show that converting the vector map shown in Figure 4.19a back to JPG and TIFF con not recover the lost information.

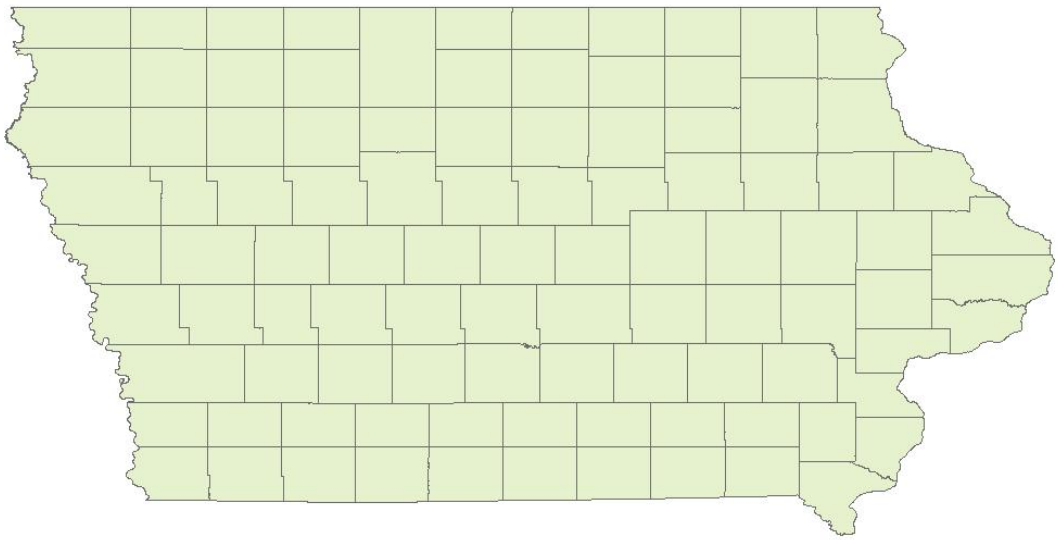


Figure 4.19b: the converted raster (JPG type) map.

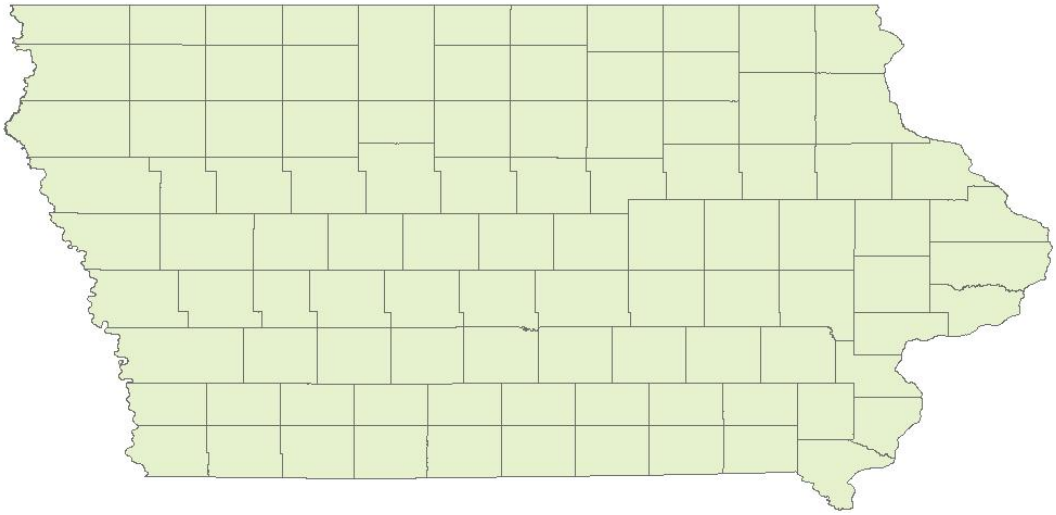


Figure 4.19c: the converted raster (TIFF type) map.

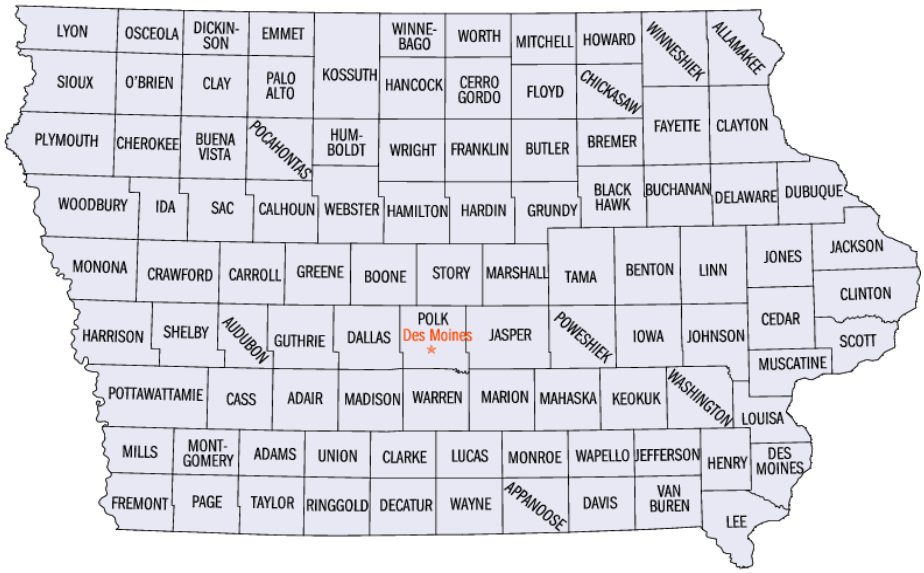


Figure 4.19d: The original map in raster type.

4.3.6.2.2.1.1.3 Parameter *file* value

Based on the results of the conversion, The parameter *file* value is set as follows:

if request.*mapType* = vector and MLIV.*mapType* = raster

then *file* = *0.55*

if request.*mapType* = raster and MLIV.*mapType* = vector

then *file = 0.65*

if request.*mapType* = MLIV.*mapType*

then *file = 1.0*

4.3.6.2.2.1.1.4 Parameter *com* value

The value of the parameter *com* indicates the likelihood of the underlying MLIV will generate a map whose *completeness* attribute matches or exceeds the one with incoming request. The values are determined by our team members. Following are some examples of the rules.

(a) if request.*completeness* = 0.9 and MLIV.*completeness* = 1.0

then *com = 1*

(b) if request.*completeness* = 0.9 and MLIV.*completeness* = 0.9

then *com = 0.9*

(c) if request.*completeness* = 0.9 and MLIV.*completeness* = 0.8

then *com = 0.5*

Case (a) indicates the MLIV has a map whose completeness attribute exceeds the need of the incoming request and thus receives a value 1 for the parameter *com*. Case (b) shows the MLIV meets the requirement of the incoming request and receives the value 0.9 for the parameter *com*. Case (c) indicates the MLIV has a map whose

completeness attribute fails to meet the need of the incoming request and thus receives a value 0.5 for the parameter *com*. The main reason for the relative low value of parameter *com* (i.e. 0.5) in case (c) is to allow the MLIV receives a lower ranking value in our ranking metric of the Quality Ranking Measure Model since it is unlikely the MLIV will generate a map with good quality in terms of *completeness*.

The values for rest of the parameters, namely pos, rel, access are generated based on the same philosophy and we only list some example rules here. The set of complete rules is listed in Appendix B.

4.3.6.2.2.1.1.5 Parameter *pos* value

if request.*positionalAccuracy* = 0.01 and MLIV.*positionalAccuracy* = 0.01

then *pos = 1*

if request.*positionalAccuracy* = 0.01 and MLIV.*positionalAccuracy* = 0.02

then *pos = 0.6*

if request.*positionalAccuracy* = 0.01 and MLIV.*positionalAccuracy* = 0.03

then *pos = 0.5*

4.3.6.2.2.1.1.6 Parameter *rel* value

if request.*reliability* = 1 and MLIV.*reliability* = 1

then *rel = 1*

if request.*reliability* = 1 and MLIV.*reliability* = 0.9

then *rel = 0.6*

if request. *reliability* = 1 and MLIV. *reliability* = 0.8

then *rel = 0.5*

4.3.6.2.2.1.1.7 Parameter *access* value

if request.*accessibility* = 5 sec and MLIV. *accessibility* = 5 sec

then *access = 1*

if request. *accessibility* = 5 sec and MLIV. *accessibility* = 10 sec

then *access = 0.6*

if request. *accessibility* = 5 sec and MLIV. *accessibility* = 25 sec

then *access = 0.5*

4.3.6.2.2.2 Parameter *res* value generation model

Resolution refers to the smallest size of geographic object that can be mapped to the data model (Burrough 1998). A spatial object with a finer resolution indicates it has more cells with smaller size than the one with a coarser resolution for a same coverage. A MLIV with finer resolution has a smaller numerical value in its *resolution* attribute and a MLIV coarser resolution has a greater numerical value. If the resolution of MLIV does not match with the one associated with the incoming

request a conversion is needed. A conversion from finer to coarser will suffer from the conversion loss which means the spatial object cannot be fully recoverable after a conversion is performed with respect to the resolution. Since the MLIV has a finer resolution it has cells with smaller size and multiple smaller cells need to merge to form a single bigger cell in order to convert into a coarser resolution. Unless a sampling process from the ground feature can be conducted there is no way to assign a value into this single cell. One well known solution is by prediction; the value of this single bigger cell is predicted based on the value of smaller cells from which it is converted.

In our approach, we use some well accepted concepts of interpolation process in geographic science. Interpolation is the prediction of a value of an attribute at an unsampled site X_0 from measurements made at other sites X_i falling within a given neighborhood. The rationale behind interpolation "is the very common observation that, on average, values at points close together in space are more likely to be similar than points further apart "(Burrough 1986). In another words, cells with the same values tends to cluster together. Based on the concept, we develop the Resolution Value Generation Model which is explained in the following paragraph. We introduce a function RF(x,y) that generates the value for the parameter *res* for inclusion in the ***RS*** rules. The values assigned to the a_i in the RF(x,y)were based on this concept and

we also conduct an empirical study to determine the values of a_i(discussed in Chapter 5).

The value of parameter *res* is determined by the following rules:

If MLIV.*resolution* = x and req.*resolution* = y then *res* = *RF(x, y)*.

The function *RF(x,y)* that computes the values is defined as follows:

$$RF(x,y) = \begin{cases} 1 & \text{where } x = y \\ \sum_{i=1}^{n^2}(i * a_i) \,/\, n^2 & \text{where } x \; y \\ & n = x / y \\ & a_i = \text{probability that } i \text{ is the number that match the one chosen and} \\ & \sum_{i=1}^{n^2} a_i = 1 \qquad a_i = \frac{1}{n^2} \end{cases}$$

4.3.6.2.3 Weight generation approaches

The ranking metric defined in Section 4.3.6.2 Quality Ranking Measure Model is as follows:

v = w_1**com* + w_2**file* + w_3* *pos* + w_4* *rel* + w_5 **res* + w_6**access, where*

w_i is the weight associated with *ith* parameter

In previous section, we introduced two approaches to generate the parameter values in the quality ranking metric above, namely, *com, file, pos, rel, res, access*. In

this section, we introduce two approaches that generates weight w_i in the ranking metric.

The reason why the values from weights for a system like GeoGrid cannot be the same as the way (Mountrakis 2004, Lim 2001) is that they looked at a set of maps as pictures and had the users rank them and it is not a possible way in a large dynamic system like GeoGrid.

We develop two approaches to identify weights. They are described in the following sections. Evaluations for each approach are given in detail in Chapter 5.

During a preprocessing stage, the spatial mediator generates a set of weight vectors, ***W*.** Each weight vector is corresponding with a given request. Since the number of possible combination of attribute values is already known it means all possible requests without bounding boxes are also known. This makes it possible to calculate the weight vector associated with each request. During the run time, when the spatial mediator receives the incoming request (with bounding box) it obtains the associated weight vector from the set ***W*** and calculates the ranking values for the MLIVs in *FullCoverageList* and *PartialCoverageList*.

The diagram shown in Figure 4.20 indicates how the weights are integrated into the spatial mediator during the running stage.

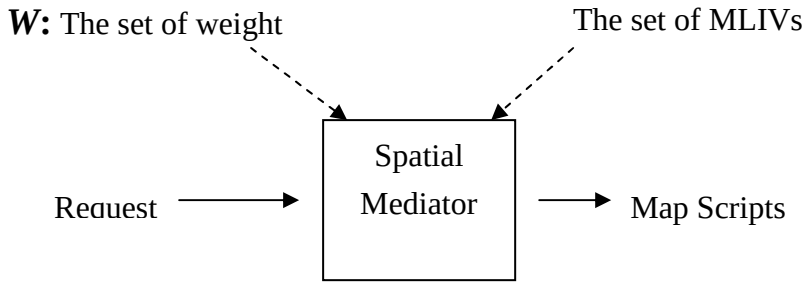


Figure 4.20. The spatial mediator in the running stage.

4.3.6.2.3.1 Partitioning weight generation approach

To calculate the parameter weights, we make use of the attribute value partition, and the availability of conversion tools. We use a matrix of weights where each line in the matrix represents a set of weights for a potential request. The set of MLIVs ***L*** is partitioned into four sets representing the system's ability to do any conversion required to use an MLIV to process the request. The four sets of MLIVs are

- **A** is the set of MLIVs that can be used in the response without conversion,
- **B** is the set of MLIVs that can be used in the response without loss and the necessary conversion tools are supported by the system,
- **C** is the set of MLIVs that can be used, but will suffer some loss of quality using the current conversion tools supported by the system, and
- **D** is the set of MLIVs where either conversion can't be done or no tools exist.

An iterative algorithm is applied to the four sets of MLIVs to adjust the weights to enforce the partial ordering implied by the four sets. The correct partial ordering is indicated by the ranking values of MLIVs inside these sets. The ranking values of MLIVs in set **A** are higher than the ones in set **B** and the ranking values of MLIVs in set **B** are higher than the ones in set **C** and same follows for set **C** and set **D**. The ranking values of MLIVs in set **A** are higher than ranking values of MLIVs in set **B** because the MLIVs in set **A** can generate maps without conversion, i.e. maps without quality loss due to conversion. In other words, MLIVs in set **A** generate maps with better quality than maps generated by MLIVs in set **B**.

The algorithm starts with a set of initial values for the weight vector and “tunes” the weight values until the partial ordering implied by the four sets is true. The “tune” means that the algorithm updates the weight values by a gradient change in each iteration. The documentation of the algorithm is shown in Figure 4.21(a) and algorithm is presented in Figure 4.21(b).

Input:

1. Vector of quality requirement of Request: Qual_R // It has six values for the quality attributes
2. Vector of initial weights: Initial_W // Value 0.5 is set for each element in vector Initial_W.
3. Set of MLIVs: MLIV // It is an object that has quality attributes (qual), label, ranking values.

output:

Vector of tuned weights: final_W

// This algorithm is used to generate a weight vector for a corresponding request.
// This algorithm identifies the corresponding weight values υ for the corresponding request.
// Each MLIV is evaluated by the ranking value υ which is defined as
// $\upsilon = w_1 * com + w_2 * file + w_3 * pos + w_4 * rel + w_5 * res + w_6 * access$
// To calculate the weights in the above formula, we make use of rules in the Rule Set and data from Fact Database.
// A vector of weights will be generated by this algorithm.
// This algorithm is decomposed into four distinct phases. For each request, this algorithm will go from phase I through
// phase IV. Phase II, Phase III and Phase IV work as a loop. If the process doesn't meet the halting condition in Phase IV
// then the algorithm goes back to Phase II, Phase III and Phase IV. When the algorithm stops a weight vector
// containing six weight values is generated
// Phase I LIV Partitioning: MLIVs will be partitioned into four sets, they are D, C, B and A.
// A is the set of LIVs that can be used in the response without conversion,
// B is the set of LIVs that can be used in the response without loss and the necessary conversion tools
// are supported by the system
// C is the set of LIVs that can be used, but will suffer some loss of quality using the current conversion tools supported
// by the system, and D is the set of LIVs where either conversion can't be done or no tools exist.
// Initially, all LIVs are not labeled. The algorithm starts to check each LIV with respect to a particular request. LIVs that are
// identified as the member of set D will be labeled first. And LIVs evaluated as members of set C, set B and set A will be
// labeled accordingly.
// Phase II Ranking Value Generating: The ranking value for each LIV will be generated by using the ranking formula stated
// above. For each parameter value in the formula, the algorithm calls a module ruleEngine. The variable parmValue is a
// vector that holds the parm values for the MLIV. "parmValue = ruleEngine(MLIV, Qual_R);"
// A method, compuRanValue, is called to calculate the ranking value for the corresponding MLIV

// Here comes the tuning process
// " while (NOT done)" is used as a halting condition
// Phase IV Weight Tuning: The algorithm will stop if the following condition is true otherwise the
// tuning process begins.
// $\varepsilon >= (e(A,B) * \theta_1 + e(A,C) * \theta_2 + e(A,D) * \theta_3 + e(B,C) * \theta_4 + e(B,D) * \theta_5 + e(C,D) * \theta_6$
// where ε is the threshold and e(A,B) is the number of incorrectly positioned MLIVs in partition A
// and partition B, etc. The threshold is set manually. A method, testErrorNum, is used to get ε.
// A method, getNoIncorrectMLIV, is used to compute the number of incorrectly positioned MLIVs
// The weight tuning procedure starts with a small adjustment on θ_i for $1 <= i <= 6$. These adjustments
// are made one at a time starting from θ_1. The algorithm will check the halting condition for every
//adjustment. A method, tuneTheta, is used to tune the θ_i
//When all θ_i for $1 <= i <= 6$ are adjusted and the halting condition is still false, the tuning
//process starts to make adjustments on the weights w_i for $1 <= i <= 6$. w_i are adjusted by small
//increment (+0.01) or decrement (-0.01) one at a time and is done in method: tuningWeights.

