FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

SEMANTIC DISTRIBUTED DATABASE MANAGEMENT WITH APPLICATIONS TO THE INTERNET DISSEMINATION OF ENVIRONMENTAL DATA

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Scott C. Graham

2004

To: Dean R. Bruce Dunlap
College of Arts and Sciences

This dissertation, written by Scott C. Graham, and entitled Semantic Distributed Database Management with Applications to the Internet Dissemination of Environmental Data, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

_____
Malek Adjouadi

_____
Chung-Min Chen

_____
Shu-Ching Chen

_____
Wei Sun

_____
Naphtali Rishe, Major Professor

Date of Defense: December 23, 2003

The dissertation of Scott C. Graham is approved.

_____
Dean R. Bruce Dunlap
College of Arts and Sciences

_____
Dean Douglas Wartzok
University Graduate School

Florida International University, 2004

DEDICATION

To my wife, Denise, whose love, caring, support, inspiration, and thoughtfulness help me do everything that I do. Always & Forever.

ACKNOWLEDGEMENTS

Portions of this research have been supported by AFRL, ARO, DoI, NASA, and NSF (including grant numbers EIA-0220562, HRD-0317692, and EIA-0320956).

I would like to thank my committee members for their support, patience, and occasional prodding. Special thanks are due to my major professor, Dr. Naphtali Rishe, who has been a mentor and the type of guide into academia that everyone should have.

The professors who guided me in my coursework were always helpful and enlightening. To Dr. John Comfort, whose joy for playing with computers continues to influence me, I thank you for that and I miss you; I believe that I would have finished this much sooner if you were still with us.

Finally I would like to thank my colleagues over the years, particularly Steve, Martha, Maxim, Andriy, Eric, Catherine, Gary, and Manish. You have always been there when I needed your help. Carlos, thanks for studying for and taking the qualifiers with me. Special thanks go to Therese, who made working at HPDRC such a pleasant experience for so many years.

ABSTRACT OF THE DISSERTATION

SEMANTIC DISTRIBUTED DATABASE MANAGEMENT WITH APPLICATIONS TO THE INTERNET DISSEMINATION OF ENVIRONMENTAL DATA

by

Scott C. Graham

Florida International University, 2004

Miami, Florida

Professor Naphtali Rishe, Major Professor

The research presented in this dissertation is comprised of several parts which jointly attain the goal of Semantic Distributed Database Management with Applications to Internet Dissemination of Environmental Data.

Part of the research into more effective and efficient data management has been pursued through enhancements to the Semantic Binary Object-Oriented database (Sem-ODB) such as more effective load balancing techniques for the database engine, and the use of Sem-ODB as a tool for integrating structured and unstructured heterogeneous data sources. Another part of the research in data management has pursued methods for optimizing queries in distributed databases through the intelligent use of network bandwidth; this has applications in networks that provide varying levels of Quality of Service or throughput.

The application of the Semantic Binary database model as a tool for relational database modeling has also been pursued. This has resulted in database applications that are used by researchers at the Everglades National Park to store environmental data and to remotely-sensed imagery.

The areas of research described above have contributed to the creation TerraFly, which provides for the dissemination of geospatial data via the Internet. TerraFly research presented herein ranges from the development of TerraFly's back-end database and interfaces, through the features that are presented to the public (such as the ability to provide autopilot scripts and on-demand data about a point), to applications of TerraFly in the areas of hazard mitigation, recreation, and aviation.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE PAGE

1. Semantic Databases

This chapter explores two issues of refinement of the Semantic Binary Object-Oriented Database, Sem-ODB. The first involves balancing the load among servers in a parallel implementation of the Sem-ODB engine. It was published as:

> N. Rishe, A. Shaposhnikov, S. Graham. "Load Balancing in a Massively Parallel Semantic Database." Computer Systems Science and Engineering (Special Issue on Massively Parallel Processing). v. 11, n. 4, July 1996, pp. 195-199.

The second issue involves adding security features to Sem-ODB. It was published as:

> N. Rishe, A. Shaposhnikov, S. Graham, G. Palacios. "Implementation of Security in Semantic Binary Object Database." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 446-449.

1.1. Load Balancing in a Massively Parallel Semantic Database

1.1.1. Summary

We are developing a massively parallel semantic database machine. Our basic semantic storage structure ensures balanced load for most parts of the database. The load to the other parts of the database is kept balanced by a heuristic algorithm which repartitions data among processors in our database machine as necessary to produce a more evenly balanced load. We present our inexpensive, dynamic load balancing method together with a fault-tolerant data transfer policy that will be used to transfer the repartitioned data in a way transparent to the users of the database.

1.1.2. Background

Database management systems are emerging as prime targets for enhancement through parallelism. In order for parallel database machines to be efficient, the processors

in the system must have comparable load. A massively parallel database machine which uses thousands of processors will allow for massive throughput of transactions and queries if no processors become a bottleneck. This section proposes a load balancing method for a massively parallel semantic database.

Much work on load balancing for relational databases and file systems has been done and can be utilized in our research. For example, [1] proposes several dynamic load balancing policies for multi-server file systems. A dynamic load balancing algorithm for large, shared-nothing, hypercube database computers which makes use of relational join strategies is presented in [2]. [3] presents a self-adjusting data distribution scheme which balances computer workload in a multiprocessor database system at a cell level during query processing. A run-time reorganization scheme for rule based processing in large databases is discussed in [4].

Our database computer will make use of a shared-nothing architecture. The computational load on each processor of our database computer will vary directly with the demand for data on that processor. Imbalances in the number of data accesses among nodes can be rectified by repartitioning the database, much as imbalances in computational demands in process scheduling can be rectified by moving processes from one machine to another. When a range of facts in our database is moved from one processor's control to another processor's control, the load on the first processor will go down. The methods for determining imbalances in our system, and the methods to relieve these imbalances in our system, is very similar to the methods used for computational dynamic load balancing in shared-nothing computers. An adaptive, heuristic method for dynamic load balancing in a message-passing multicomputer is presented in [5]. A semi-

distributed approach to load balancing in massively parallel multicomputer systems is presented in [6].

Our massively parallel database machine architecture makes use of a distributed system of many processors, each with its own permanent storage device. This shared-nothing approach requires that any load balancing operations be performed by message passing. The data distribution scheme that is used in our database system allows load balancing to be achieved by data repartitioning among the nodes of our system.

This section refines the results reported in [7] and extends them by adding a fault tolerant data transfer policy for data repartitioning.