Figure 4.21a. Comments of partitioning weight generation approach

Input:

1. Vector of quality requirement of Request: Qual_R // It has six values for the quality attributes
2. Vector of initial weights: Initial_W // Value 0.5 is set for each element in vector Initial_W.
3. Set of MLIVs: MLIV // It is an object that has quality attributes (qual), label, ranking values.

Output:

Vector of tuned weights: fianl_W

```
// Phase I   LIV Partitioning:   MLIVs will be partitioned into four sets, they are D, C, B and A.
if (MLIV. completeness < Qual_R.completeness) then MLIV.label = "D";
      else if (MLIV.positional accuracy > Qual_R.positional accuracy) then MLIV.label = "D"
            else if (MLIV.reliability < Qual_R.reliability) then MLIV.label = "D"
                  else if (MLIV. accessibility > Qual_R.accessibility) then MLIV.label = "D"
// Check if an unlabeled LIV belongs to Set C
if (MLIV.resolution > Qual_R.resolution) then MLIV.label = "C" ;
      else if (MLIV. file = raster and Qual_R.file = vector) then MLIV.label = "C";
            else if (MLIV. file = lossyRaster and Qual_R.file = vector) then MLIV.label = "C";
                  else if (MLIV. file = lossyRaster and Qual_R.file= raster) then MLIV.label = "C";
//Check if an unlabeled LIV belongs to Set B that can be used in the response without loss
if (MLIV.resolution   <   Qual_R.resolution) then MLIV.label = "B";
      else if (MLIV.file = vector and Qual_R.file = raster) then MLIV.label = "B";
            else if (MLIV.file = vector and Qual_R.file = lossyRaster) then MLIV.label = "B";
                  else if (MLIV.file = raster and Qual_R.file = lossyRaster) then MLIV.label = "B";

// If an MLIV is unlabeled then it must belongs to Set A
      if (MLIV.label == " ") then MLIV.label = "A";
// Phase II Ranking Value Generating:
parmValue = ruleEngine(MLIV, Qual_R);
for (i = 1; i++; i<=6){
      updated_W.element(i) = intial_W.element(i) ;
// Here comes the tuning process
boolean done = false;
while (NOT done) {
      MLIV.rankingValue = compuRanValue(parmValue, updated_W);
      for (i = 1; i++; i<=6){    //Phase IV Weight Tuning:
            theta [i] = 0.5; // Initialize the theta with value 0.5
      arrayNumIncorrectM[1] = getNoIncorrectMLIV(parA, parB);
      arrayNumIncorrectM[2] = getNoIncorrectMLIV(parA, parC);
      arrayNumIncorrectM[3] = getNoIncorrectMLIV(parA, parD);
      arrayNumIncorrectM[4] = getNoIncorrectMLIV(parB, parC);
      arrayNumIncorrectM[5] = getNoIncorrectMLIV(parB, parD);
      arrayNumIncorrectM[6] = getNoIncorrectMLIV(parC, parD);
      i = 1;
      if (ε >= testErrorNum(arrayNumIncorrectM, theta[i]))
            done = true;
      while ((i <= 6) and (NOT done)) {
            tunetheta(i);
            i++;
            if (ε >= testErrorNum(arrayNumIncorrectM, theta[i]))
                  done = true;
      }
      if (NOT done) then
            updated_W = tuningWeights(updated_W);
      else final_W = updated_W;
}
```

Figure 4.21b. Partitioning weight generation algorithm.

4.3.6.2.3.2 Map grouping weight generation approach

Another method to generate weight values in the quality metric of the Quality Ranking Measure Model employed by the spatial mediator makes use of experts. The Map Grouping Weight Generation Approach is used when *FullCoverageList* is empty or doesn't contain any MLIVs above the threshold value, that is, spatial mediator cannot find a MLIV that can generate a map whose bounding box covers the one with the map request at a sufficient level of quality. After spatial mediator identifies all possible map groupings from the MLIVs in *PartialCoverageList* for the incoming request, the *correct map grouping sequence* is identified using geographic interpretation of available data. An iterative algorithm is applied to the MLIVs to enforce the *correct map grouping sequence*.

The ranking value for a map grouping is defined as the minimum ranking value within the group, i.e., the minimum ranking value associated with a MLIV within the group. Before we introduce the *correct map grouping sequence* we first present the following definitions. We have defined the *grouping* and *map grouping* in Section 4.3.3

Definition 4.13:

MG_i is defined as a *preferred map grouping* over MG_j *iff* MG_i generates a map with better quality than the map generated by MG_j

One *map grouping* is identified as a *preferred map grouping* over another *map grouping* based on the need of application and also on our knowledge of geographic data quality. To model the decisions is not the focus of this research. We reveal some of the rational behind the decisions in the following paragraph.

Rationale behind the decision in the Map Grouping Weight Generation in deciding a *preferred map grouping*:

R1: The grouping with the least number of MLIVs is preferred.

R2: The least number of "file conversions" needed inside the map grouping is preferred.

R3: The least number of "resolution conversions" needed inside the map grouping is preferred.

R4: The higher value in non-convertible parameters is preferred.

The rationales are written in the order of priorities, the one on top has the higher priority over the one below.

We use the following example to demonstrate the rationales behind the decision.

Attribute	Completeness	Map type	Positional accuracy	reliability	resolution	accessibility
REQ	***0.9***	***SHAPE***	***0.01***	***1.0***	***25***	***6***

==========

L41	1	'SHAPE'	0.01	1	25	6
L43	1	'SHAPE'	0.01	1	1	4
L45	1	'SHAPE'	0.01	1	10	4
L46	1	'SHAPE'	0.01	1	25	5
L47	1	'SHAPE'	0.01	1	5	6
L49	1	'SHAPE'	0.01	1	25	4
L50	0.9	'TIFF'	0.02	0.95	25	10
L55	0.8	'JPG'	0.03	0.5	100	15
L58	0.9	'SHAPE'	0.01	1	25	5
L59	1	'TIFF'	0.01	1	25	10

Map grouping:

MG_1: <L41, L49>

MG_2: <L41, L46, L58>

MG_3: <L43, L45, L47>

MG_4: <L41, L45, L55>

Best

Poorest

Based on the rational stated above, the map grouping MG_1 is the *preferred map grouping* over MG_2, and MG_2 is the *preferred map grouping* over MG_3, etc,. In another word, MG_1 is the best *map grouping* and the MG_4 is the least preferred. The reasons are as follows:

- There are only two MLIVs in the map grouping MG_1, fewer than others.
- File types and resolution are all the same with the group MG_1, MG_2, MG_3.
- The map grouping MG_2 is preferred than MG_3 is because file types of all three MLIVs are all SHAPE files, resolution are the same too, no conversion is needed.
- The map grouping MG_3 is preferred than MG_4 is because no file conversion is needed in MG_3 while a raster-to-vector conversion is needed in MG_4.

We now define a *correct map grouping sequence* by first introducing *a correct local sequence* of a *map grouping.*

Definition 4.14:

Correct local sequence of a *map grouping* MG_k = {$MLIV_0$,..$MLIV_i$, $MLIV_j$,..$MLIV_m$} is defined as a sequence that satisfies the following condition:

For each pair of $MLIV_i$ and $MLIV_j$, the ranking value of $MLIV_i$ is higher than the ranking value of $MLIV_j$ where $MLIV_i$ and $MLIV_j$ are i^{th} and j^{th} term in the sequence respectively with 0<= i < j<= m

Definition 4.15:

A *correct map grouping sequence:*{$MG_{0,..}$ MG_i, $MG_{j,..}$ MG_m}is defined as a sequence that satisfies the following two conditions:

(1) $min(MG_i) > min(MG_j)$ where *min* is the minimum function that return the minimum ranking value inside a *map grouping* MG_i and MG_j are *map grouping* with *correct local sequence* and i^{th} and j^{th} term in the sequence respectively with 0<=i < j<= m.

(2) MG_i is a *preferred map grouping* over MG_j.

In the Map Grouping Weight Generation Approach spatial mediator uses an iteration algorithm to generate weight values in the quality measure metric of the Quality Ranking Measure Model. With a given request and the *PartialCoverageList*, the *correct map grouping sequence* is identified with respect to the given request. The

algorithm then starts with a set of initial values for the weight vector and updates the weight values with a gradient change until the *correct map grouping sequence* is found. Once the *correct map grouping sequence* is found the weight vector is stored in the set of weight vectors, $\boldsymbol{W}$ mentioned in Section 4.3.6.2. Figure 4.22 shows the algorithm.

```
Input:
4.      Vector of initial weights: Initial_W // Value 0.5 is set for each element in vector Initial_W.
5.      Vector of MGs: MG_Vec // It is a vector of map groupings generated by the spatial mediator.
                              //MG is an object that has id and a set of MLIVs which composes the map gouping
                              //MLIV is an object that has quality attributes (qual), label, ranking value

Output:
        Vector of tuned weights: final_W

// To ensure the correct local sequence of a map grouping, a method, checkLocalSeq, is used
// The method, correctMGSeq, checks to see if the map groupings inside the vector MG_Vec is in correct map grouping
// sequence, that is min(MGi) is less than min(MGj) where MGi is located before MGj inside the vector.
// The method, preferred(MGi, MGj), returns true if MGi is a preferred map grouping than MGj
// The method, min(MG), returns the minimum ranking values of MLIV inside the map grouping MG.

boolean correctMGSeq(MG_Vec) {
int i = 0;
int j = i + 1;
Boolean continue = true;
while (j <= MG_Vec.size() and continue) {
        if (min(MG_Vec.elementAt(i)) < min(MG_Vec.elementAt(j)))
                continue = false;
        else {
                if (preferred(MG_Vec.elementAt(i), MG_Vec.elementAt(j)){
                        i++;
                j = i;
                }
                else continue = false;
        }
}

for (i = 1; i++; i<=6){
                updated_W.element(i) = intial_W.element(i) ;
boolean done = false;
If (correctMGSeq(MG_Vec)) then done = true;
//Tuning process starts if the MG_Vec doesn't hold the correct map grouping sequence
while (NOT done) {
        updated_W = tuningWeights(updated_W);
        while ( NOT MG_Vec.isEmpty()){
                MLIV = MG_Vect.nextElement();
                parmValue = ruleEngine(MLIV, Qual_R);
                MLIV.rankingValue = compuRanValue(parmValue, updated_W);
        }
        done = correctMGSeq(MG_Vec);
}
if (done) then final_W = updated_W;
```

Figure 4.22. Map grouping weight generation algorithm.

4.3.6.3 Scoring function ranking model

To overcome the limitations inherent in the use of the ranking value, we introduce a second model that makes use of an MLIV ranking function that considers the application developers concerns of attribute importance. The main value of this approach is that it removes the need to create the somewhat artificial separation between parameter and weight values.

4.3.6.3.1 Single attribute scoring function

The scoring function of a $MLIV_{i,}$ S_i, depends on six attributes which are the same as the one used in the quality measure metric of the Quality Ranking Measure Model, namely *completeness, mapType, positional accuracy, reliability, resolution and accessibility*. A score $S^{r\,j}_i$ indicates the degree of quality superiority in the j^{th} parameter for the $MLIV_i$ with respect to the incoming request *r*. The single attribute scoring function is defined as the follows:

Definition 4.16:

$S^{r\,j}_i = B^j * M^{r\,j}_i$ where

B^j is base score for attribute j

$M^{r\,j}_i$ is the magnitude for attribute j with respect to $MLIV_i$ and request ***r***

The score $S^{r}_{i}{}^{j}$ defined above is implemented with rules in the Rule Set ***RS.*** There are two factors that affect the degrees of superiority. The first one is the base score indicating the relative importance between attributes and decided by applications /users. In our implementation, three scales are used for each attribute, namely “critical”, “important” and “non-important”, each scale is assigned with a non-zero positive numeric value; the higher the value indicates more important the attribute is to the applications /users. The applications/users can decide which attributes are more important than other attributes. The application designers (people that registered the applications) are asked to complete a preference selection when they registered with the GeoGrid infrastructure and then information is then stored in the Fact Database ***FD***.

Our motivation of developing this approach comes from our observation that different users have different needs in the multiple geographic data sources environment. For example, a user in a situation with a need of an urgent response is willing to scarify the quality loss due to the conversion of the file type to exchange with the fast accessibility of a request map. In this case, this user can mark the attribute *accessibility* with a “critical” scale and “non-important” for the attribute *mapType* when registering with the infrastructure. Another motivating scenario is when an application conducting a land use survey doesn’t ask for a fast retrieval of a

map but require a map with precise file type it can make the preference selection with attribute *mapType* marking “critical” and attribute *accessibility* marking “non-important” during the registration process.

Another factor that affects the score is magnitude of the superiority. The magnitude for each attribute is not the same and is determined based on our knowledge of geographic data quality. When the MLIV has a poor attribute compare to the one associated with the request this magnitude is assigned with a negative value. The magnitude reflects the characteristics of the attributes. For example, the non-linear characteristic of attribute *resolution* is reflected by the following rules given the base score for *resolution* is 15:

if *req.resolution* = 10 and *MLIV.resolution* = 1 then S_i^{res} = 15 * -5 = -75.

if *req.resolution* = 25 and *MLIV.resolution* = 1 then S_i^{res} = 15 * -6 = -90.

if *req.resolution* = 100 and *MLIV.resolution* = 1 then S_i^{res} = 15 * - 8 = -120.

We can see the magnitude for each rule is different, namely -5, -6 and -8, and is not proportional to the ratio of *MLIV.resolution* to *req.resolution.* Similar observations are found in (Mountrakis 2004). They require users’ interaction to quantify the users’ preferences in (Mountrakis 2004) while our approach makes use of the help from the application experts.

There are three possible cases for the score. We discuss these cases to illustrate the implementation of the rules.

Case 1: *The score is zero.* If the attribute value associated with MLIV is the same as the one included in the request, then the rule generates the value of zero for S_i^j. For example, if *req.completeness* = 0.9 and *MLIV.completeness* = 0.9 then $S_i^{com} = 0$.

Case 2: *The score is a positive value greater than zero.* If the MLIV has a better attribute value than the one of request then the rule generates non-zero positive value for S_i^j. For example, if req. *req.completeness* = 0.8 and *MLIV.completeness* = 0.9 then S_i^{com} = *pos.scr* where *pos.scr* is a numerical value and *pos.scr* > 0.

Case 3: *The score is a negative value less than zero.* An MLIV will receive a negative value in S_i^j if the MLIV has a poorer attribute value than the one of request. For example, if *req.completeness* = 0.9 and *MLIV.completeness* = 0.8 then S_i^{com} = *neg.scr* where *neg.scr* is a numerical value and *neg.scr* < 0. The idea behind this is to allow a MLIV with a poorer quality than the one associated with the request to receive a lower score.

4.3.6.3.1.1 Scoring function

We now give the definition of the scoring function. The scoring function $\boldsymbol{S}_i$ of $MLIV_i$ is the sum of scores $S^{r\,j}_i$. The following equation gives the scoring function for $MLIV_i$ with respect to the request *r*.

$$\boldsymbol{S}_i = S^{r}_{i}{}^{com} + S^{r}_{i}{}^{file} + S^{r}_{i}{}^{pos} + S^{r}_{i}{}^{rel} + S^{r}_{i}{}^{res} + S^{r}_{i}{}^{access}$$

where $S^{r\,j}_i$ is the score with respect to attribute j

Example 7:

We illustrate the scoring function by the following example. A user indicates that attribute *mapType*, *resolution* and *positional accuracy* are "critical" and the attribute *completeness* and *reliability* are "important" and attribute *accessibility* is "not-important". Application experts assign a base score 5 for the "not-important" attribute, a base score 10 for "important" attribute and 15 for the "critical" attribute. The following vectors indicate the attributes values associated with the $MLIV_i$ and request *r*.

Attribute	Completeness	Map type	Positional accuracy	reliability	resolution	accessibility
REQ *r*	***0.9***	***SHAPE***	***0.01***	***1.0***	***25***	***10***
$MLIV_i$	*1*	*SHAPE*	*0.01*	*0.9*	*25*	*25*
$MLIV_j$	*1*	*JPG*	*0.01*	*0.9*	*25*	*5*

For *$MLIV_i$* the following rules in ***RS*** are fired:

if *req.compleness* = 0.9 and *MLIV.completeness* = 1 then $S_i^{res} = 10 * 1 = 10$.

if *req.mapType* = vector and *MLIV.mapType* = vector then $S_i^{file} = 15 * 0 = 0$.

if *req.positionalAccuracy* = 0.01 and *MLIV.positionalAccuracy* = 0.01 then $S_i^{pos} = 15 * 0 = 0$.

if *req.reliability* = 1.0 and *MLIV.reliability* = 0.9 then $S_i^{rel} = 10 * -1 = -10$.

if *req.resolution* = 25 and *MLIV.resolution* = 25 then $S_i^{res} = 15 * -4 = 0$.

if *req.accessibility* = 10 and *MLIV.accessibility* = 25 then $S_i^{access} = 5 * -1 = -5$.

The score for $MLIV_i$ is sum of scores above and is -5.

For $MLIV_j$ the following rules in ***RS*** are fired:

if *req.compleness* = 0.9 and *MLIV.completeness* = 1 then $S_i^{res} = 10 * 1 = 10$.

if *req.mapType* = vector and *MLIV.mapType* = raster then $S_i^{file} = 15 * -5 = -75$.

if *req.positionalAccuracy* = 0.01 and *MLIV.positionalAccuracy* = 0.01 then $S_i^{pos} = 15 * 0 = 0$.

if *req.reliability* = 1.0 and *MLIV.reliability* = 0.9 then $S_i^{rel} = 10 * -1 = -10$.

if *req.resolution* = 25 and *MLIV.resolution* = 5 then $S_i^{res} = 15 * -4 = -60$.

if *req.accessibility* = 10 and *MLIV.accessibility* = 5 then $S_i^{access} = 5 * 1 = 5$.

The score for $MLIV_j$ is sum of scores above and is -130.

The reason for both MLIVs receiving a negative score is that attributes for MLIVs are either same as or poorer than the request. Because the designers select the attribute *mapType* as a “critical” so $MLIV_i$ receive zero for the attribute *mapType* while $MLIV_j$ obtains a negative number due to the huge negative magnitude.

Note that some additional rules need to be fired in order to convert the file type.

if fileType = SHAPE then mapType = vector

if fileType = JPG then mapType = raster

The final scores for $MLIV_i$ and $MLIV_j$ are -5 and -130 respectively. The scoring function rank $MLIV_i$ *higher than* $MLIV_j$.

The algorithm for the scoring function is given in Figure 4.23.

```
Input:
1.    Vector of quality requirement of Request: Qual_R // It has six values for the quality attributes associated with the
      given reqeust
2.    Vector of quality requirement of a MLIV // It has six values for the quality attributes associated with the given MLIV
output:

      score s for the associated MLIV

//This algorithm is used to generate a score vector for a corresponding request.
//   This algorithm identifies the corresponding score s for the corresponding request.
//   Each MLIV is evaluated by the score s which is defined as
//       S_i = S^{r_i com} + S^{r_i file} + S^{r_i pos} + S^{r_i rel} + S^{r_i res} + S^{r_i access}   Where S^{r_i j} is the score with respect to attribute j
//   To calculate the score in the above formula, we make use of rules in the Rule Set and data from Fact Database.
//   A vector of scores will be generated by this algorithm.
}
  for (i = 1; i++; I <= 6) {
      s = singleScore(attribute i);
return s;
}
/ A method, singleScore(attribute i), is used to generate a score for a signal attribute i
//   S^{r_i j} = B^j * M^{r_i j}   where B^j is base score for attribute j and
//   M^{r_i j} is the magnitude for attribute j with respect to MLIV_i and request r
// The method base(attribute i) returns the base score for the attribute i
//The method magnitude(request.attribute i, MLIV.attribute i) returns the magnitude for the attribute i
// It calls ruleModule((Qual_R.i, MLIV.i) uses the following formula to generate the desired score for the attribute i

int singleScore(attribute i) {
      int singScore;
      singScore = base(i) * magnitude(Qual_R.i, MLIV.i);
      return singScore;
}
//
int magnitude(Qual_R.i, MLIV.i) {
int mag;
if request.attribute i= Qual_R. i and MLIV. attribute I = MLIV.i then mag = ruleModule(Qual_R.i, MLIV.i);
return mag;
}
```

Figure 4.23. Scoring function algorithm.

4.4 Relational script generation

An example of the object structure of an RLIV is shown in Figure 4.24. A block diagram of the process for generating the relational script is shown in Figure 4.25. The distributed nature of data sources supporting the spatial mediator means that two types of queries are needed to produce the relational results, namely, subqueries for each RLIVand a *framework query* to combine the RLIVs.

```
package relationalLIV;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
public class Querydb {
    public Relation derive (String query) {
        private Relation rel = new Relation():
        Class.forName("com.mysql.jdbc.Driver");
        String url = "jdbc:mysql://localhost/geogrid";
        Connection connection = DriverManager.getConnection(url);
        Statement stmt = connection.createStatement();
        ResultSet rs = stmt.executeQuery(query);
        ResultSetMetaData rsmd = rs.getMetaData();
        int columnCount = rsmd.getColumnCount();
        String tuple = "";
        while (rs.next()) {
                for (int col = 1; col <= columnCount; col++) {
                        tuple = tuple + rs.getString(col);
                        if (col < columnCount) tuple = tuple + "\t";
                                rel.addCurrentTuple(tuple);}
                tuple = "";}
        return rel;}
```

Figure 4.24. An example of the object structure for RLIV.

To connect the relational data with the map, we make use of the service inside the computation server to obtain a geographic location for a corresponding spatial data.

The actual connection depends on the nature of the data that is returned to the computation server. Occasionally, the data may have coordinate information as attributes in the table obtained form an RLIV. In those cases, the merge tool can use the data directly. However, in general the available spatial data will have a geographic connection only through state, city or address values. We make use of two tools in the computation server to fine the appropriate coordinate information. First, for purely symbolic data values like a state name (e.g. Iowa), we use a computation based tool based on the ***GSO*****.** For specific addresses, the computation server makes use of a web-tool, such as *GPS Visualizer* (http://www.gpsvisualizer.com/geocoder/) to convert the addresses to geographic coordinates.

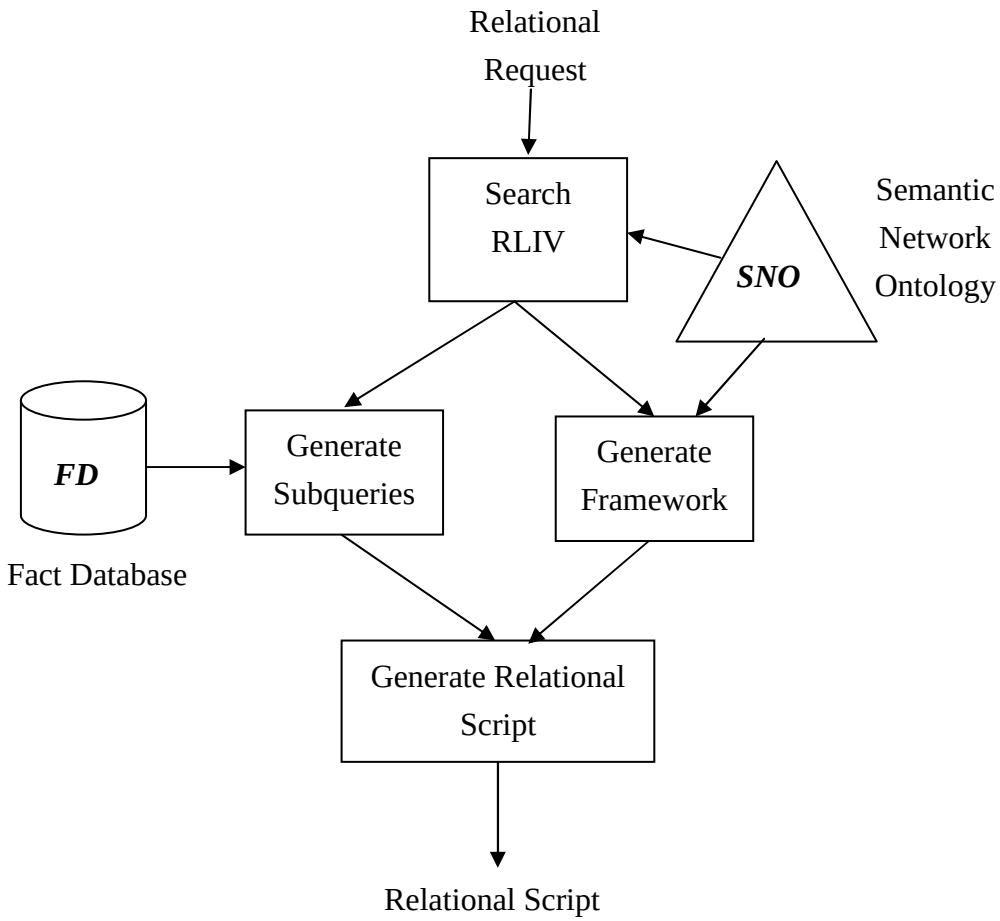


Figure 4.25. The process for generating a relational script.

A query has to be generated for each RLIV that is used to generate the final result. We use the term subquery for these queries. The computation server distributes the subqueries to the data sources that support the RLIV over which they are defined. The results of the subqueries are returned to the computation server as individual relations.

To combine the resulting relations into the final result, the spatial mediator creates a *framework query*.

Definition 4.17:

The *framework query* is defined as a query of the form:

$R_1, join_1, R_2\ join_2 \ldots join_{m-1}\ R_m$ where

R_i, i = 1,2,…,m-1 are connected in the semantic network of the ***SNO*** with a join criteria ($join_i$ i=1,2,…,m-1) defined in the association node that connects the two RLIVs.

The computation server receives the subqueries and the *framework query* as part of the integration script and uses it to guide the distribution of the subQueries and the generation of the final result. The process of generating the relational script starts with the spatial mediator determines the appropriate RLIVs for the relational request. Then it generates the subqueries for each of the required RLIVs, determines the correct *framework query* for combining the RLIVs, and creates a relational script. We formally define the *relational script* in the following paragraph and then elaborate the process in the following sections.

Definition 4.18:

A *relational script* has the form:

relation (<$\mathbf{S}_1$; $\mathbf{S}_2$;…$\mathbf{S}_m$><**A**, **C**> **F**) where

$\mathbf{S}_i$ are subquery where i = 1,..m

A is attribute list and **C** is the condition from the user request

F is framework query

4.4.1 Search RLIVs

The search for the RLIVs needed makes use of the Fact Database (***FD)*** and the Semantic Network Ontology ***(SNO).*** The ***SNO*** consists of a triple (Σ,θ,ϑ), where Σ is the set of search operations defined in Chapter 3, θ is an ontology and ϑ is a semantic network of RLIVs connected by association nodes that define the join criteria for two RLIVs. The search terms from the relational request are used to search θ to locate the RLIVs that contain the attributes necessary to respond to the user request and are connected by join criteria through the connection in ϑ.

We start with an example before examining the algorithm to illustrate how the Semantic Network Ontology ***(SNO)*** is used to locate the RLIVs and how to combine the relations from RLIVs. An application generates a requests with request id 1008 and ask for people's id who is injured in event of a natural disaster. To answer this request the spatial mediator searches ontology of ***SNO*** and locates $RLIV_1$ that provides the properties information such as owner's id. Another data source; $RLIV_3$ that has the natural disaster reports is also identified by the spatial mediator. The fragment shown in Figure 4.26 illustrates the search terms from for relational request which are *identity* and *natural disaster* are used to search the ontology θ and locates two RLIVs namely, $RLIV_1$ and $RLIV_3$. When the search algorithm stops two

attributes namely, *id* and *stormName* are referenced by pointers from the algorithm. Since *id* is an attribute of $RLIV_1$ the spatial mediator identifies $RLIV_1$ as a RLIV containing information relevant to the request. The spatial mediator also finds $RLIV_3$ because *stormName* is attribute that belongs to $RLIV_3$. As one can be seen from this example, the ontology portion of the ***SNO*** allows us to find a set of RLIVs that contain the information required to respond to the relational request.

Unfortunately the RLIVs found in this search process may not typically not be connected by join criteria. In the example (Figure 4.26) the search process returned $RLIV_1$ and $RLIV_3$, but the semantic network portion of ***SNO*** doesn't define a way of joining the result of the two RLIVs (Figure 4.26). It is clear from the example that $RLIV_2$ has to be included in order be able to join $RLIV_1$ and $RLIV_3$ to generate the results needed to respond to the request.

The ontology θ of the ***SNO*** which is the upper portion of supports the search of RLIVs while the lower portion of ***SNO*** defines the join criteria for combining the RLIVs together. In Figure 4.26, the join criteria shows that join condition for $RLIV_1$ and $RLIV_2$ is "$RLIV_1$. id = $RLIV_2$.ownerId" and join condition for $RLIV_2$ and $RLIV_3$ is "$RLIV_2$. address = $RLIV_3$.location". The relation schemes inside each RLIVs are stored inside the Fact Database (***FD***). Note that Figure 4.26 only shows a fragment of ***SNO.***

Relation schemes for each RLIV:

$RLIV_1$ (Gender (genderType, gender), Resident (name, id))

$RLIV_2$ (Property (ownerId, address), Facility (type, capacity))

Corresponding request: *<r1008, {identity, natural disaster}>*

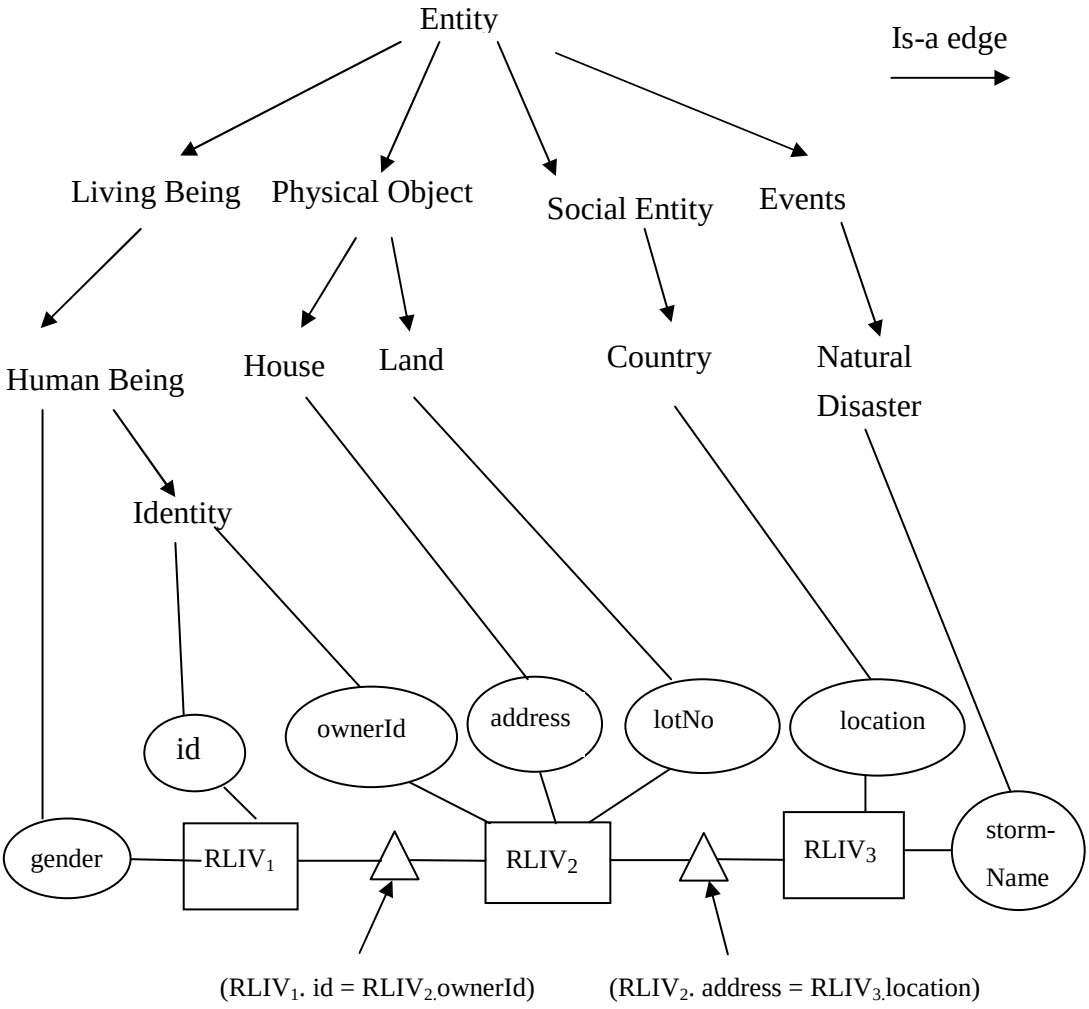


Figure 4.26. A fragment of Semantic Network Ontology.

Once all of the RLIVs (and join criteria) are identified for the *framework query*, the spatial mediator must determine the subqueries that will have to be sent to each RLIV that appears in the *framework query*. The subqueries must return any attributes needed to respond to the request along with the attributes needed by the join criteria. While any type of join criteria could be supported by our ***SNO*** model (Chapter 3), for this thesis we have restricted the join criteria to be equijoins. Also while any joins supported by relational database management systems can be supported for the subqueries, we have assumed that the joins are natural joins.

In the remainder of this chapter we look in detail at the algorithms for generating the RLIV subqueries and *framework query*, respectively.

4.4.2 Generate SubQueries

Each RLIV defines a set of relations such that the registered views have undergone the renaming process (Miller et al. 2002). When there are two attributes within different relations refer to the same characteristics and have different names then attributes are renamed by using view. In a similar way attributes with different semantics are given different names through the views. The Fact Database contains the information on the relations that exist for each RLIV. For example, the attributes stored in each relation (the attributes might have been renamed under the renaming

process), attribute data types and the functional dependencies that have included by the individual registering the RLIV.

The relations registered for an RLIV may or may not have the lossess join property, but the RLIVs that support this property are registered by the owners as supporting the universal relation principle.

Since we assume that the renaming process has been used to make the attribute names uniform, we are able to represent the relations defined for an RLIV as a hypergraph.

A hypergraph is a couple H = (N,E) where N is a set of vertices, and E is a set of hyperedges which are non-empty subsets of N (Berge 1973). A natural coorespondance existes between a hypergraph and a database where N is the set of attributes and E is the set of the relations schemes for the database. For example, the hypergraph in Figure 4.27 represent a database scheme R = {R_1, R_2, R_3} where R_1 (A, B, C), R_2 (C, D, E), R_3(A, E) are relation schemes. The hypergraph can be used to model both the logical design and query operations (Owrang & Miller 1988).