1.1.3. The Semantic Binary Database Model

The semantic database models in general, and the Semantic Binary Model SBM ([8] and others) in particular, represent the information of an application's world as a collection of elementary facts categorizing objects or establishing relationships of various kinds between pairs of objects. The central notion of semantic models is the concept of an *abstract object*, which is any real world entity that we wish to store information about in the database. The objects are categorized into classes according to their common properties. These classes, called *categories*, need not be disjoint — that is, one object may belong to several classes. Further, an arbitrary structure of sub-categories and super-categories can be defined. The representation of the objects in the computer is invisible to the user, who perceives the objects as real-world entities, whether tangible, such as persons or cars, or intangible, such as observations, meetings, or desires. The database is perceived by its user as a set of facts about objects. These facts are of three types: facts

stating that an object belongs to a category: *xC*; facts stating that there is a relationship between objects: *xRy*; and facts relating objects to data, such as numbers, texts, dates, images, tabulated or analytical functions, etc: *xRv*. The relationships can be of arbitrary kinds; stating, for example, that there is a many-to-many relation *address* between the category of persons and texts means that one person may have an address, several addresses, or no address at all.

1.1.4. Storage Structure

An efficient storage structure for semantic models has been proposed in [9] and [10]. The collection of facts forming the database is represented by a file structure which ensures approximately one disk access to retrieve queries of any of the following forms:

1. For a given abstract object *x*, verify/find what categories the object belongs to.
2. For a given category, find its objects.
3. For a given abstract object *x* and relation *R*, retrieve all/certain *y* such that *xRy*.
4. For a given abstract object *y* and relation *R*, retrieve all/certain abstract objects *x* such that *xRy*.
5. For a given abstract object *x*, retrieve (in one access) all (or several) of its direct and/or inverse relationships, *i.e.* relations *R* and objects *y* such that *xRy* or *yRx*. The relation *R* in *xRy* may be an attribute, *i.e.* a relation between abstract objects and concrete objects.
6. For a given relation (attribute) *R* and a given concrete object *y*, find all abstract objects such that *xRy*.

7. For a given relation (attribute) R and a given range of concrete objects $[y_1, y_2]$, find all objects x and y such that xRy and $y_1 \leq y \leq y_2$.

The entire database can be stored in a single file. This file contains all of the facts of the database (xC and xRy) as well as additional information called inverted facts: Cx, Ryx. The inverted facts ensure that answers to queries of forms 2, 4, 6, and 7 are kept in a contiguous segment of data in the database which allows them to be answered with one disk access. The direct facts xC and xRy allow the database to answer queries of forms 1, 3, and 5 with one disk access. The file is maintained as a B-tree. The variation of the B-tree used allows both sequential access according to the lexicographic order of the items comprising the facts and the inverted facts, as well as random access by arbitrary prefixes of such facts and inverted facts. Facts which are close to each other in the lexicographic order reside close to each other in the file. (Notice that although technically the B-tree-key is the entire fact, it is of varying length and on the average is only several bytes long, which is the average size of the encoded fact xRy.)

Consider, for example, a database containing information regarding products manufactured by different companies. The following set of facts can be a part of a logical instantaneous database:

1. object1 *COMPANY*
2. object1 *COMPANY-NAME* `IBM'
3. object1 *MANUFACTURED* object2
4. object1 *MANUFACTURED* object3
5. object2 *PRODUCT*
6. object2 *COST* 3600

7. object2 *DESCRIPTION* `Thinkpad'
8. object3 *PRODUCT*
9. object3 *COST* 100
10. object3 *DESCRIPTION* `TrackPoint'

The additional inverted facts stored in the database are:

1. *COMPANY* object1
2. *COMPANY-NAME* `IBM' object1
3. object2 *MANUFACTURED-BY* object1
4. object3 *MANUFACTURED-BY* object1
5. *COST* 3600 object2
6. *COST* 100 object3
7. *DESCRIPTION* `Thinkpad' object2
8. *DESCRIPTION* `TrackPoint' object3
9. *PRODUCT* object2
10. *PRODUCT* object3

To answer the elementary query "Find all objects manufactured by object1", we find all the facts whose prefix is object1_*MANUFACTURED*. (`_' denotes concatenation.) These entries are clustered together in the sorted order of direct facts.

To answer the elementary query "Find all products costing between $0 and $800", we find all the facts whose prefix is in the range from *COST*_0 to *COST*_800. These entries are clustered together in the sorted order of inverted facts.

In the massively parallel version that we are developing, the B-tree is partitioned into many small fragments, each residing on a separate storage unit (*e.g.*, a disk or non-

volatile memory) that is associated with a fairly powerful processor. This disk-processor pair is called a *node*. Each node can retrieve information from the disk, perform the necessary processing on the data and deliver the result to the user, or to the other nodes. For updates the node verifies all of the relevant integrity constraints and then stores the updated information on the disk. Many database fragments can be queried or updated concurrently.

The queries and transactions will enter into the network through host interfaces. Every host interface maintains a copy of the Partitioning Map (PM) of the entire database. Since the whole database is a lexicographically ordered file represented by a set of B-trees, the map needs to contain only a small number of facts for each node: the lexicographically minimal and maximal facts for each B-tree fragment that is stored on that node. The map changes only when the database is re-partitioned. The distribution policy that we propose in this work provides repartitioning that is rare, inexpensive, localizable, and invisible to the system until all of the shifting of data is complete, and that does not interfere with the normal operation of the system.

Most of the physical facts that are in our implementation of a semantic binary database start with an abstract object. These facts are ordered by the encoding of the abstract objects, which assigns a unique quasi-random number to each abstract object. Since there are so many of these facts, and since the objects are randomly ordered, we can assume that traffic to each partition of these facts will be balanced over time. Other facts in a semantic binary database start with an inverted category or an inverted attribute (i.e. a relation between an abstract object and a printable value). It is possible that at some time there may be a need to access a certain attribute or category more often than

other attributes or categories. The same may be true for a specific range of values of a given attribute. Since all facts that refer to a particular inverted attribute or inverted category are clustered together, this may cause a higher load on some processor/disk pairs than on others. Since load imbalances can occur in some kinds of facts but not others, the file containing the facts will be split into two subfiles. The first subfile will contain all the facts that begin with an abstract object. The second subfile will contain the facts that begin with an inverted attribute or category. Additionally there is a third subfile containing long data items: texts, images, etc., which are pointed to by facts. Each subfile will be initially partitioned evenly over all the processor/disk pairs in the system. The first subfile is already balanced; the second and third subfiles may become unbalanced and will require a block placement algorithm that allows the data to be repartitioned. By repartitioning data, we will be able to more evenly balance the load to each data partition.