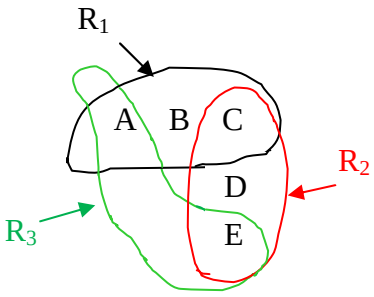


Figure 4.27. The hypergraph representation of database scheme R.

A hypergraph is connected if every pair of its hyperedges is connected by some path of hyperedges. In our algorithms, individual RLIVs are represented by a hypergraphs.

The algorithm generates the query needed for one RLIV. Naturally, the spatial mediator must use the algorithm for each RLIV needed in the relational script.

Algorithm subquery {

$\mathscr{R}$= relation schemas for LIV;
F = Fds defined over R;
$\mathscr{A}$= attributes required from LIV;
rest = relation conditions and additional clauses;
$\mathscr{S}$= getJoinSequence ($\mathscr{R}$, $\mathscr{A}$);
$\mathscr{C}$= $\mathscr{R}$ – $\mathscr{S}$
while $\mathscr{S}$changes
for every s $\in \mathscr{S}$
for every $\mathscr{R}$ $\in \mathscr{S}$ – {s}
if $\mathscr{R} \cap \mathscr{S} \rightarrow$ W $\in F^+$ where W $\subseteq \mathscr{R}$
s = s + W;

$\mathscr{G}$ = getConnectedComponents ($\mathscr{C}$);
For (every G $\in \mathscr{G}$)
If (!oneEdge(G, $\mathscr{S}$))
$\mathscr{S}$= addEdges(G,$\mathscr{S}$, *F*);
Subquery = generateQuery($\mathscr{A}$,$\mathscr{S}$, *rest*);
}

The focus of the algorithm is to obtain a query that can be executed at the data source site that supports the RLIV the query is generated for. The starting conditions of the algorithm include the set of relation schemes R supported by the RLIV

(obtained from the Fact Database), the functional dependencies defined over R, the set of attributes required from the RLIV (those needed in the request result and those needed by the *framework query* equijoins), and any conditions required by application initiating the request (e.g. WHERE clause). The algorithm starts by obtaining an initial join sequence (the relations that contain the needed attributes and those that are needed to provide a connected join result). The functional dependencies are used to create what are called fd-hyperedge (Miller 1992) in the underlying hypergraph. The fd-hyperedges are used to test whether connected subcomponents of the underlying hypergraph that are not included in the initial join set ($\mathscr{S}$) intersect with just one fd-hyperedge. The motivation for including this in the subquery algorithm is that we are able to maintain losslessness in the subquery if the relations defined by $\mathscr{R}$ support the universal relation principle. More details on this process are given in Chapter 5.

An overview of the functionality of the methods called in the *subquery algorithm* is given in Table 4.3. We provide the pseudo code for some of the less traditional methods in the remainder of this subsection.

```
Algorithm oneEdge(G, 𝒮) {
        for (every G  ∈𝒢)
           if (attr(G) intersects only one edge of 𝒮)
                        return true;
           else
                        return false;
}
```

```
Algorithm addEdges(G,𝒮, F) {
    for (every G ∈𝒢)
        if (!oneEdge({G},𝒮)) {
            𝒩 = addNewEdges(𝒮, G);
            𝒮=𝒮U 𝒩
        }
        𝒮=expand(𝒮F)
    return 𝒮
}

Algorithm addNewEdges(𝒮, G) {
    for (every G ∈𝒢)
      while changes occur {
        for every g ∈G
            if ((g ∩ attr(𝒮))      Ø) {
                𝒮= 𝒮U {g};
                G = G – {g};
            }
    return 𝒮
}
```

Table 4.3. Methods used in *subquery algorithm.*

Name of Method	Description
getJoinSequence (ℛ, 𝒜)	This method implements a join sequence generation scheme as the one presented in (Owrang & Miller 1988)..
getConnectedComponents(ℰ)	This method takes a set of edges and returns a set of connected components.
oneEdge(G, 𝒮)	This method tries to collapse the edges of G on one edge of 𝒮
addEdges(G,𝒮, F)	This method adds edges to G if no one edge of 𝒮is found.
generateQuery(𝒜,𝒮, rest)	This method use the relations defined by the edge in 𝒮, attributes in 𝒜, join conditions generated from the Fact Database and condition in the parameter *rest* for the where clause to generate SQL subquery.

4.4.3 Generate framework query

The framework query provides a structure for combining the subqueries generated for each RLIV.

Algorithm frameworkQuery {
ϑ = semantic network portion of ***SNO***
FrameworkRest = relational conditions and additional clause;
$\mathscr{A}$ = attributes required from RLIV;
$\mathscr{S}$ = getFrameworkJoinSequence (ϑ);
$\mathscr{G}$ = getFrameworkConnectedComponents (ϑ, $\mathscr{S}$);
For every G ∈ $\mathscr{G}$
If (!oneRLIV(G,$\mathscr{S}$)
$\mathscr{S}$ = addRLIVs(G,$\mathscr{S}$);
frameworkQuery = getFrameWorkQuery($\mathscr{A}$, $\mathscr{S}$FrameworkRest);
}

The *framework query algorithm* is similar to the *subquery algorithm* in that it operates on a hypergraph (see proof of Lemma 4 for details on how the hypergraph for this algorithm is constructed). Note that a key difference here is that the RLIV have been registered independently and that a lot less information is known about the inter relationships of the attributes in different RLIVs. We only assume that the join attributes in the equijoin statements must be equal and that we can use the unique names for those attributes.

An overview of the functionality of the methods called in the f*ramework query algorithm* are presented in Table 4.4.

Figure 4.28 shows an example of the framework query and *relational script* generated by the spatial mediator to respond to the request: *r1008* shown in Figure 4.28.

request: *<r1008, {identity, natural disaster}>*

Framework query:

$RLIV_1.$ *id* = $RLIV_2.$*ownerId* and $RLIV_2.$ *address* = $RLIV_3.$*location*

Relational script:

relation(<"id", $RLIV_1$; *"ownerId, address"*, $RLIV_2$; *"location, stormName"*, $RLIV_3$><*id, stormName*>*"*$RLIV_1.$ *id* = $RLIV_2.$*ownerId* and $RLIV_2.$ *address* = $RLIV_3.$*location")*

Figure 4.28 An example of a framework query.

Table 4.4. Methods used in *frameworkQuery algorithm.*

Name of Method	Description
getFrameworkJoinSequence(ϑ)	This method uses Semantic Network Ontology and Fact Database to identify the RLIVs needed to respond to the request and chooses additional LIVs to ensure a connected set of RLIVs (if possible)
getFrameworkconnectedComponents(ϑ, $\mathscr{S}$)	This method determines RLIVs in ϑ that are not in $\mathscr{S}$and returns a set of the connected components of the RLIVs not in $\mathscr{S}$.
oneRLIV(G, $\mathscr{S}$)	The method returns TRUE if the RLIVs in G are connected to at most one RLIV in $\mathscr{S}$.
addRLIV(G, $\mathscr{S}$)	This method adds RLIVs connected to $\mathscr{S}$until all the RLIVs in G have been added or the remaining RLIVs in G connect to only one RLIV in $\mathscr{S}$.

4.5 Integration script generation

Before we present the process to generate the *integration script* we formally define the *integration script*.

Definition 4.19:

The *integration script* has the form:

O(**R**;**M**) where

O is either a merge tool which has the form (tool_name, tool_location, mergeToolType) or ***NoOp*** where indicates no tool is needed

R is the *relational script*

M is the *map script*

The spatial mediator replaces the **O** of the *integration script* with a ***NoOp*** for the map request and relational request. The process for generating *integration script* is the same as the one generating *relational script* or *map script* in such case. For a merge request, the spatial mediator first generates both *relational script* and *map script* and then replaces the **O** of the *integration script* with merge token.

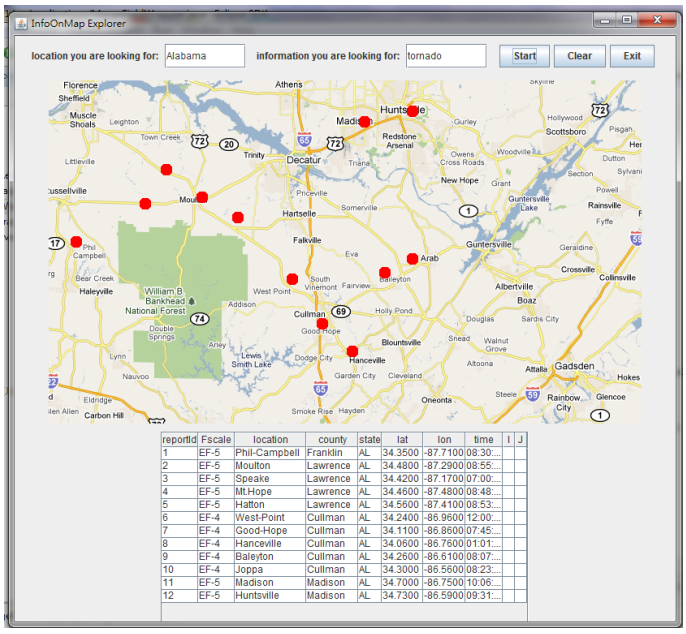
Figure 4.29 shows an example of the integration script in response to a request. Figure 4.30 shows a map results from execution of the integration script shown in Figure 4.29. $MLIV_1$ has the correct resolution but needs to be converted to JPG. Finally $MLIV_2$ is JPG, but needs to be converted to the appropriate resolution

The request: *<r1009, {Alabama}, {<0.8, JPG, 0.02, 0.9, 25, 45>, counties}, {tornado, scale, location }, {date = '2011-4-28'}>*

The corresponding integration script:

(merge, IP of mergeTool, mergeToolType)(map ((<mosaic, IP of mosaicTool, combineTypeTool>)(<convertJPG, IP of convertJPGTool, convertJPGToolType>(IP of $MLIV_1$, <" MapServer", "5,5,72" >, <" 34.350015,-87.712215", "34.482731, -87.290015" >)),(<setRes, IP of setResTool, setResolutionType>(IP of $MLIV_2$, <" MapServer", "5,5,72" >, <" 34.712204, -86.75221", " 34.730743, -86.590924" >,25)));
relation(<"reportId, scale", $RLIV_1$; "id, location", $RLIV_2$><tornado, scale, location>"$RLIV_1$. id = $RLIV_2$ id"))

Figure 4.29. An example of integration script.



reportId	Fscale	location	county	state	lat	lon	time	I	J
1	EF-5	Phil-Campbell	Franklin	AL	34.3500	-87.7100	08:30:...		
2	EF-5	Moulton	Lawrence	AL	34.4800	-87.2900	08:55:...		
3	EF-5	Speake	Lawrence	AL	34.4200	-87.1700	07:00:...		
4	EF-5	Mt.Hope	Lawrence	AL	34.4600	-87.4800	08:48:...		
5	EF-5	Hatton	Lawrence	AL	34.5600	-87.4100	08:53:...		
6	EF-4	West-Point	Cullman	AL	34.2400	-86.9600	12:00:...		
7	EF-4	Good-Hope	Cullman	AL	34.1100	-86.8600	07:45:...		
8	EF-4	Hanceville	Cullman	AL	34.0600	-86.7600	01:01:...		
9	EF-4	Baileyton	Cullman	AL	34.2600	-86.6100	08:07:...		
10	EF-4	Joppa	Cullman	AL	34.3000	-86.5600	08:23:...		
11	EF-5	Madison	Madison	AL	34.7000	-86.7500	10:06:...		
12	EF-5	Huntsville	Madison	AL	34.7300	-86.5900	09:31:...		

Figure 4.30. A map results from execution of the integration script shown in Figure 4.29.

CHAPTER 5. EVALUATION

In this chapter, we look at the correctness of the spatial mediator model. Since people's definitions of quality are different it is not practical to try to prove anything about quality. Rather we have done on empirical study to address the issue of map quality (Section 5.1.1). Coverage is measureable so we looked at map coverage with a set of lemmas (Section 5.1.2). For evaluation of the relational portion of the spatial mediator we concentrate on showing that our query generation process generated queries with a lossless join property whenever the underlying data semantics supported a lossless join (Section 5.2).

5.1 Map generation process correctness

The discussion of the map generation process introduced in Section 4.3.5 highlights the need of ranking data sources in the infrastructure. We start by looking at an empirical study to evaluate the quality of the maps produced in Section 5.1.1. Section 5.1.2 looks at the correctness of the map coverage.

5.1.1 Empirical study

We look at the results from empirical study in this section. Section 5.1.1.2 looks at the empirical results for the Quality Ranking Measure Model. The empirical

results for the Scoring Function Ranking Model are examined in Section 5.1.1.3. We present correctness issues in Section 5.1.1.4.

5.1.1.1 Empirical study data set

A geographic dataset for the counties of State of Iowa was used as the test set. It contains geographic information about state, county and city. Three hundred and fifty five spatial objects (i.e. MLIVs) are used as the test dataset. We validate the approaches by creating twelve requests to conduct the testing.

5.1.1.2 Evaluation of quality ranking measure model

The spatial map mediator utilizes the quality measure v to evaluate the potential contribution of each MLIV to the generation of a useful map.

A MLIV ranking value v is defined as follows:

$v = w_1 * com + w_2 * file + w_3 * pos + w_4 * rel + w_5 * res + w_6 * access$, *where*

w_i *is the weight associated with* ^{i}*th parameter*

Our contributions come from the ways we determine the parameters and weights. We start by looking at the generation of parameter. We develop a model, Parameter Resolution Value Generation Model, to generate the parameter *res* because of its geographic characteristics. The result is shown in Table 5.1.

Table 5.1. Res values generated by the Parameter Resolution Value Generation Model.

MLIV \ Req	1 m	5 m	10 m	25 m	100 m
1 m	**1**	**0.7833**	**0.70749**	**0.64063**	**0.5034**
5 m	**0.7833**	**1**	**0.875**	**0.7833**	**0.56651**
10 m	**0.70749**	**0.875**	**1**	**0.8472**	**0.70749**
25 m	**0.64063**	**0.7833**	**0.8472**	**1**	**0.78906**
100 m	**0.5034**	**0.56651**	**0.70749**	**0.78906**	**1**

The first row lists attribute resolution of the coming request and the first column is the value associated with the MLIV. The values in bold font are values for the parameter *res* generated by Parameter Resolution Value Generation Model. The value for parameter *res* is 1 if both the MLIV and request has the same values for the attribute resolution. We found that there are several values are higher than we expected. They are values in the cell with solid color background. When the attribute resolution associated with MLIV is coarser than the one associated with the request the parameter *res* should be lower.

We proposed two approaches to calculate w_i, namely Partitioning Weight Generation Approach and Map Grouping Weight Generation Approach. The following sections described the evaluation of these two methods.

5.1.1.2.1 Evaluation of partitioning weight generation approach

In Section 5.3.2.1 we have described the process of Partitioning Weight Generation Approach for deriving the weight values in the quality ranking measure.

The set of MLIVs is partitioned into four sets representing the system's ability to do any conversion required to use an MLIV to process the request. We use the number of incorrectly partitioned MLIVs to demonstrate the effectiveness of this approach. The incorrectly partitioned MLIVs mean MLIVs that have been partitioned into wrong set. For example, a MLIVs that can be used in the response without conversion is partitioned to the set that contains MLIVs that can be used, but will suffer some loss of quality using the current conversion tools supported by the system.

The Figure 5.1 shows the comparison of error rates in terms of the number of incorrectly ranked MLIVs between two sets of weights, namely initial weights and tuned weights. The initial weights are set to 0.5 for all w_i in the quality ranking measure. After the tuning process the final tuned weights are generated. Twelve requests are put to evaluate the approach. Two out of twelve does not support the effectiveness of this approach, namely Request 2 and Request 4, since the number of incorrectly partitioned MLIVs before tuning process are lower than the one when tuned weights apply.

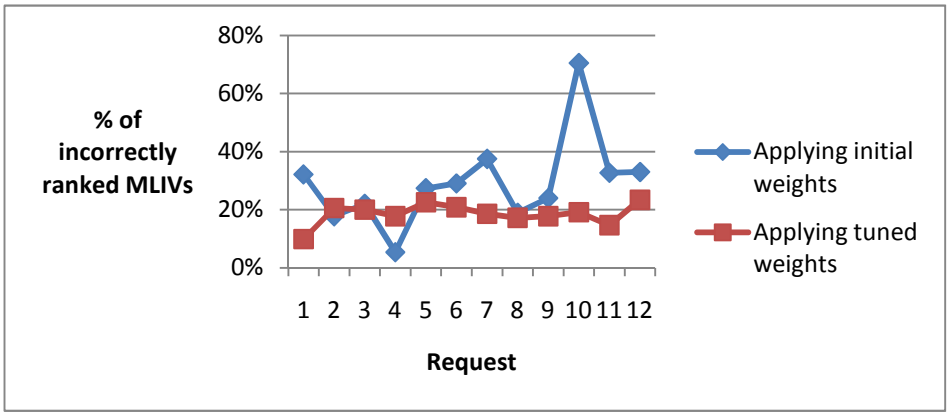


Figure 5.1. A comparison of incorrectly ranked MLIVs between sets of initial weights and tuned weights.

Figure 5.2 shows the comparison of incorrectly ranked map groupings between sets of initial weights and tuned weights. The result shows that one out of twelve requests, namely request 3, doesn't support this approach since the number of incorrectly ranked map groupings are the same for both sets of weights.

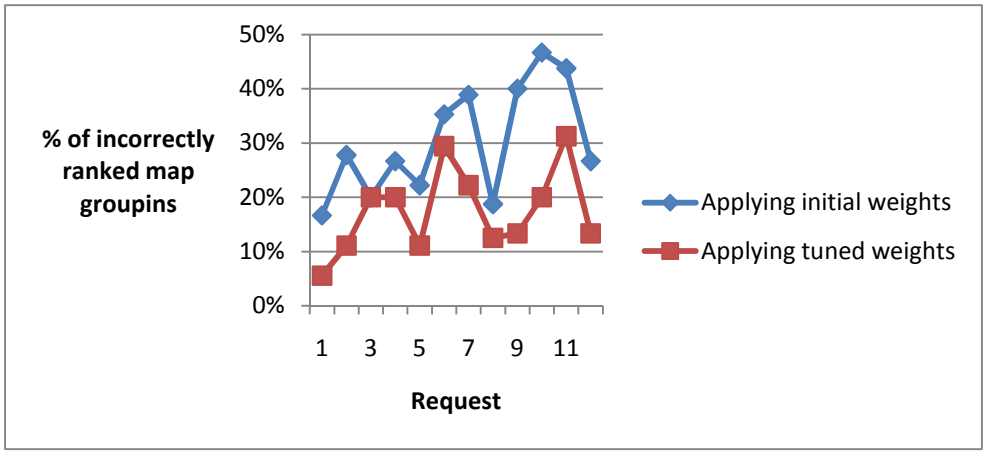


Figure 5.2. A comparison of incorrectly ranked map groupings between set of initial weights and tuned weights.

5.1.1.3 Evaluation of scoring function ranking model

To show the evaluation of this ranking model, three different sequences are used in the testing, namely *incoming sequence*, *random sequence* and *sorted sequence*.

The *incoming sequence* corresponds to the MLIV's identifier: $MLIV_{id}$. The MLIV with the smallest $MLIV_{id}$ will come in as the first element in the *incoming sequence*. In another words, the spatial mediator will select MLIV with the smallest $MLIV_{id}$ since it is on the top of the PartialCoverageLise. *Sorted sequence* is ordered by the score associated with each MLIV. The MLIV with highest score is on the top of the *sorted sequence* and will be selected by the spatial mediator first. All $MLIV_{id}$ are rendered into a random order and becomes the *random sequence*.

Figure 5.3 show the comparison of the performance in terms of incorrectly ranked MLIVs between these three sequences, namely sorted sequence, random sequence and incoming sequence. In this sorted sequence, no incorrectly ranked MLIV is found. Both incoming sequence and random sequence have some incorrectly ranked MLIVs. Except for one request, request 5, random sequence has more incorrectly ranked MIVs than the incoming sequence. Thus the sorted sequence is proved to be the best. In another words, the scoring function generates the correct ranking MLIVs with respect to the incoming map request.

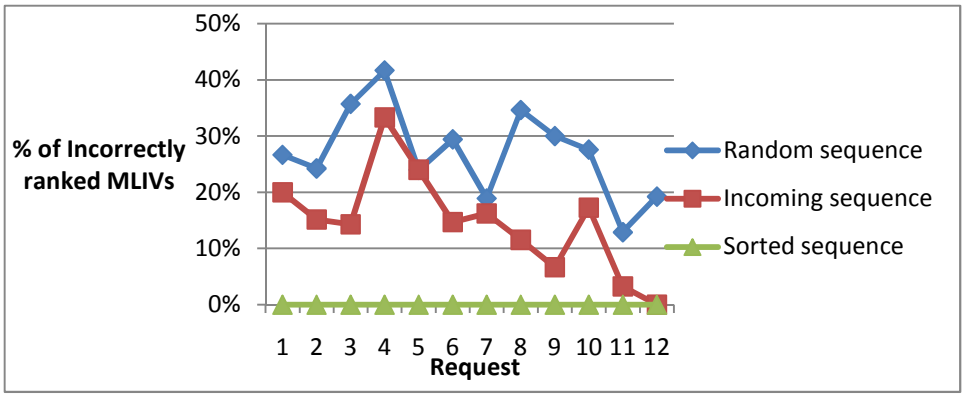


Figure 5.3. Number of incorrectly ranked MLIVs between random sequence, incoming sequence and sorted sequence.

Figure 5.4 shows the comparison of the performance in terms of the number of incorrectly ranked map groupings between these three sequences. In this sorted sequence, incorrectly ranked map groupings are found only for three requests, namely request 5, request 10 and request 11. Only four requests out of twelve, the incoming sequence has more number of incorrectly ranked map groupings than the random sequence. Thus the random sequence has the poorest performance among three sequences. The sorted sequence is proved to be the best.

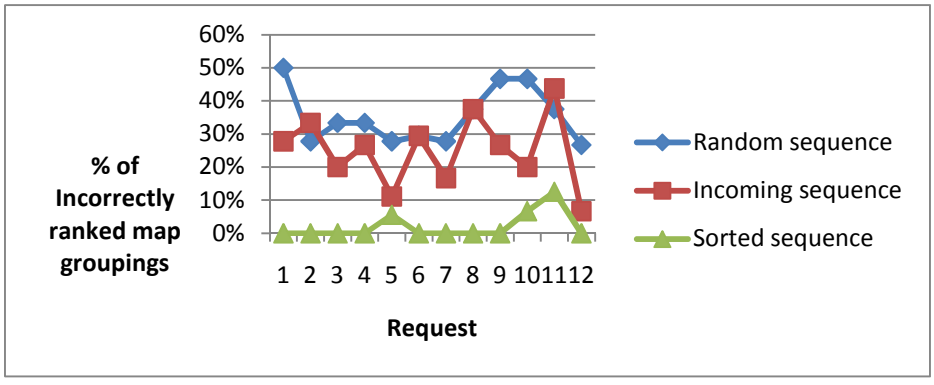


Figure 5.4. Number of incorrectly ranked map groupings between random sequence, incoming sequence and sorted sequence.

5.1.1.4 Quality evaluation of map grouping weight generation model

We present our performance evaluation of the map grouping weight generation model of the spatial mediator in our empirical study in this section. We use a quantity measure, equivalence class, to show the performance of our spatial mediator. Each parameter is assigned a value indicating the degree of quality in an equivalence class. We compare the median of the equivalence class of MLIVs of map grouping in the map script generated by the spatial mediator with the best available MLIVs for the corresponding request.

The tables of the values of the equivalence class of six parameters in the quality ranking measure model and tables of comparison values are shown in Appendix C. Figure 5.4 through Figure 5.9 show the Comparison between MLIVs in map script and best available MLIVs (as chosen by our research group) in terms of six parameters in Quality Ranking Measure Model, namely, *com, file, pos, rel, res and access*. From these comparisons we found out that MLIVs in map script generated by the spatial mediator are the best MLIVs available in more than half of the requests. For example, MLIVs in map script generated by the spatial mediator are the best available in nine out of twelve requests in terms of the parameter *res*. The worst case is the parameter *pos*, MLIVs in map script generated by the spatial mediator are the best available are only seen in six out of twelve requests.

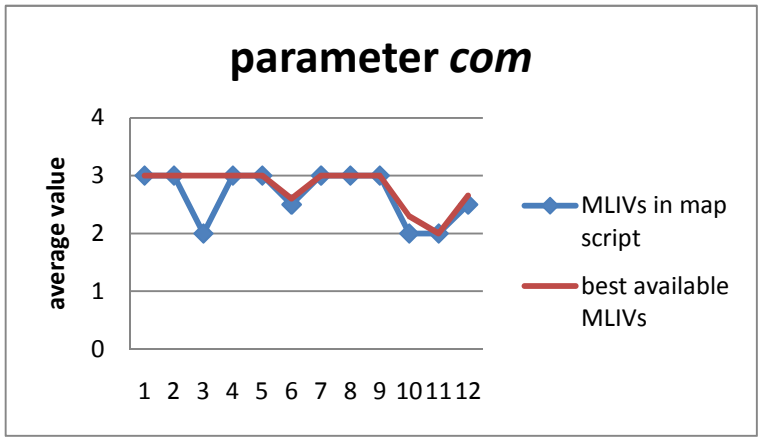


Figure 5.5. Comparison between MLIVs in map script and best available MLIVs in terms of parameter *com.*

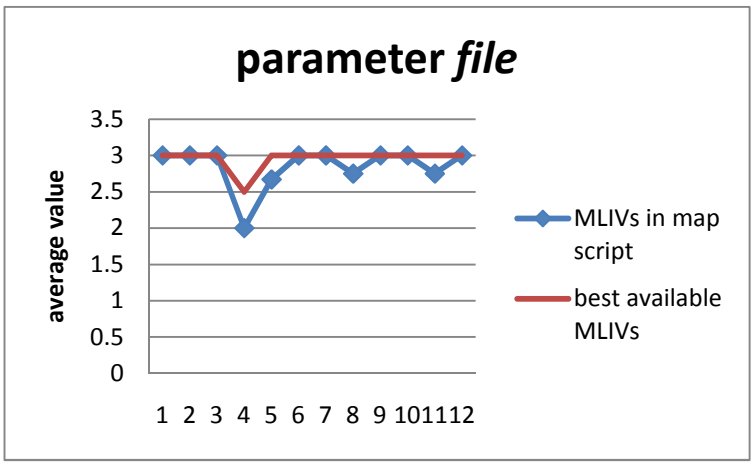


Figure 5.6. Comparison between MLIVs in map script and best available MLIVs in terms of parameter *file.*

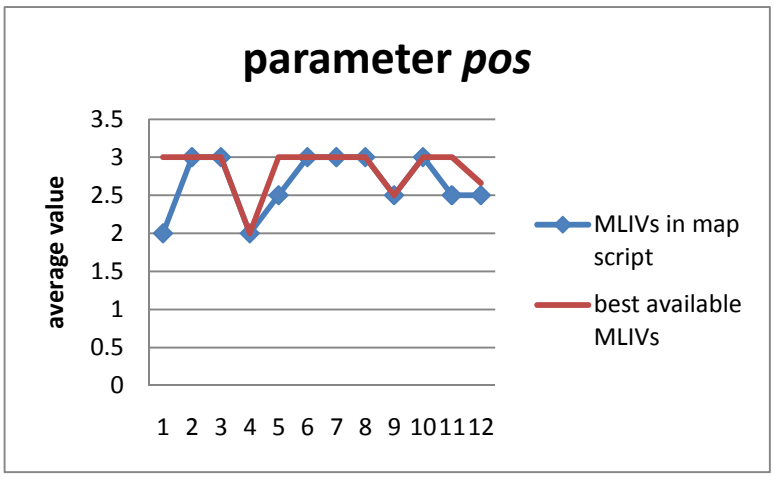


Figure 5.7. A comparison between MLIVs in map script and best available MLIVs in terms of parameter *pos.*

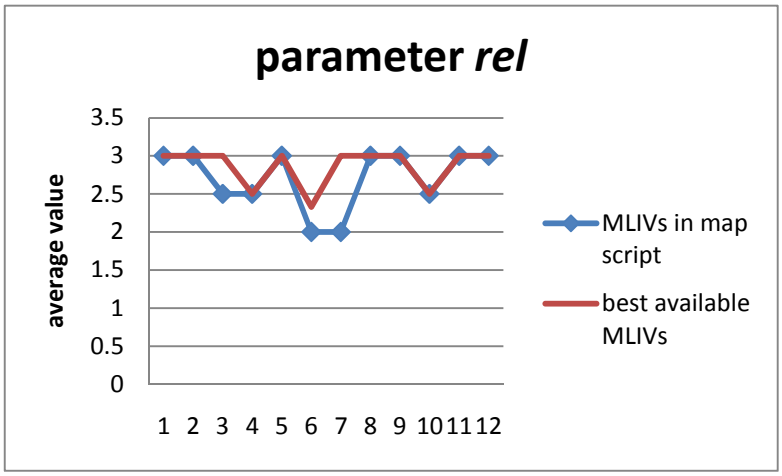


Figure 5.8. Comparison between MLIVs in map script and best available MLIVs in terms of parameter *rel*.

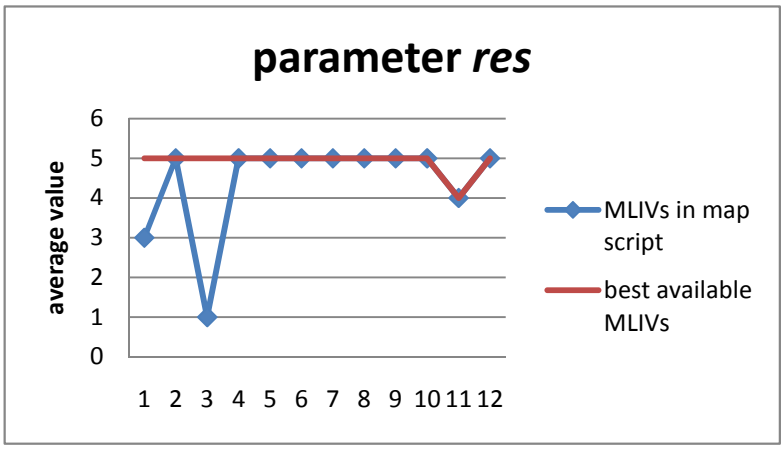


Figure 5.9. Comparison between MLIVs in map script and best available MLIVs in terms of parameter *res*.

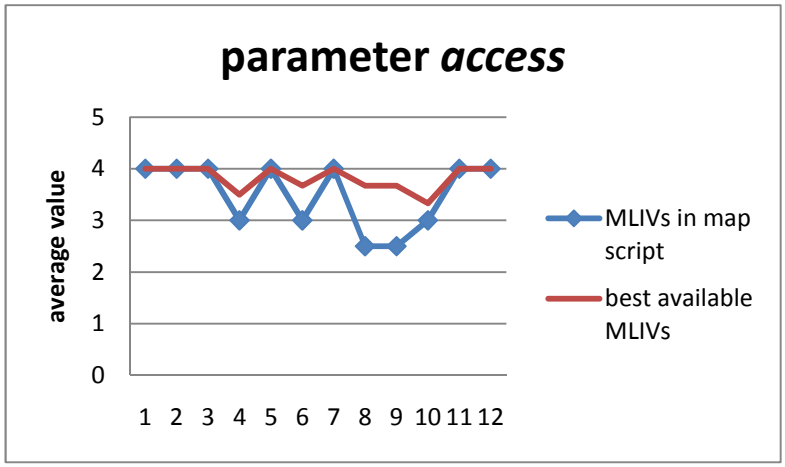


Figure 5.10. Comparison between MLIVs in map script and best available MLIVs in terms of parameter *access*.

5.1.2 Conceptual correctness

Let ℓ_g be the geographic search region (point, point & radius or bounding box) for the current request. We first define a *correct MLIV* identified by the R_Tree component (***RT***) to be an MLIV that either contains ℓ_g or overlaps ℓ_g. The MLIV that only share a common border with ℓ_g or touches ℓ_g isn't considered a *correct MLIV*.

The followings are examples of ℓ_g

1. ℓ_g represents a bounding box defined by the points *<41.5233, 93.1402>, <42.1341, 94.1435>* $\ell_g = \{<41.52, 93.14>,<42.13, 94.14>\}$
2. ℓ_g represents region with a center point at *<41.5211, 93.1463>* and a *50 m* radius $\ell_g = \{<41.5211, 93.1463>, 50m\}$
3. ℓ_g presents a point *<41.5211, 93.1463>* $\ell_g = \{<41.5211, 93.1463>\}$

We define a *correct MLIV* identified by the Geographic Symbolic Ontology (***GSO*)** to be an MLIV that either contains ℓ_s or overlaps ℓ_s where ℓ_s is the region defined by the symbolic terms in the request. The following is example of ℓ_s.

ℓ_s represents the region: "Midwest" $\ell_s = \{Midwest\}$

5.1.2.1 Correctness coverage using *RT*

Lemma 1: Given a map request **r** associated with the geographic location requirement ℓ_g, component ***RT*** identifies correct MLIVs if the requested area is covered in the complete set of MLIVs available.

Proof:

We prove this lemma by contradiction.

Assume that given the request ***r***, component ***RT*** identifies a MLIV that doesn't overlap nor contains at least part of geographic location requirement ℓ_g. The MLIV must be in one of the following cases.

Case 1: The ℓ_g is a point and falls outside the MLIV.

Case 2: The ℓ_g is point and radius that results in a circle shape region and doesn't overlap with the MLIV.

Case 3: The ℓ_g is a circle shape region that only contact with the MLIV at one point.

Case 4: The ℓ_g is a bounding box that shares a common border with the MLIV.

Case 5: The ℓ_g is a bounding box that doesn't overlap with the MLIV.

For Case 1, the function in ***RT,*** *Include(Point, Polygon)* will be applied. According the definition of this function (which is described in Section 4.2.2) it returns TRUE if the point meets the following two conditions: (a) the point is located

inside the polygon and (b) the point doesn't touch the borders of the polygon. Since ℓ_g is a point and falls outside the MLIV in Case 1 the value FALSE will be returned by this function. An example of this case is shown in Figure 5.11.