1.1.5. Request Execution Scheme

We employ a deferred update scheme for transaction processing. This means that transactions are not physically performed until they are committed, but are accumulated by the database management system as they are run. Upon completion of the transaction the DBMS checks its integrity and then physically performs the update. A completed transaction is composed of a set of facts to be deleted from the database, a set of facts to be inserted into the database, and additional information needed to verify that there is no interference between transactions of concurrent programs. In our parallel database, each node is responsible for a portion of the database. When an accumulated transaction is

performed, the sets of facts to be inserted into, and deleted from, the database must be broken down into subsets that can be sent to the processors which are responsible for the relevant ranges of data.

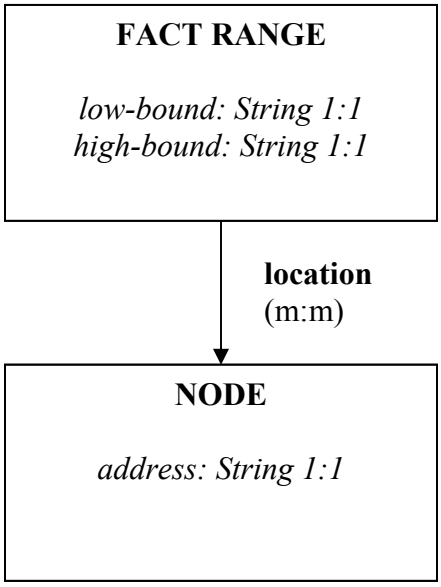


Figure 1. A Partitioning Map

Each host in the system will have a copy of the Partitioning Map (PM). The Partitioning Map is a small semantic database containing information about data distribution in the system. Figure 1 is a semantic schema of the partitioning map.

The partitioning map contains a set of ranges and their lexicographical bounds — the *low-bound* and the *high-bound* values. When a query or transaction arrives, the host will identify its lexicographical bounds. The host will then use the partitioning map to determine a set of ranges that needs to be retrieved or updated and hence the nodes which will be involved in the current transaction or query.

The partitioning map will be replicated among hosts. However, this does not imply that we need a global data structure; the algorithm described below allows updates of the partitioning database to be performed gradually, without locking and interrupting all hosts.

A range can be obtained from the node pointed to by the *location* reference in the partitioning database. This node should either have the range or a reference to another node which contains the range.

To perform load balancing we will need to move ranges from one node to another. A moved range will be accessible via an indirect reference that is left at its previous location. Such an indirect access slows down the system, especially when the range is frequently accessed by users. To allow a direct access to the moved range we need to update the *location* reference in the partitioning database. We will not perform this update simultaneously for all the host interfaces. The update will be performed when a host executes the first query or transaction that refers to the range that was transferred. The node that actually holds the range will send the results to the host along with a request to update the partitioning map. This means that the first transaction will have to travel a little further than all subsequent transactions. The second and future queries or transactions made through this host will be executed directly by the node pointed to by the *location* reference.

The data structure at each node which supports indirect referencing will be exactly the same as the partitioning map described above. We will call this data structure a local partitioning map.

Each range of facts will be represented as a separate B-tree structure which will reside on the node pointed to by the partitioning map. Consider a case where a range has been moved several times from one node to another. We may have multiple indirection references to the actual location of the range. These indirect references will be changed to direct references as described above.

1.1.6. Data Transfer Policy

In order to ensure that the database remains consistent throughout a load balancing data transfer, load balancing actions are executed as transactions initiated by the system. A large range of facts is transferred by executing a series of small system transactions that transfer small portions of data from one partition to another. The system transactions are subject to the same logging and recovery actions as regular, user initiated, transactions. Apart from the data transfer, each small load balancing transaction also includes the data necessary for updating the partitioning map. To ensure that the partitioning map remains consistent, the partitioning map update is executed using a 2-phase commit protocol.

1.1.7. Load Balancing Policy

When idle, the host interfaces will send data and work load statistics recently accumulated from the nodes to a Global Performance Analyzer (GPA). The host interfaces accumulate this data as the results of queries and transactions flow through them back to the user. The GPA is a process that analyzes the statistical information obtained and generates preferable directions of data transfer for each node.

The statistics report will contain only the changes since the previous report:

- Changes in data partitioning
- Number of accesses for each range
- Free space on each node

The GPA will use a heuristic search algorithm which uses a choice function to select a small number of possible data movements for the system. The final state will be estimated by a static evaluation function S. The GPA will select the data movement with the lowest value of the resulting static evaluation S.

The choice function should comply with the following strategies:

1. Whenever possible load balancing should be achieved by joining ranges together. Joining ranges will result in faster query execution and smaller partitioning maps.
2. A criterion for determining preferable destinations for a range transfer is the desire to move a range to a destination node which contains the lexicographically closest range to the transferred range. In other words, it is desirable to locate lexicographically close ranges on the same node whenever possible.
3. If a range has an exceptionally high number of access or requires an exceptionally large amount of storage — split the range into several parts and transfer them to other nodes.

Each node will be characterized by two parameters:

1. The amount of free disk space on the node, *F*
2. The percentage of idle time *I*. In other words the *I* is:

   $I = Idle / T,$ where

   T is a given time interval and *Idle* is the node's idle time during the time *T*.

The resulting state will be estimated by the following parameters:

1. A — the total amount of data that will be necessary to transfer in the system
2. D_F — the mean square deviation of F
3. D_I — the mean square deviation of I
4. M — total number of ranges in the system

The static evaluation function can be represented as:

$$S = C_1 * A + C_2 * D_F + C_3 * D_I + C_4 * M,$$

where C_1, C_2, C_3, and C_4 are constants.

1.1.8. Results

Our load balancing algorithm will provide our massively parallel semantic database machine with a method to repartition data to evenly distribute work among its processors. The algorithm has very little overhead, as its statistics are accumulated during the normal processing of transactions. The load balancing is accomplished by repartitioning parts of the database over the nodes of the database machine. The repartitioning will be transparent to the users and will not adversely affect the performance of the system. Our fault-tolerant data transfer policy will ensure that the database and its partitioning maps remain consistent during repartitioning.

1.2. Implementation of Security in a Semantic Binary Object Database

1.2.1. Summary

In this section we introduce a semantic binary object database technology with userviews and show how the security and data protection can be implemented in a client-server semantic database. We describe how user accesses can be restricted to parts of

semantic schemas (userviews) utilizing a schema management solution based on a semantic metaschema. We also describe how network and password security is implemented in our semantic binary database server.