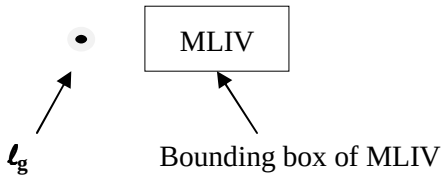
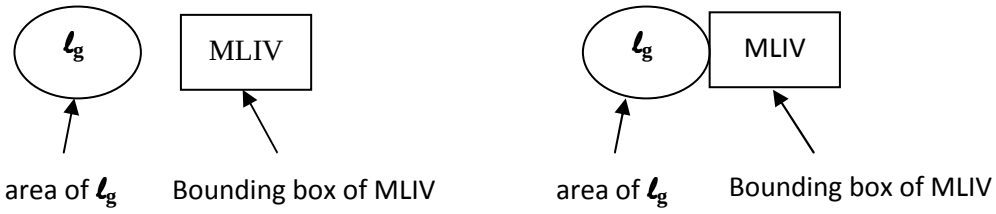


Figure 5.11. An example of the topological relationship between a point location requirement and the bounding box of MLIV.

For Case 2 and 3, the function in ***RT,*** *Overlap(Polygon, Circle)* will be applied. According the definition of this function (which is described in Section 4.2.2) it returns TRUE if the circle and the polygon overlapped. In Case 2 shown in Figure 5.12(a), ℓ_g is a circle and doesn't overlap with MLIV the function returns FALSE. ℓ_g is a circle and only contact with the MLIV at one point the function returns FALSE in Case 3 which is shown in Figure 5.12(b) shows an example of this case.



(a) Case 2 (b) Case 3

Figure 5.12. Examples of possible relationship between a circle region location requirement of request and the bounding box of MLIV.

For Case 4 and 5, the function in ***RT***, *Overlap(Polygon, Polygon)* will be applied. According the definition of this function (which is described in Section 4.2.2) it returns TRUE if these two polygons overlapped. An example of Case 4 is shown in Figure 5.13(a), the bounding box of ℓ_g shares a common border with the bounding box of MLIV and the function returns FALSE. An example of Case 5 is shown in Figure 5.13(b), the bounding box of ℓ_g doesn't overlap with the bounding box of MLIV and the function returns FALSE.

(a) Case 4 (b) Case 5

Figure 5.13. Examples of possible topological relationship between the bounding box of location requirement of request and the bounding box of MLIV.

The functions of component ***RT*** , namely *Overlap(Polygon, Polygon), Overlap(Polygon, Circle), Include(Point,Polygon),*will not return TRUE for any case from Case 1 through Case 5 stated above.

According to the definition of component ***RT,*** only when at least one of the functions returns TRUE for an MLIV that the component ***RT*** identifies it as a correct MLIV. We found the contradiction for the assumption and thus obtain the following

conclusion: given a request **r** associated with location requirement ℓ_g, component ***RT*** identifies correct MLIVs.

Figure 5.14 shows examples that the area or the bounding box of location requirement ℓ_g overlaps with the bounding box of MLIV. In such cases, MLIV will be identified by component ***RT*** and thus a correct MLIV. In (a), the location requirement ℓ_g is a circle that overlap with the bounding box of MLIV. In (b) the location requirement ℓ_g is a bounding box that overlaps with the bounding box of MLIV. In (c) the location requirement ℓ_g is a point that falls inside the bounding box of MLIV. □

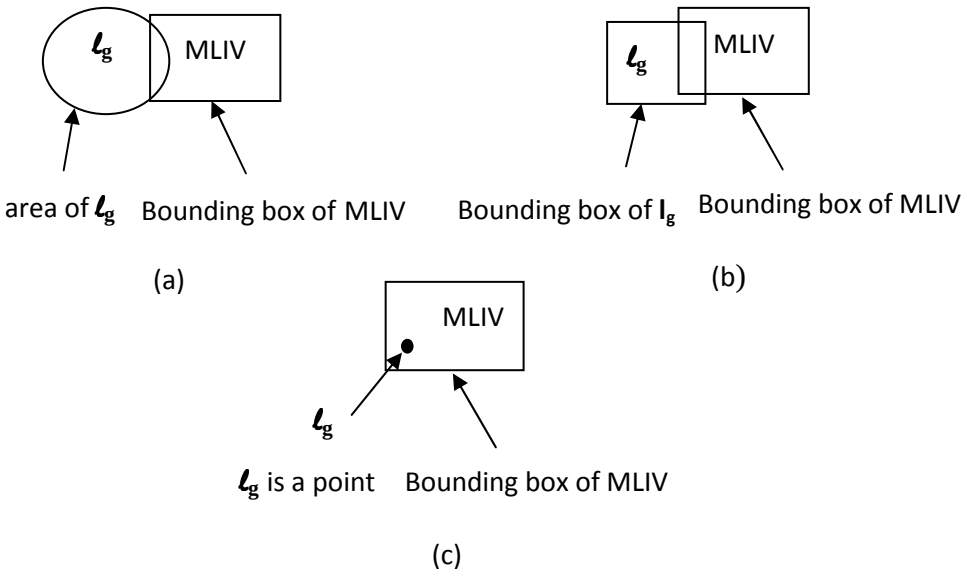


Figure 5.14. examples of possible relationship between the location requirement of request and the bounding box of MLIV.

The next lemma looks at the correctness of map coverage using the ***GSO*** search.

5.1.2.2 Correctness of coverage using *GSO* :

Lemma 2: Given a map request **r** associated with symbolic location requirement ℓ_s, ***GSO*** identifies MLIVs that correctly cover the requested area.

Proof:

We prove the Lemma 2 by contradiction.

Assume a request ***r*** with a symbolic location requirement ℓ_s, component ***GSO*** identifies MLIVs that neither contains the requested location nor overlaps part of location.

Based on the assumption, for the ***GSO*** to identify the MLIV there must exit a path starting from the root to the MLIV. According to the definition of ***GSO*** an edge "include" must be included in the path for a MLIV to be identified. It means that the MLIV must be included in the region that triggers the search. A contradiction to the assumption occurs. So we conclude that given a request ***r*** associated with symbolic location requirement ℓ_s, ***GSO*** identifies correct MLIVs that correctly cover the request.

□

5.1.2.3 Correctness of coverage of map grouping

Lemma 3: Given a map request **r** and the set of MLIVs L, where L is the set of MLIVs in *PartialCoverageList* produced by the ***RT*** and ***GSO*** components. If the

map grouping algorithm terminates with success the map grouping algorithm generates a map grouping that covers the location requirement $\boldsymbol{\ell}$ of **r**.

Proof:

Given request $\boldsymbol{r} = < rid, \boldsymbol{\ell}, \boldsymbol{\Omega}>$ assume that map grouping algorithm generates MG_i. Let *space* (***B***) returns the space covered by ***B*** where ***B*** is a geographical polygon. We are proving the following:

$space(\boldsymbol{\ell}) \subseteq space(\boldsymbol{ML})$ where ***ML*** is the union of bounding boxes of MLIVs in MG_i

After the grouping process starts and before it terminates, the region $\boldsymbol{\ell}$ is divided into two parts: ℓ_α and ℓ_β where ℓ_α is the region that hasn't covered by any MLIV in MG_j and ℓ_β is the region covered by MLIVs in MG_j where $MG_j \subseteq MG_i$ and $space(\ell_\alpha)$ $space(\ell_\beta) = space(\boldsymbol{\ell})$. An example is shown in Figure 5.15.

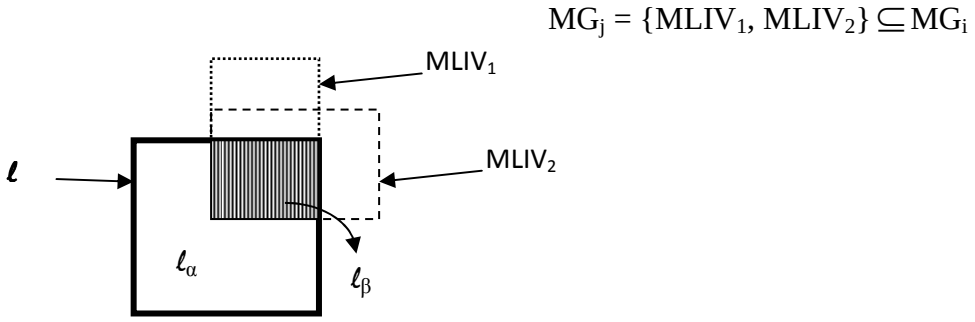


Figure 5.15. The bounding box of map request is covered by a subset of a map grouping MG_j that contains two MLIVs, namely $MLIV_1$, $MLIV_2$.

There are only two possibilities for the grouping algorithm to be terminated, namely, return a map grouping successfully and return a message indicating fail to find a map grouping.

If the algorithm returns a map grouping then it indicates the algorithm terminates successfully i.e., it will identifies all MLIVs in $\boldsymbol{L}$ that overlaps with ℓ_α and discard $MLIV_j$ where the bouonding box of $MLIV_j$ is covered by $space(\ell_\beta)$. -----(a)

The condition for the algorithm to terminate successfully is $space(\ell_\alpha) = \emptyset$ -----(b)

From (a) and (b) we conclude that map grouping algorithm generates a map grouping: MGi which contains $MLIV_\square$, where $1 \leq i \leq n$ and $space(\boldsymbol{\ell}) \subseteq space(\boldsymbol{ML})$,where $\boldsymbol{ML}$ is the union of bounding boxes of MLIVs in $MG_{i.}$ In another words, the total coverage of MLIVs in MGi covers the region of $\boldsymbol{\ell}$ in request $\boldsymbol{r}$. □

Theorem 1:

Given a map request, when the map grouping algorithm terminates successfully the spatial mediator generates a map script whose bounding box covers the bounding box of the location requirement of the map request by using the components ***RT*** and ***GSO***.

Proof:

In Lemma 1 we prove that the ***RT*** identifies MLIVs that overlap or contain the bounding box of the geographic location requirement of the map request. In Lemma 2 we prove that the ***GSO*** identifies MLIVs that overlap or contain the bounding box of the symbolic location requirement of the map request. In Lemma 3 we prove that if the map grouping algorithm terminates with success the map grouping algorithm generates a map grouping that covers the location requirement of the map request. □

5.2 Relational query correctness

For the issue of relational correctness, we make use of the lossless join where possible. Note that it is possible for people to register data sources that are not lossless. To this point the individual registering an RLIV whether the relations in the LIVs satisfy the universal relation principle. The following proofs of correctness are only valid in the instances where the registered RLIVs support this assumption.

Two aspects of the process of generating a relational result have to be considered. Section 5.2.1 looks at the correctness of the process for generating the framework query and Section 5.2.2 exams correctness for the subqueries needed for each RLIV.

5.2.1. Framework query correctness

We start by looking at the required definitions in relational database theory. If U is the set of attributes, then a database scheme $\mathscr{R} = (R_1, R_2,\ldots, R_k)$ is defined as a set of subsets of U. We use $r(R_i)$ is to be a relation instance over scheme R_i.

Let $\mathscr{R} = (R_1, R_2,\ldots, R_k)$ be a set of relation schemes over U. A relation r(U) satisfies the *join dependency* (jd)$\bowtie$[$R_1, R_2,\ldots, R_k$], if and only if $r = \pi_{R1}(r)\bowtie\ldots\bowtie \pi_{Rk}(r)$.

The relation u(U) is called the *universal relation* if U is the set of all attributes. A set of relations over U defined by $\mathscr{R} = (R_1, R_2,\ldots, R_k)$ is said to satisfy the *universal relation principle* if the join dependency $\bowtie$ [$\mathscr{R}$] holds for the universal relation u(U).

We defined *attr*(R) for a relation scheme R as the set of attributes of R and *attr*(RLIV) is defined as the union of sets of attributes of relations inside the RLIV.

Two representation of a set of relations are *join equivalent* if they produce an equivalent join result.

We use the *renaming process* used by Miller et al. (2002) to insure that attribute names are unique. The process uses attribute names in views to insure that attributes that have the same semantics have the same name and that attributes with different semantics have different names.

We briefly review some of the relevant terminology concerning the relationship between database and hypergraphs in the following paragraph. A hypergraph is a couple $H = (N, E)$, where N is the set of vertices and E is a set of hyperedges which are nonempty subsets of N. We define $attr(E)$ is the union of the edges in E (Berge 1989). A hypergraph is reduced if no hyperedge of H is properly contained in another hyperedge of H. H is connected if every pair of its vertices is connected by some path of hyperedges. If H is reduced connected hypergraph with the vertex set N and the edge set E, then E' is a *complete subset* of E if and only if $E' \subset E$ and for each E_i in E if $E_i \subseteq attr(E')$ then E_i belongs to E'. E' is said to be a *trivial subset* of E if $|E'| \leq 1$ or $E = E'$.

Let E' be a nontrivial complete subset of E and $\psi_1, \psi_{2,\ldots,}\psi_p$ be connected components of $E - E'$ with respect to E'. Then E' has the *bridge-property* if and only if for every i = 1,2,…,p there exists $E_i \in E'$ such that $(attr(E') \cap N_i) \subseteq E_i$, where $N_i = attr(\psi_i)$. E_i is called a *separating edge* of E' corresponding to ψ_i. A nontrivial complete subset E' of E with the *bridge property* is called a *hinge* of H (Gyssens & Paredaens 1984).

A set of RLIVs in an ***SNO*** is defined as a set of *connected RLIVs* if for any two RLIVs in the set they are connected by some path. A path in an ***SNO*** is a sequence of the form $RLIV_1$, $Association_1$, $RLIV_2$,…$Association_{m-1}$, $RLIV_m$, where

Association $_i$ is connected to $RLIV_i$ and $RLIV_{i+1}$. We have defined *framework query* in Section 4.4 as a query that has the form: $R_1\ join_1\ R_2\ join_2 \ldots join_{m-1}\ R_m$ Where R_i, i = 1,2,…,m-1 are RLIVs connected in the semantic network of the ***SNO*** with a join criteria ($join_i$, i=1,2,…,m-1) (equijoin in the current model)defined in the association node that connects the two RLIVs.

Lemma 4: Let $\mathcal{L}$ be a set of *connected RLIVs* in an ***SNO***, where the joins defined by the association in the ***SNO*** for the RLIVs in $\mathcal{L}$ are equijoins. A connected hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{E})$ exits, such that, the set of relations defined by the relations schemes in $\mathcal{E}$ is *join equivalent* to the set of RLIVs in $\mathcal{L}$.

Proof:

Without loss of generality, let $\mathcal{L} = \{L_1, L_2, \ldots L_m\}$. Create the set $\mathcal{L}' = \{L'_1, L'_2, \ldots, L'_m\}$ where $\mathcal{L}'$ is created from $\mathcal{L}$ by applying the *renaming process* given by (Miller et al. 2002).

Let $\mathcal{E}' = \{attr(L'_1), attr(L'_2), \ldots attr(L'_m)\}$. Construct $\mathcal{E} = \{E_1, E_2, \ldots E_m\}$ from $\mathcal{E}'$ by using $E_i = \mathbf{sub}(attr(L'_i))$ $(1 \le i \le m)$ where **sub**( ) is an operation that replaces any attribute names equated in equijoin statements with a unique name.

Let $\mathcal{N} = \bigcup_{i=1}^{m} attr(E_i)$. We now need to show that the resulting hypergraph $\mathcal{H} = (\mathcal{N}, \mathcal{E})$ is *join equivalent* to the set of RLIVs in $\mathcal{L}$.

[To show: Equijoin of RLIVs in $\mathcal{L}$ implies ($\rightarrow$) $\bowtie_{i=1}^{m} e_i(\mathbf{E_i})$, where e_i is a relation over Scheme $\mathbf{E_i}$].

Let *t* be a tuple in the relation u($\mathcal{U}$) formed by joining the tuples in the $\mathbf{L_i}$ of $\mathcal{L}$ ($1 \leq i \leq m$) under the equijoin conditions. Since the process of converting the scheme of $\mathbf{L_i}$ to $\mathbf{E_i}$ ($1 \leq i \leq m$) has only changed the attributes names and eliminated columns that are equal to columns under the new names, we can find a tuple δ generated in $\bowtie_{i=1}^{m} e_i(\mathbf{E_i})$ that has the same values as *t* only with the duplicated columns missing. $\therefore$ Equijoin of RLIVs in $\mathcal{L}$ with the duplicated columns removed is a subset of $\bowtie_{i=1}^{m} e_i(\mathbf{E_i})$.

[To show: $\bowtie_{i=1}^{m} e_i(\mathbf{E_i})$ implies the equijoin of RLIVs in $\mathcal{L}$]

Let *s* be a tuple in the relation $\bowtie_{i=1}^{m} e_i(\mathbf{E_i})$, Again since we haven't changed relational values in the e_i ($1 \leq i \leq m$) and only the attribute names, we can find a tuple *t* in the equijoin of the $\mathbf{L_i}$ of $\mathcal{L}$ that has the same values plus some duplicated columns.

$\therefore \bowtie_{i=1}^{m} e_i(\mathbf{E_i})$ with the duplicated columns added is a subset of the equijoin of the RLIVs in $\mathcal{L}$. Hence the set of connected RLIVs is join equivalent to the edges of the hypergraph. □

After the renaming process, the attribute names in the equijoin statement have been changed and the columns under the names have also been eliminated. This

guarantees no duplicate attribute names or columns after the join operations. Note this lemma holds whether or not the join is a lossless join.

Figure 5.16 shows an example of a set of RLIVs, namely {$RLIV_1$, $RLIV_2$} that is *join equivalent* to the set of relations: {R_1,R_2}. The join criteria in the ***SNO*** fragment says: "$RLIV_1$.Household-size = $RLIV_2$.Capacity". The *renaming process* renames both "Household-size" and "Capacity" as "Num_of_Person". The resulting tuples from the join operation from these two sets are join equivalent.