1.2.2. Background

Semantic Object Binary Database—SemODB [8] is a technology developed for storing information as a collection of binary facts about real world objects. The Semantic Binary Database uses more flexible data representation than the regular object oriented models. For example, an object can be dynamically assigned to arbitrary new classes called categories. One-to-many relationships are represented efficiently and naturally using the same binary facts used to represent the regular many-to-one relationships. Like object-oriented databases, semantic databases use database schemas that describe a set of possible object classes called categories, sets of object attributes, methods for each category and relationships between objects called relations. Only binary relationships are allowed in a semantic binary database, meaning that all possible relationships are limited to represent a relation between two objects. While this limitation allows simple and efficient data structures, it does not limit the generality of semantic databases, because N-ary type of relationships can be easily converted to N binary relations to an additional object that represents the relation itself. By limiting ourselves to binary relationships, we can store all possible types of object relationships and attributes as an ordered collection of binary facts. Without loss of generality, the object attributes can also be considered as a special case of binary relation: a relation between an object and a value representing the attribute value, such as a number or a string.

Userviews allow database administrators to create a number of virtual semantic databases that are derived from an existing database. A view defines virtual categories and relations that map to physically stored categories and relations. Such mapping can be a direct one-to-one mapping or a more complex mapping that is defined by database queries. By limiting user accesses to certain userviews, we can implement very flexible data protection schemes for semantic databases. Not only can we hide certain categories, attributes and relations from unauthorized users, we can also rename the categories and relations, as well as define completely new schemas that use database queries to filter out potentially security-sensitive information.

This section will introduce how the security and userviews were implemented in our semantic database. We start by introducing semantic schemas in general and a semantic metaschema for userviews [8]. We also discuss the implementation of security features in our database server.

1.2.3. Semantic Schemas

Categories and relations of a particular database form a semantic schema. Categories are represented as boxes on the schema. Attributes of objects are represented as a collection of names and types inside the boxes. Relations between categories are represented as directed arrows pointing from domain of a relation to the range of the relation. Figure 2 shows a semantic schema of a simple semantic database.

Here the dashed line represents an inheritance relation between STUDENT and PERSON. A more detailed description of semantic binary schemas can be found in [8].

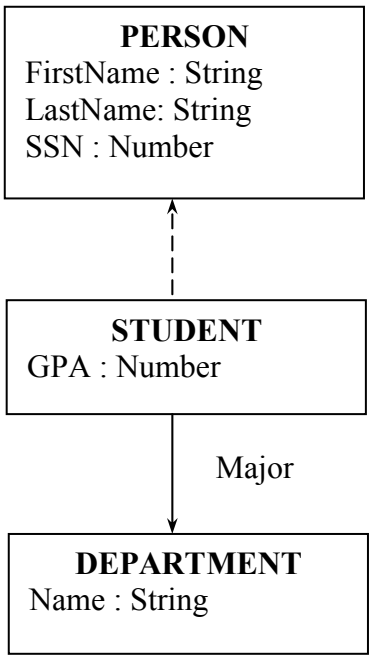


Figure 2. A Simple Semantic Schema

1.2.4. Metaschemas

Our implementation of a semantic database stores the database schemas as a part of the database itself. The categories and relations of an arbitrary database schema are represented as objects conforming to a semantic metaschema. The semantic metaschema is a schema of a database for storing any semantic schema. The objects in any semantic schema such as the category Student and attribute GPA are metaobjects belonging to some categories of the metaschema. Figure 3 shows the metaschema for our semantic binary database.

The category STUDENT is an object of the metacategory CATEGORY. The relation Major is an object that belongs to the metacategory RELATION. The categories can be abstract (represented by the ABSTRACT CATEGORY) and concrete (CONCRETE CATEGORY). Concrete categories include attributes such as numbers,

strings, enumerated type, and binary attributes. The concrete categories are represented by the categories NUMBERS RANGE, STRINGS RANGE, ENUMERATED TYPE, and BINARY DATA on the metaschema. Abstract categories such as STUDENT, are stored in the category ABSTRACT CATEGORY on the metaschema. A supercategory METAOBJECT stores both categories and relations. Basic integrity constraints are represented by the categories CO-IDENTIFICATION KEY, MANY TO ONE, ONE TO MANY, TOTAL, COVERING GROUP, and DISJOINT GROUP at the metaschema.

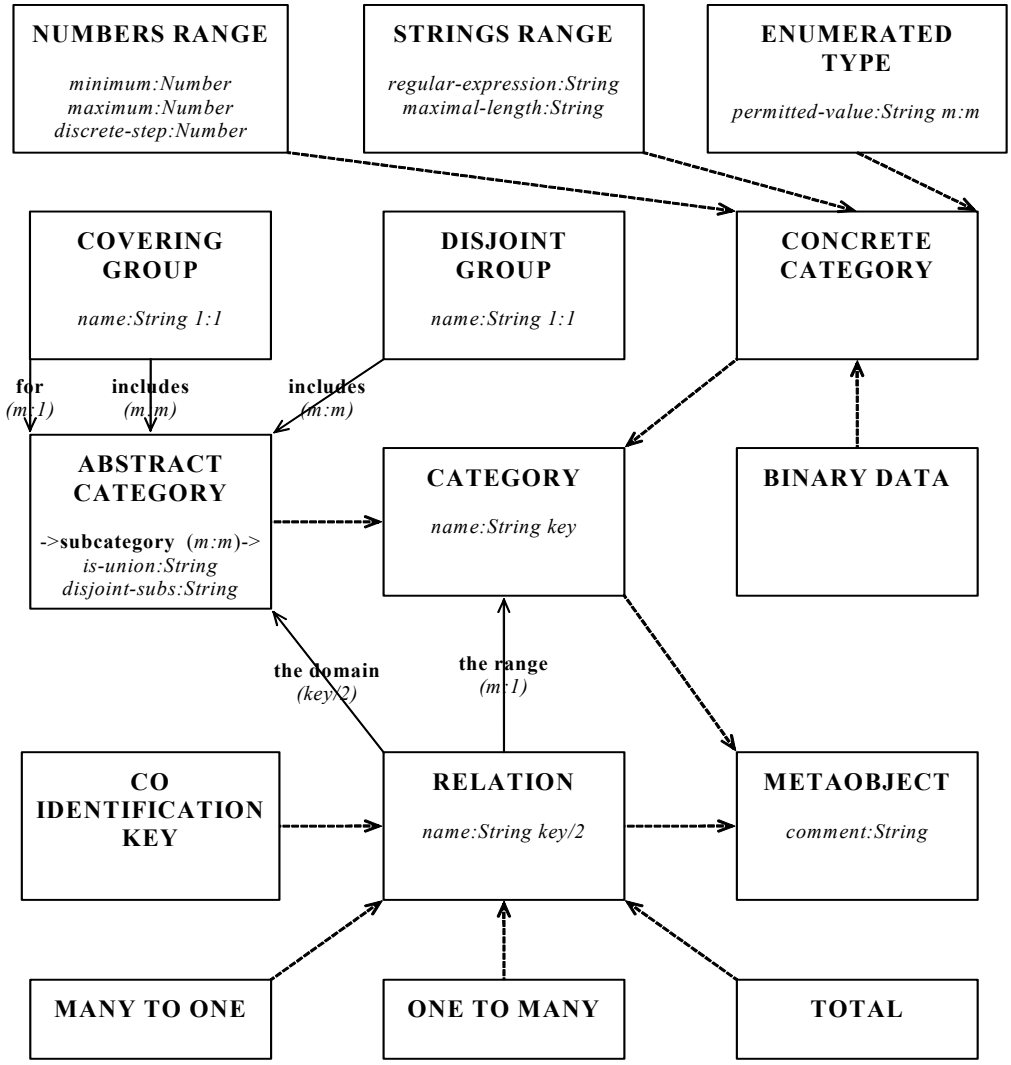


Figure 3. Semantic Binary Metaschema

Our implementation of SemODB allows the users to perform the same database operations with respect to schemas as to the regular data. The metaschema is an integral part of each database. This permits the semantic database to easily implement on-the-fly schema changes. Database applications using SemODB can dynamically change the database schemas as well as data stored in the database in the same transaction.

This is one of the advantages of SemODB as compared to regular object oriented databases that usually do not permit on the fly schema changes.

1.2.5. Userviews

Userviews are also an integral part of each database. A userview defines a sub-schema of a semantic schema. A user has access only to a limited number of userviews and always opens a database through some userview. A database administrator may access the complete schema that includes all categories and relations without restriction.

Figure 4 shows the metaschema for userviews. The USERVIEW category is a subcategory of the category METAOBJECT. It defines a user's view of the schema. Userview is a collection of metaobjects with restricted VIEWABILITY. The VIEWABILITY defines which metaobjects can be read, inserted or deleted in a particular userview. The category USER stores the information about the users who can access the USERVIEW.

The database server uses the meta-data stored according to the metaschema for userviews to verify if a user has permissions to read or modify objects that belong to certain categories and relations. If a user does not have access to a category, relation, or attribute, such metaobjects and the data will not be visible to the user application.

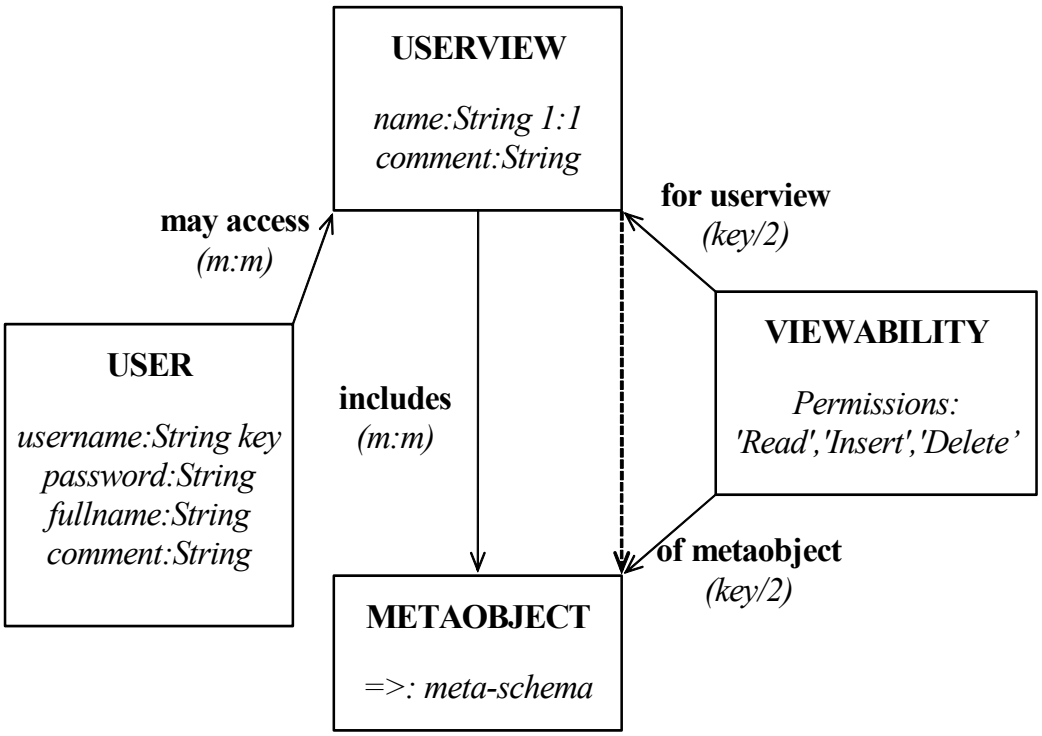


Figure 4. Metaschema of Userviews

1.2.6. Security in Client-Server Model

The SemODB server is started by running a SemODB kernel application that we call Control Server (see Figure 5). After the SemODB Control Server is started, the remote clients can connect to the server via the internet and the TCP-IP protocol stack. There are two types of remote SemODB clients: SemODB API remote clients and semantic SQL/ODBC remote clients. SemODB API remote clients include Java and C++ applications connecting to the database directly and utilizing the SemODB application programming interface. SQL/ODBC clients connect to the database via TCP-IP to the ODBC driver.

Figure 6 shows the architecture design of our security implementation for a client-server semantic database. Database clients connect to the database control server using an

SSL encrypted connection. Network password security is protected by SSL as well as by hash digests. The passwords are never transmitted over the network, instead we transmit their hash digests obtained using the Haval [11] secure hash algorithm. It is computationally infeasible to find the password given the secure hash digest.