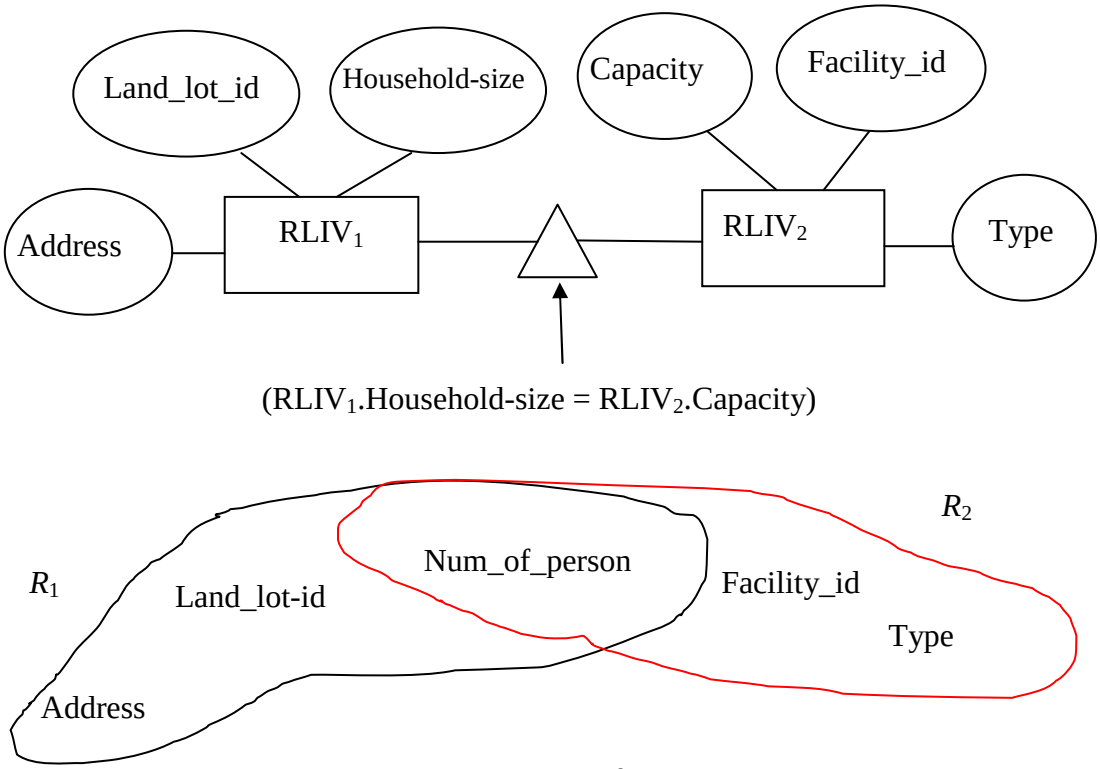


Figure 5.16. An example of join equivalent sets.

Theorem 2: The framework query algorithm generates framework queries with a lossless join sequence, if the set of RLIVs in the join sequence are connected,

the joins in the ***SNO*** are equijoin, the subqueries for the individual RLIVs are lossless, and the connected components of the semantic network of RLIVs satisfy the universal relation principle.

Proof:

Let $\mathcal{L}$ be the set of RLIVs from a framework query, where $\mathcal{L}$ is a subgraph of a connected component $\mathcal{G}$ of the semantic network of RLIVs that satisfies the universal relation principle.

Construct two hypergraphs $\mathcal{H}_{\mathcal{L}}$ and $\mathcal{H}_{\mathcal{G}}$ using the process described in the proof of Lemma 4, where $\mathcal{H}_{\mathcal{L}}$ represents the RLIVs in $\mathcal{L}$ and $\mathcal{H}_{\mathcal{G}}$ represents the RLIVs in $\mathcal{G}$, respectively.

Without loss of generality, let $\mathcal{H}_{\mathcal{L}} = (\mathcal{N},\mathcal{E})$, $\mathcal{E} = \{\mathrm{E}_1, \mathrm{E}_2,\ldots\mathrm{E}_m\}$ and $\mathcal{H}_{\mathcal{G}} = (\mathcal{O},\mathcal{D})$, $\mathcal{D} = \{\ \mathrm{D}_1, \mathrm{D}_2,\ldots\mathrm{D}_k\}$. Note that $\mathcal{E} \subseteq\ \mathcal{D}$.

It is clear from the framework query algorithm that any connected component of $\mathcal{H}_{\mathcal{G}}$ that is not part of $\mathcal{H}_{\mathcal{L}}$ is connected to only one edge of $\mathcal{H}_{\mathcal{L}}$.

$\therefore$ $\mathcal{H}_{\mathcal{L}}$ is a hinge and the join of the relations represented by the edges in $\mathcal{E}$ is lossless.

Moreover, by Lemma 5.3, the RLIVs in $\mathcal{L}$ are *join equivalent* to those defined by $\mathcal{H}_{\mathcal{L}}$. Hence the join of the RLIVs in $\mathcal{L}$ are lossless. □

5.2.2. Subquery correctness

We start by looking at the definition of the *F-fd-hinge* defined by (Miller 1992). Let $H = (U, \mathcal{R})$ represent a universal join dependency. Let Σ be an arbitrary subset of $\mathcal{R}$ and *F* be a set of functional dependencies. From Σ and F, we generate a set of edges $\Sigma^* = \{\mathbf{E}_1^*,\ldots,\mathbf{E}_n^*\}$by expanding the edges of Σ using the functional dependencies in F. Specifically, $\mathbf{E}_i^* = \mathbf{E}_i \cup W_1 \cup \cdots \cup W_m$ where $\mathbf{E}_i \cap \mathbf{E}_j \rightarrow W_j \in F^+$ for j = 1,…,n. We use the notation $H^*_{\Sigma, F}$ to denote the hypergraph with nodes U and edges $\mathcal{R} - \Sigma \cup \Sigma^*$. We say Σ is an *F-fd-hinge* of *H* if Σ^* is a hinge of $H^*_{\Sigma,F}$. We will use *Fd-hinge* in the remainder of this chapter where it is clear what set of functional dependencies is being used.

Theorem 3 looks at the correctness of the subquery generation algorithm with respect to generating lossless join sequences.

Theorem 3: The subquery algorithm generates a subquery with a lossless join property for a RLIV, if the relations in the RLIV satisfy the universal relation principle.

Proof:

Let $\mathcal{R}$ be the set of relations in an RLIV that satisfies the universal relation principle. By definition of the universal relation principle $\bowtie [\mathcal{R}]$ holds. Let $\mathcal{D}$ be the set of Fds (functional dependency) defined over the attributes in *attr*($\mathcal{R}$).

Construct the hyper graph $\mathcal{H}$ = (*attr*($\mathcal{R}$), $\mathcal{R}$) to represent the set of relations in the RLIV. Let $\mathcal{S} \subseteq \mathcal{R}$ be the set of relations used in the join sequence of the query generated by the subquery algorithm and let $\mathcal{C} = \mathcal{R} - \mathcal{S}$.

Without loss of generality, let $\mathcal{S} = \{S_1,\ldots,S_m\}$. The algorithm expands each edge $\mathcal{S}$ of using the fds in $\mathcal{D}$. In particular, at each step of the addEdges(G,$\mathcal{S}$, *F*) if $S_i \cap S_j \rightarrow W$ exists in $\mathcal{D}$, the attributes in W are added to both S_i and S_i. The process continues until the expansion is complete. At this point, we have generated the hypergraph $H^*_{\mathcal{S},\mathcal{D}}$. Since the algorithm doesn't stop until every connected component of G intersects with only one edge of $H^*_{\mathcal{S},\mathcal{D}}$, $\mathcal{S}$ is an $\mathcal{D}$ –fd-hinge.

Since $\bowtie[\mathcal{R}]$ holds and $\mathcal{S}$' is an *Fd-Hinge* of $\mathcal{R}$, $\bowtie[\mathcal{S}]$ holds by the result of Miller (1992). □

Theorem 3 shows that if the relations for a given RLIV satisfy the universal relation principle, then the subquery generated for the RLIV will have a lossless join. Meanwhile, Theorem 2 shows that if subqueries possess a lossless join sequence and the set of RLIVs have the potential to generate a lossless join the framework query algorithm will generate a lossless join. The result is that the two algorithms will generate a lossless join whenever possible.

Conclusions and thoughts on future work are presented in the next chapter.

CHAPTER 6. CONCLUSION AND FUTURE WORK

In this dissertation, we have presented a spatial mediator model to support the integration of heterogeneous data in the context of the geographic data domain. We conclude with a brief description of our model in this chapter. We also discuss our future research work that will expand on the model we presented here.

6.1 Conclusion

The key part of this research has been the development of a spatial mediator for the GeoGrid environment. The spatial mediator has been designed to dynamically respond to requests for geographic maps and related spatial data in a relational format. We have developed the spatial mediator to support data source selection, creation of geographic data manipulation operations and construction of query parameter vector sets used in the LIVs of the chosen data sources. There are three types of data the spatial mediator can return, namely a map, a set of spatial data and an integrated spatial object merged with a map and set of spatial data. The correctness of the model was evaluated.

We now conclude our work with the contributions of our model:

1. We define a weighted ontology search model to help user application designers use domain terms to access data from multiple heterogeneous spatial data sources.
2. We combine the weighted ontology with the semantic data model to further release the burden of dealing with system heterogeneity from users/applications.
3. We propose algorithms that allow the map mediator to examine the quality of spatial objects that can fulfill requests from users/applications. Our contribution comes from ways in which we determine the parameters and the weights in the quality measure.

6.2 Future work

Our spatial mediator is supported by an ontology based search module. We introduce a weighted ontology that allows users to access data by domain query. Future work will include the function ontology in our spatial mediator model. This function ontology can enrich a search by providing necessary information and more search terms. Another future consideration for using the weighted ontology in the spatial mediator will be to add questions to the registration process about the nature of the way a user application will use concepts. By doing that, the weights will be able to reflect the application needs more closely than our current static model can.

Finally, we will look at relaxing the restriction that the nonmap data is relational. In particular we will look at the use of the ***SNO*** structure for object and semi-structured data in addition to relational data.

APPENDIX A. Summary of registration data

Table a.1. Registration data for user node

User node Id	Id of the user
Owner	Information of the owner
Date	Registration date
Application type	The application type running on the user node
Device type	Type of device, e.g. mobile device or note pc
Screen code	This field specifies the type of request and the capability of the device display. Different values indicates different capabilities: 1. is a map request without panning functionality 2. is a map request with panning functionality 3. is relation request 4. is a merged request with panning functionality
Device Display Size	Size of the display
Preference Selection	Select category of quality attribute which is critical, important, non-important

Table a.2. Registration data for MLIV

MLIV Id		Id of the MLIV
Data Source ID		Id of the data source that provides this MLIV
Date		Registration date
BBX		Bounding box
Theme		Theme of the map
Symbolic term		An identifying term
Geographic Quality	completeness	Attribute completeness
	resolution	Attribute resolution
	mapType	Attribute mapType
	Positional Accuracy	Attribute Positional Accuracy
	reliability	Reliability
	accesssibility	

Table a.3. Registration data for RLIV

RLIV Id	Id of RLIV
Date	Registration date
Relation Scheme	The relation schemes of all relations inside the RLIV, include relation names, attributes names, attribute types, key attributes.
Join attribute	The attributes in the join condition with other RLIVs and the joining RLIV's name

Table a.4. Registration data for data source

DS id	Id of the data sources
Owner	Information of owner
Date	Registration date
LIVs provided	List of MLIVs and/or RLIVs provided
Reliability	The degree of the reliability of the node

Table a.5. Registration data for tool node

Tool node Id	Id of the tool node
Owner	Information of the owner
Date	Registration date
Tool Type	Type of the tool,
Tool Name	Name of the tool, e.g. mosaic, crop
IP address	IP address of the tool node

APPENDIX B. Rule set

Rule Sets:

- The first part of the rule set is used in the quality ranking model.
- The second part of the rule set is used in the scoring function model.

Each rule ***Rule*** $_{ij}$ corresponds to the j^{th} rule for each parameter i in the ranking function. Some of the forms use simpler versions. The values x, y and z are values in the tables.

Rules for parameter completeness, *com*:

$Rule_{10}$: IF R.com <= L.com THEN *com* = 1.0;
$Rule_{11}$: IF R.com = x and L.com = y THEN *com* = z;

Parameter *file* has the following pre-existing rules:

IF fileType = JPG THEN mapType = lossyRaster;
IF fileType = TIFF THEN mapType = raster;
IF fileType = GeoTIFF THEN mapType = raster;
IF fileType = DEM THEN mapType = raster;
IF fileType = DLG THEN mapType = vector;
IF fileType = VPE THEN mapType = vector;
IF fileType = SHAPE THEN mapType = vector;

Rules for parameter file, *file*:

$Rule_{20}$: IF R.file = L.file THEN *file* = 1.0;
$Rule_{21}$: IF R.file= x and L.file = yTHEN *file* = z;

Rules for parameter positional accuracy, *pos*:

$Rule_{30}$: IF R.pos => L.pos THEN *com* = 1.0;
$Rule_{31}$: IF R.pos = x and L.pos = y THEN *pos* = z;

Rules for parameter reliability, *rel*:

$Rule_{50}$: IF R.rel =< L.rel THEN *rel* = 1.0;
$Rule_{51}$: IF R.rel = x and L.rel = y THEN *rel* = z;

Rules for parameter resolution, *res*:

$Rule_{60}$: IF R.res = L.res THEN *res* = 1.0;

$Rule_{61}$: IF R.res = x and L.res = y THEN *res* = z;

Rules for parameter resolution, *access*:

$Rule_{70}$: IF R.access = L.access = THEN *access* = 1.0;

$Rule_{71}$: IF R.access = x and L.access = y THEN *access* = z;

Rules for scoring function model:

$Rule_{80}$: if *req.compleness* = i and *MLIV.completeness* = j then S_i^{res} = b * m;

$Rule_{90}$: if *req.mapType* = i and *MLIV.mapType* = j then S_i^{file} = b* m ;.

$Rule_{100}$: if *req.positionalAccuracy* = i and *MLIV.positionalAccuracy* = j then S_i^{pos} = b *m;

$Rule_{200}$: if *req.reliability* = i and *MLIV.reliability* =j then S_i^{rel} = b * m ;.

$Rule_{300}$: if *req.resolution* = i and *MLIV.resolution* = j then S_i^{res} = b * m;

$Rule_{400}$: if *req.accessibility* = i and *MLIV.accessibility* = j then S_i^{access} = b * m;

APPENDIX C. Equivalence class comparison for the parameters

Table c.1 equivalence class comparison for the parameter *com*

parameter ***com***	value
1	3
0.9	2
0.8	1

Table c.2 equivalence class comparison for the parameter *com*

	parameter ***com***	
	MLIVs in map script	best available MLIVs
request 1	3	3
request 2	3	3
request 3	2	3
request 4	3	3
request 5	3	3
request 6	2.5	2.6
request 7	3	3
request 8	3	3
request 9	3	3
request 10	2	2.3
request 11	2	2
request 12	2.5	2.66

Table c.3 equivalence class comparison for the parameter *pos*

parameter ***pos***	value
0.01	3
0.02	2
0.03	1

Table c.4 equivalence class comparison for the parameter *pos*

	parameter ***pos***	
	MLIVs in map script	best available MLIVs
request 1	2	3
request 2	3	3
request 3	3	3
request 4	2	2
request 5	2.5	3
request 6	3	3
request 7	3	3
request 8	3	3
request 9	2.5	2.5
request 10	3	3
request 11	2.5	3
request 12	2.5	2.66

Table c.5 equivalence class comparison for the parameter *rel*

parameter ***rel***	value
1	3
0.9	2
0.8	1

Table c.6 equivalence class comparison for the parameter *res*

parameter ***res***	value
1	5
5	4
10	3
25	2
100	1

Table c.7 equivalence class comparison for the parameter *rel*

	parameter ***rel***	
	MLIVs in map script	best available MLIVs
request 1	3	3
request 2	3	3
request 3	2.5	3
request 4	2.5	2.5
request 5	3	3
request 6	2	2.33
request 7	2	3
request 8	3	3
request 9	3	3
request 10	2.5	2.5
request 11	3	3
request 12	3	3

Table c.8 equivalence class comparison for the parameter *res*

	parameter ***res***	
	MLIVs in map script	best available MLIVs
request 1	3	5
request 2	5	5
request 3	1	5
request 4	5	5
request 5	5	5
request 6	5	5
request 7	5	5
request 8	5	5
request 9	5	5
request 10	5	5
request 11	4	4
request 12	5	5

Table c.9 equivalence class comparison for the parameter *access*

	parameter ***access***	
	MLIVs in map script	best available MLIVs
request 1	4	4
request 2	4	4
request 3	4	4
request 4	3	3.5
request 5	4	4
request 6	3	3.67
request 7	4	4
request 8	2.5	3.67
request 9	2.5	3.67
request 10	3	3.33
request 11	4	4
request 12	4	4

Table c.10 equivalence class comparison for the parameter *access*

parameter ***access***	value
5	4
10	3
25	2
45	1

BIBLIOGRAPHY

Athanasiadis, I. N. and Janssen, S. (2008). Semantic mediation for environmental model components integration. *Information Technologies in Environmental Engineering*, 1, 3-11.

Baglioni, M., Masserotti, V., Renso, C., and Spinsanti, L. (2007). Building geospatial ontologies from geographical databases. *In Proceeding of the 2nd International Conference, GeoS* 2007, Mexico City, Mexico, 195-209.

Berge, C. (1973). Graphs and Hypergraphs. North-Holland, Amsterdam.

Bergamaschi, S., Castano, S., and Vincini, M. (1999). Semantic integration of semistructured and structured data sources. *SIGMOD Record*, 28(1), 54-59.

Brewster, C. and O'Hara, K. (2004). Knowledge representation with ontologies: the present and future. *IEEE Intelligent Systems,* 19(1), 72-81.

Boucelma, O. and Colonna, F. (2004). Mediation for online geoservices. *In Proceedings of the 4th International Workshop on Web and Wireless GIS,* 112-119.

Buitelaar, P., Cimiano, P., Frank, A., Hartung, M., and Racioppa, S. (2008). Ontology-based information extraction and integration from heterogeneous data sources. *International Journal of Human-Computer Studies,* 66(11), 759-788.

Burrough, P. and McDonnell, R., (1998)Principles of Geographical Information Systems. New York, Oxford University Press,

Castano, S., Ferrara, A., Montanelli, S., and Zucchelli, D. (2003). HELIOS: a general framework for ontology-based knowledge sharing and evolution in P2P systems. *In Proceedings of the 14th International Workshop on Database and Expert Systems Applications*, 597–603.

CEN/TC287/WG02. (1995). Geographic Information - Data Description - Quality. 1995-1-24, European Committee for Standardisation.

Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., et al. (1994). The TSIMMIS project: Integration of heterogeneous information sources. *In Proceedings of the 10th meeting of the information processing society of Japan (IPSJ)*, 7–18.

Clark, K. (2001)Analytical and Computer Cartography. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

Comibra, A., (2009).Geographic Data Integration to Support Web GIS Development. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, ACM MEDES,Oct. 27-30.

Cote, R., Jones, P., Apweiler, R. and Hermjakob, H.(2006)The ontology lookup service, a lightweight cross-platform tool for controlled vocabulary queries. *BMC Bioinformatics*. Feb 28; 7(1):7-97.

Cruz, I. and Calnan, P. (2002). Object interoperability for geospatial applications: a case study. *The Emerging Semantic Web, IOS Press*, 281-295.

Cui, G.Y., Lu, Q., Li, W.J., and Chen, Y.R. (2009). Automatic acquisition for ontology construction. *In the 22nd International Conference on the Computer Processing of Oriental Languages (ICCPOL2009),* Hong Kong, 248-259.

Davis, B. (2001). GIS: A Visual Approach. Onword Press, Santa Fe, NM.

Devillers, R., Bedard, Y., and Jeansoulin, R. (2005). Multidimensional management of geospatial data quality information for its dynamic use within GIS. *Photogrammetric Engineering and Remote Sensing*, 71(2), 205-216.

Devillers, R., Bedard, Y., Jeansoulin, R. and Moulin, B. (2007). Towards spatial data quality information analysis tools for exports assessing the fitness for use of spatial data. International Journal of Geographical Information Science. Vol. 21, No. 3, March, 261-282.

Essid, M., Colonna, F., Boucelma, O., and Betari, A. (2006). Querying mediated geographic data sources. *EDBT 2006, LNCS 3896*, 1176-1181.

Fan, H. and Poulovassilis, A. (2003). Using automed metadata in data warehousing environments. *DOLAP '03 Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP,* New Orleans, Louisiana, 86-93.

Gervais, E., Liu, H. Nussbaum, D., Roh, Y., Sack, J. and Yi, J. (2007)Intelligent Map Agents — An Ubiquitous Personalized GIS. ISPRS Journal of Photogrammetry and Remote Sensing Vol. 62, Issue 5, October, 347-365.

Ghulam, A. (2010). A Framework for Creating Global Schema Using Global Views from Distributed Heterogeneous Relational Databases in Multidatabase System. *Global Journal of computer Science and Technology*, Vol. 10, Issue 1, Apl. 2010, 31-34.

Golfarelli, M. and Rizzi, S. (1998). A methodological framework for data warehouse design. *In Proc. DOLAP '98 Proceedings of the 1st ACM International Workshop on Data Warehousing and OLAP*, 3-9.

Goodchild, F., Fu, P., and Rich, P. (2007). Sharing geographic information: an assessment of the geospatial one-stop. *Annals of the Association of American Geographers*, 97(2), 250-266.

Gruber, T. R. (1993). A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2),199-220.

Guarino, N., Oberle, D., and Staab, S. (2009). What is an ontology? *Handbook on Ontologies, International Handbooks on Information Systems, Springer-Verlag Berlin Heidelberg*, 1-17.

Gupta, A., Marciano, R., Zaslavsky, I., and Baru, C. (1999). Integrating GIS and imagery through XML-based information mediation. *In P. Agouris and A. Stefanidis (Eds). Integrated Spatial Databases: Digital Images and GIS, Lecture Notes in Computer Scienc*, LNCS 1737, 211-234.

Gupta, A., Zaslavsky, I., and Marciano, R. (2000). Generating query evaluation plans within a spatial mediation framework. *Proceedings of the 9th International Symposium on Spatial Data Handling*.120-128.

Guttman, A. (1984). R-Trees A Dynamic Index Structure for Spatial Searching. *In Proceeding of ACM SIGMOD Internatinal Conf on Management of Data*, 47-57.

Gyssens, M. and Paredaens, J.(1984). A decomposition methodology for cyclic databases, *Adv. Database Theory 2.* 85-122.

Hammer, J. and Pluempitiwiriyawej, C. (2001). Overview of the integration wizard project for querying and managing semistructured data in heterogeneous sources. *In Proceedings of the 5th National Computer Science and Engineering Conference (NCSEC 2001), Chiang Mai University, Chiang Mai, Thailand*.

Hariharan, R., Shmueli-Scheuer, M., Li,C. and Mehrotra, S.(2005). Quality-driven approximate methods for integrating GIS data. In *GIS '05 Proceedings of the 13th annual ACM international workshop on Geographic information systems.* 97-104.

Heywood, I., Comelius, S.,Carver, S. (2006). An Introduction to Geographical Information Systems. London, Pearson Prentice Hall.

Hribernik, K.; Kramer, C.; Hans, C.; Thoben, K.-D. (2010). A Semantic Mediator for Data Integration in Autonomous Logistics Processes. *Enterprise Interoperability IV. Making the Internet of the Future for the Future of Enterprise*, Springer, London, 157-167.

ISO (2003). Geographic information – Metadata. ISO 19115:2003, *International Organization for Standardization*.

ISO(2002). Geographic Information – Quality Principles, ISO/TC 211, 19113:2002, International Organization for Standardization.

Janowicz, K., Schade, S., Boring, A., Kebler, C., Maue, P. and Stasch, C. (2010). Semantic Enablement for Spatial Data Infrastructures. *Transactions in GIS*. 14(2), 111-129.

Juran, J.M., Gryna, F.M.J. and Bingham, R.S. (1974). Quality Control Handbook. (New York) McGraw-Hill.

Kim, W. and Seo, J. (1991). Classifying schematic and data heterogeneity in multidatabase systems. *IEEE Computer.* 24(12), 12–18.

Lassoued, Y., Essid, M., Boucelma, O., and Quafafou, M. (2007). Quality-driven mediation for geographic data. *In Proceeding of 5th International Workshop on Quality in Databases*. 27-38.

Lee, Y., Strong, D. Kahn, B. and Wang, R. (2002). AIMQ: a methodology for information quality assessment. *Information & Management* Vol.40, 133-146.

Lim, J. Wu, J. Singh, S. and Narashimhalu, D. (2001). Learning similarity matching in multimedia content-based retrieval. IEEE Transaction on Knowledge and Data Engineering. Vol. 13, No. 5. 846-850.

Ludascher, B., Papakonstantinou, Y., and Velikhov, P.(1999). A framework for navigation-driven lazy mediators. *In ACM Workshop on the Web and Database*.22-28.

Mena, E.V., Kashyap, A., Illarramendi, A., and Sheth, A. (1998). Domain specific ontologies for semantic information brokering on the global information infrastructure. *International Conference of Formal Ontologies in Information Systems (FIOS'98)*. Trento, Italy.54-88.

Michalowski, M., Ambite, J., Thakkar, S., Tuchinda, R., Knoblock, C., and Minton, S. (2004). Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems,* 19(3), 72-79.

Miller, L. (1992) Generating hings from arbitrary subhypergraphs. Information Processing Letters. Vol. 41, 307-312.

Miller, L. Yu, X. and Nilakana, S., (2002). Integration of relational databases and record-based legacy systems for populating data warehouses. *In Proceedings of the Hawaii International Conference on System Science*.3033-3041.

Miller, L., Bing Z., Ming, H., and Nusser, S. (2004). Supporting a virtual office environment for federal agency field operations. *The 3rd International Conference on Politics and Information Systems: Technologies and Applications (PISTA '04)*, Orlando, FL. 84-88.

Miller, L., Yu, X., and Nilakanta, S. (2002). Integration of databases and record-based legacy systems for populating data warehouses. *The 35th Hawaii International Conference on System Sciences.* 8, 3033 – 3041.

Miller, L., Ming, H., Tsai, H., Wemhoff, B. and Nusser, S. (2007). Supporting Geographic Data in the Mobil Computing Environment. In *proceedings of Parallel and Distributed Computing Systems (PDCS-2007),* ISCA 20th Int'l. Conf. Sept. Las Vegas, Nevada, 24-26.

Miller, L., Nikhil Sathe, Ming, H., Dhigha Sekaran, Nusser, S., and Pheishang Zhao. (2001). An infrastructure for delivering geospatial data to field users. *The IASTED International Conference on Parallel and Distributed Computing and Systems.* Anaheim, CA.

Ming, H. (2006). A Computation Infrastructure to Support Field Users. Creative compont, Iowa State University.

Morrison,A., Morrison, J., Muehrche,P., Kimerling,P., Guptill, S. (1995). Elements of Cartography. Canada, John Wisley & Sons.

Moulton, A., Madnick, S.E., and Siegel, M. (2002). Context interchange mediation for semantic interoperability and dynamic integration of autonomous information sources in the fixed income securities industry. *MIT Sloan Working Paper No. 4404-02*.

Mountrakis, G. Stefanidis, A. Schlaisich, I. and Agouris, P. Supporting quality-based image retrieval through user preference learning. Photogrammetric Engineering and Remote Sensing, Vol. 70, No. 8. 973-981.

Moussaoui, S., Guerroumi, M. and Badache, N. (2006). Data Replication in Mobile Ad Hoc Networks. *Lecture Notes in Computer Science (LNCS)*, 4325,685-697.

NSSDA(1998). Using the National Standard for Spatial Data Accuracy to Measure and Report Geographic Data Quality. *Positional Accuracy Handbook.*19-42.

Neches, R., Fikes, E. , Finin, T., Gruber, R., Senator, T. and Swartout, R.(1991). Enabling technology for knowledge sharing, *AI Magazine*, 12(3), 36-56.

NJOIT (2010). State of New Jersey Composite of Parcels Data, New Jersey State Plane NAD83 and MOD-IV Tax List Search Database. *New Jersey Office of Information Technology (NJOIT), Office of Geographic Information Systems (OGIS)*

Nusser, S., Miller,L., Clarke, K.and Goodchild, M.(2003). Geospatial IT for Mobile Field Data Collection. *Communications of the ACM.* Vol. 46. No. 1. 45-46.

O'Brien, J and Gahegan, M. (2005) A knowledge framework for representing, manipulating and reasoning with geographic semantics, in P.F. Fish (Ed.), *Developments in Spatial Data Handling 11th International Symposium on Spatial Data Handling*, 585 – 598.

Owrang, M. and Miller, L.(1998) Query Translation Based on Hypergraph Models. *The computer Journal*, Vol. 31, no. 2. 155-164.

Park, J. and Ram, S. (2004). Information systems interoperability: What lies beneath? *ACM Transactions on Information Systems (TOIS)*, 22(4), 595-632.

Peachavanish, P. and Karimi, H. (2007). Ontological engineering for interpreting geospatial queries. *Transactions in GIS*. 11(1), (Feb. 2007), 115–130.

Qu, S. (2003). An Infrastructure for Delivering Geospatial Data to Field users. Thesis, Iowa State University.

Ram, S., Park, J., and Lee, D. (1999). Digital Libraries for the Next Millennium: Challenges and Research Directions. *ISF, Springer*. 1(1), 75-94.

SEDAC(2008). Metadata: 2.Data Quality Information. http://sedac.ciesin.columbia.edu/metedata/guide/dataqual.html. (7/22/2008) CIESIN's Guide to FGDC Compliant Metadata.

Seng, J. and Kong, I.L. (2009). A schema and ontology-aides intelligent information integration. *Expert Systems with Applications*, 36, 10538-10550.

Smart, P., Jones, C., and Twaroch, F. (2010). Multi-source toponym data integration and mediation for a meta-gazetteer services. *GIScience 2010, LNCS 6292*, 234-248.

Sowa, J. (2001). Knowledge representation: logical, philosophical, and computational foundations. *Computational Linguistics*, 27(2), 286-294.

Stuchenschmidt, H., Visser, U., Schuster, G., and Vogele, T. (2002). Ontologies for geographic information integration. *Computers and Geosciences*, 28 (1), 103-117.

Thakkar, S.□ Ambite, J.L.□ and Knoblock, C.A. (2003). Efficient execution of recursive integration plans. *Proc 2003 IJCAI Workshop on Information Integration on the Web*. 255-268.

Thakkar, S., Knoblock, C., and Ambite, J. (2007). Quality-driven geospatial data integration. *In Proceeding of the 15th International Symposium on Advances in Geographic Information Systems, ACM GIS.*44-49.

Tsai, H., Xu, J., Lin, S., and Miller, L. (2003). Incorporating entity and function ontologies into the integration of heterogeneous, distributed data sources. *ISCA 18th International Conference on Computers and their Applications.*184-187.

Tsai, H., Miller, L., Ming, H., Wemhoffm, B., and Nusser, S. (2006). Combining spatial data from multiple data sources. *ISCA 19th International Conference on Computer Applications in Industry and Engineering*, 89-94.

Tsai, H., Miller, L., and Xu, J. (2001). Using ontologies to integrate domain specific data sources. *The 3rd International Conference on Information Reuse and Integration (IRI-2001)*, 62-67.

Tuchinda, R., Thakkar, S., Gil, and Deelman. (2004). Artemis: Integrating scientific data on the grid. *IAAI Emerging Applications*, 892-899.

UNBC GIS (2006). Mapbasics. http://www.gis.unbc.ca/courses/geog205/lectures/mapbasics/index.php , 7/27/2006. *UNBC GIS LAB*.

Vidal, V., Sacramento, E., Macedo, J., and Casanova, M. (2009). An ontology-based framework for geographic data integration. *ER 2009 Workshop, LNCS 5833*, 337-346.

Visser, U., Stuckenschmidt, H., Schuster, G., and Vogele, T. (2002). Ontologies for geographic information processing. *Computer and Geosciences*, 28(1), 103-117.

Wache, H., Vogele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hubner, S. (2001). Ontology-based integration of information – a survey of existing approaches. *Workshop on Ontologies and Information Sharing*, 108-117.

Wang, R. and Strong, D. (1996). Beyond Accuracy: what data quality means to data consumers. *Journal of Management Information Systems.* Vol. 12 (4). 5-34.

Weng, S., Tsai, H.J., Liu, S., and Hsu, C. (2006). Ontology construction for information classification. *Expert Systems with Applications*, 31(1), 1-12.

Wiederhold, G. (1992). Mediators in the architecture of future information systems. *IEEE Computer*, 25(3), 38-49.

Wu, L., Miller, L. and Nilakanta, S. (2001). Design of data warehouses using metadata. *Information and Software Technology*, 43, 109-119.

Yen, C. H. and Miller, L.L., (1995). An extensible view system for multidatabase integration and interoperation. *Integrated Computer-Aided Engineering*, 2(2), 97-123.

Yen, C. H., Miller, L. L., Sirjani, A. and Tenner, J. (1998). Extending the object-relational interface to support an extensible view system for multidatabase integration and interoperation. *International Journal of Computer Systems Science and Engineering*, 13(4), 227-240.

Yen, C. H., Miller, L. L. and Pakzad, S. H. (1994). The design and implementation of the zeus view system. *Hawaiian International Conference on Systems Science*, 206-215.

Zaslavsky, H., Baru, I., Bhatia , C., Memon , A., Velikhov , P. and Veytser, V. (2003) Grid-enabled mediation services for geospatial information. In *Proceedings of Workshop on Next Generation Geospatial Information.* 35-37.

Zaslavsky, H., Tran, J., Martone, E. and Gupta, A. (2004) Integrating Brain Data Spatially: Spatial Data Infrastructure and Atlas Environment for Online Federation and Analysis of Brain Images , *Biological Data Management Workshop (BIDM 2004) in conjunction with 15th International Workshop on Database and Expert Systems Applications (DEXA'04),* Zaragosa, Spain, Aug/Sept. 389-393.

Zou,B. Integrating Mobile computing into Fixed Network. Thesis. Iowa State University.

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to those who helped me with various aspects during the process of the research and the writing of this thesis. First and foremost, I would like to express my gratitude to my major professor, Dr. Leslie Miller, for his patience, encouragement, valuable suggestions and guidance throughout my graduate career. This dissertation would not have been possible without his support and guidance. I would also like to thank the committee members for their efforts and contributions to this work: Dr. Shashi Gadia, Dr. Sree Nilakanta, Dr. Wallapak Tavanapong and Dr. Johnny Wong.

I would also like to thank all the members of Dr. Miller's research group for their advice and assistance during the course of my research. I especially want to thank Ming Hua for his long term support and work on implementation, Becca Wemhoff for her insightful discussion and the preparation of data set, Sheng Qu for his support and assistance.

Finally, I want to thank my family: my husband, Shang-Chi Gong, and my children, Ryan and Stephanie, for their love, encouragement, patience and support.