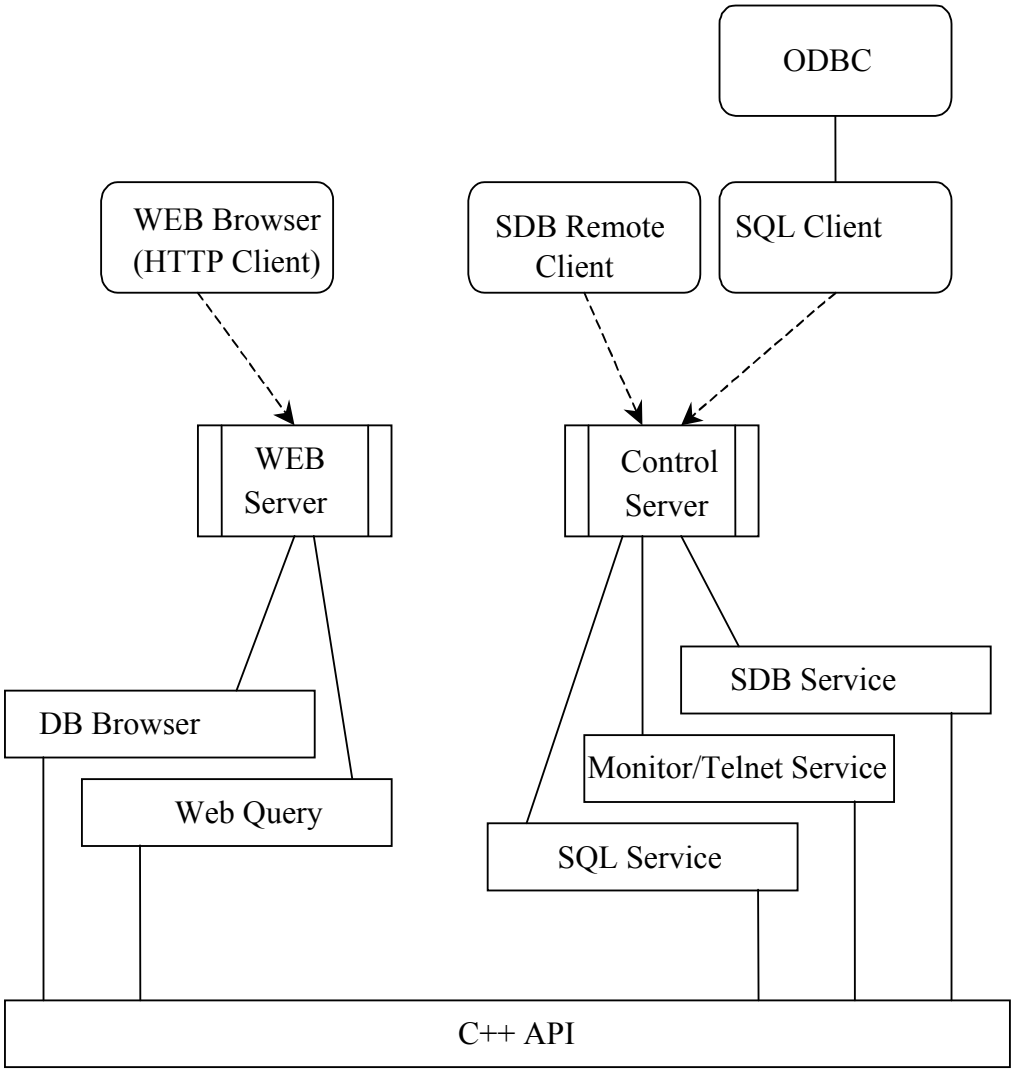


Figure 5. Client-Server SemODB

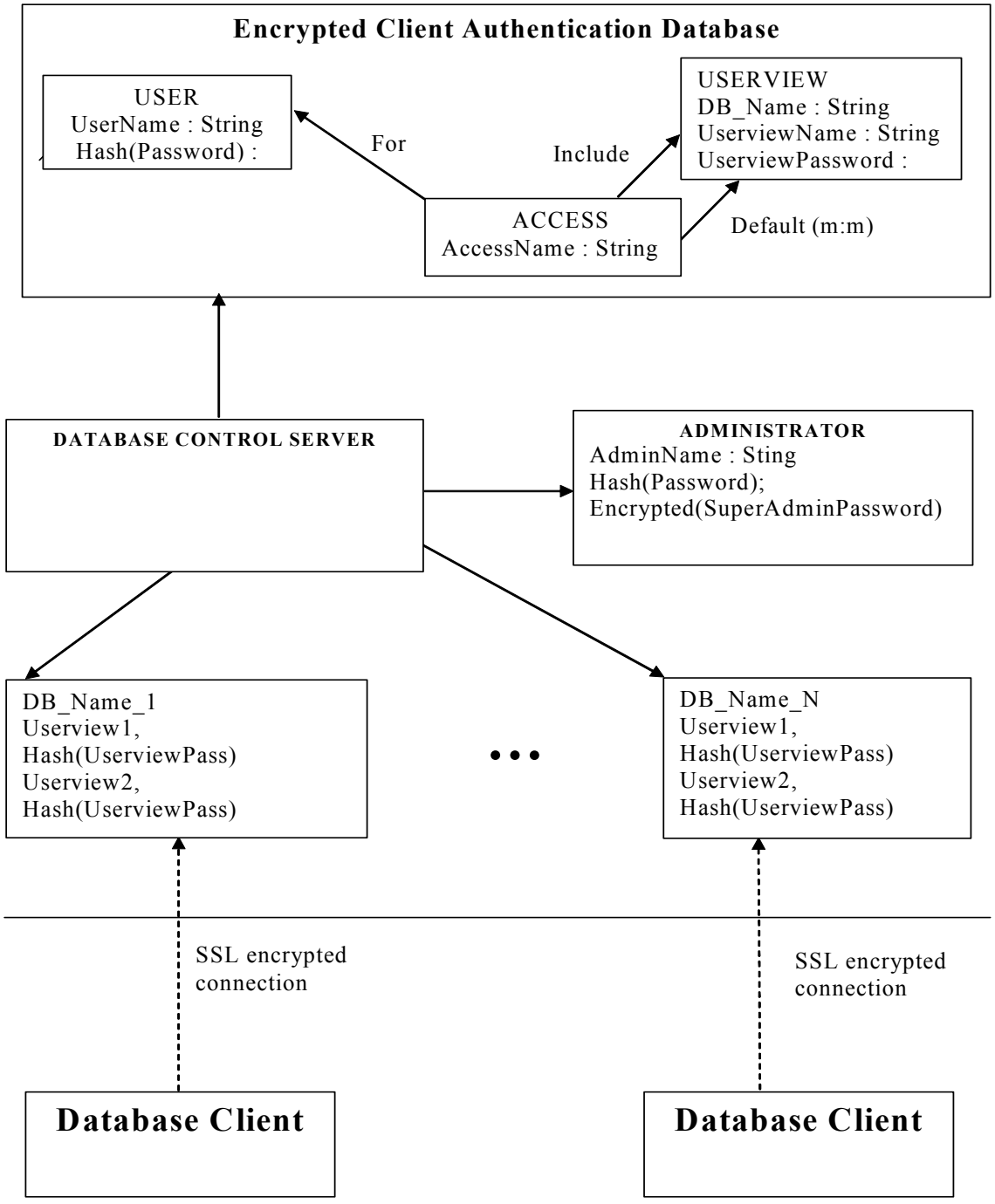


Figure 6. Security Implementation

The database Control Server authenticates the clients using the Encrypted Client Authentication Database. The database contains access rights for each user. The access

rights are grouped into access groups represented by the category ACCESS. An access group may include several userviews to one or more databases. There is one default userview for each access group that is used when a userview is not specified by the client. When a database client connects to the database, the Database Control Server queries the Client Authentication Database and assigns a userview to the client based on the username, password and the requested name of userview. Once a connection is established, the user can only retrieve the data in the corresponding userview.

The client authentication database is encrypted by using a strong 128 bit twofish encryption algorithm [12] to prevent unauthorized access to passwords even if the access to the authentication database file is compromised.

Only an administrator can start the database control server. When the Control Server starts, the administrator's password is verified by using a small database shown in the ADMINISTRATOR category box that stores secure hash digests derived from the administrator passwords using the Haval algorithm [11]. Once the administrator credentials have been verified by the server, the SuperAdminPassword is decrypted using the administrator's password. The SuperAdminPassword is then used to decrypt the encrypted Client Authentication Database that stores the permissions for the clients. This setup allows us to have more than one database administrator and password. A GUI administrative tool was developed to manage the userviews, user access permissions, users, and administrators.

1.2.7. Results

We have described the security model of our semantic database based on a semantic metaschema and userviews. The Semantic database offers convenient and flexible tools to restrict user access to parts of databases called userviews. Userviews are implemented as a part of semantic binary Metaschema. We also described how our implementation of semantic database server implements network and password security.

2. Heterogeneous Databases

This chapter explores three issues related to managing heterogeneous data sources through Semantic Modeling. The technique employed employs the design of Semantic Wrappers for the heterogeneous data sources. This allows the various data sources to all be seen as Semantic database by the users, allowing the benefits of the Semantic model to be exploited.

The first item describes the algorithms used in creating a Semantic Wrapper for Relational Databases. It was published as:

> N. Rishe, M. Chekmasov, R. Rodriguez, S. Graham, D. Mendez. "On the Algorithm for Semantic Wrapping of Relational Databases." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 436-440.

The second item describes the system architecture of a Web application for accessing relational databases through the Internet via a Semantic Wrapper. It was published as:

> N. Rishe, T. Huang, M. Chekmasov, S. Graham, L. Yang, S. Himmelsbach. "Semantic Wrapping Tool for Internet." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 441-445.

The third item describes a process for extracting data from Web sites and creating a structured wrapper for them. This allows web sites to be accessed via Sem-ODB's query tools. This item was published as:

> D. Beryoza, N. Rishe, S. Graham, I. DeFelipe. "Data Extractor Wrapper System." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 425-431.

The next two items describe methods for mapping XML DTD's and Schema's into the Semantic Binary Data model. This will allow the management of XML data in Sem-ODB. These items were published as:

> N. Rishe, L. Yang, M. Chekmasov, M. Chekmasova, S. Graham, A. Roque. "Mapping from XML DTD to Semantic Schema." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 450-455.
>
> L. Yang, N. Rishe, M. Chekmasov, S. Graham, I. DeFelipe. "XML Schema Modeling Using Sem-ODM." 9th International Conference on Distributed Multimedia Systems (DMS 2003). pp. 187-192.

2.1. On the Algorithm for Semantic Wrapping of Relational Databases

2.1.1. Summary

It is well-known that the relational database model and the SQL query language are presently the most popular tools to implement and query databases. A certain level of expertise is needed to use SQL. Thus, it is a common practice to create numerous applications atop relational database aiding users to query the database and receive results. In the present section we discuss and illustrate an algorithm to represent the relational database schema in a semantically rich way. Use of the resulting semantic schema (also called semantic view of the relational database schema) in database applications substantially reduces the size and complexity of queries to the relational

database, shortens database application development cycle, and improves maintenance and reliability by reducing the size of application programs.

2.1.2. Background

An algorithm describing the translation process of a relational database schema into a semantic schema, which is an essential part of the semantic wrapping methodology, is thoroughly discussed in [13]. The semantic wrapping methodology, in turn, is based on the semantic modeling approach proposed in [8]. The algorithm under consideration facilitates creation of a high-level semantic database design, which is free of the technicalities and complications of the relational database schema. This algorithm is implemented as a component of a software tool, called Semantic Wrapper. Semantic Wrapper provides access to the relational databases by a broader audience of users by employing Semantic SQL for querying semantic views rather than the relational database directly. We refer to [14] for details on Semantic SQL. Furthermore, the use of the mapping algorithm presented here is extended in the design of a heterogeneous multi-database environment, which includes relational, semantic databases and Internet data sources, as explained in [13]. Throughout the section we use the term 'semantic schema' as it is introduced in [8]. By relational database schema we assume a set of objects (like tables, primary/foreign key constraints, etc.) created under a certain RDBMS (relational database management system) to serve users' data storage and retrieval needs.

2.1.3. Description of the Algorithm

The process of creating a semantic view of a relational database assumes that metadata describing tables, their respective attributes, primary/foreign keys and other

constraints can be received via RDBMS hosting the database. This information is used to map relational constructs into the concepts of a semantic schema. The mapping algorithm does not affect the data, which is stored in the relational database. The algorithm for creating a semantic view of the relational database can be described as follows:

1. For each table in the relational database create a category on the semantic view.
2. For each column in the relational table create an attribute in the corresponding category on the semantic view.
3. For each functional dependency in the relational database create a relation on the semantic view.
4. Since relations are established between the categories, remove the attributes which represent foreign keys from the domain categories of the relations on the semantic view.
5. Replace the categories that represent tables serving the role of many-to-many relationship between other tables in the relational database with many-to-many relations between the corresponding categories on the semantic view.
6. Replace the categories that represent tables serving the role of recursive reference in the relational database with the relation of type is-part-of in the corresponding category on the semantic view. The cardinality of this relation may be many-to-one or many-to-many depending on the original relationship implemented in the database.
7. Replace the categories that represent tables serving the role of one-to-many relationships in the relational database with the attributes of cardinality one-to-many in the corresponding category on the semantic view.
8. Introduce subcategory/supercategory hierarchy on semantic view.

Note that performing steps (1)-(4) of the algorithm will already lead to a valid semantic view of the relational database. Steps (5)-(8) lead to further refinement and enhancement of the semantic view. Thus in practice, steps (5)-(8) of the algorithm are not mandatory but are highly recommended. Implementing all the steps of the algorithm will usually lead to the semantic schema that shows the semantics of the data in the database in a most efficient way.

2.1.4. Illustration of the Algorithm

We will now turn to Figure 7 to illustrate the algorithm of creating a semantic view for the relational database.

Figure 7 shows a Geography database which stores information about airports, cities, countries, proximity of cities to each other, and currencies used in the countries. There are several requirements to the database that led to the current implementation of the relational schema. The first requirement mandates that the country may use more than one currency. For example, The United Kingdom uses British Pounds (GBP) and Euro of the European Union (EUR). Another agreement is that the database stores not only official names of the country, but also other commonly used names. For example, the database may store 'The United States of America', 'The United States', and 'The U.S.A.' for this country. Finally, the database should relate the cities that are located relatively close to each other. This may help travelers to identify cities they may take a flight to. For example, a person traveling to Miami may arrive at Miami International (MIA) or Ft. Lauderdale International (FLL) airport. Let us now use the algorithm described above to create a semantic view for the database.

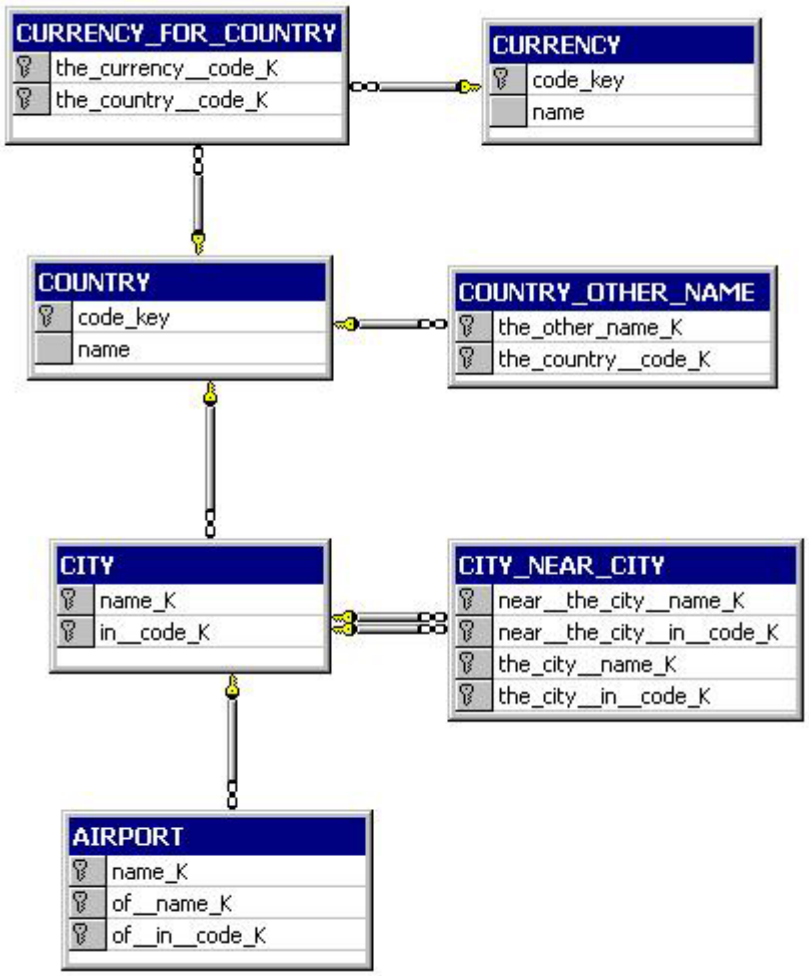


Figure 7. Relational schema of Geography database

Steps (1)-(4) of the algorithm are relatively straightforward and will lead us to the semantic view presented on Figure 8.

Having the intermediate semantic schema of the database we can implement step (5) of the algorithm. We recognize that category CURRENCY-FOR-COUNTRY has no attributes and it is a domain of two many-to-one relations the-currency and for to the categories CURRENCY and COUNTRY respectively. It is evident that CURRENCY-FOR-COUNTRY does not have any additional information to what we have in the categories CURRENCY and COUNTRY and that it solves solely the purpose of many-

to-many relationship between the respective categories. As a result of step (5) of the algorithm we replace CURRENCY-FOR-COUNTRY with the many-to-many relation for between CURRENCY and COUNTRY as indicated on Figure 9.

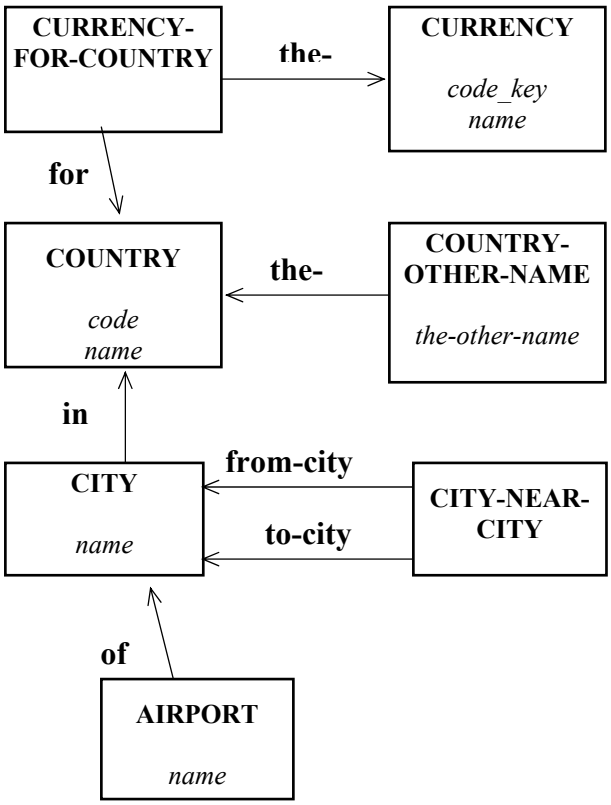


Figure 8. Intermediate Semantic Schema for Geography Database

We have the case on our intermediate semantic view to employ step (6) of the algorithm. As we see on the schema the category CITY-NEAR-CITY has no attributes. However the important elements of the schema are two relations from-city and to-city of cardinality many-to-one from CITY-NEAR-CITY to CITY. They represent the information on proximity of cities to one another. At this stage we can eliminate category CITY-NEAR-CITY with its relations and replace it with the relation near of type is-part-of with cardinality many-to-many in category CITY as illustrated on Figure 10.

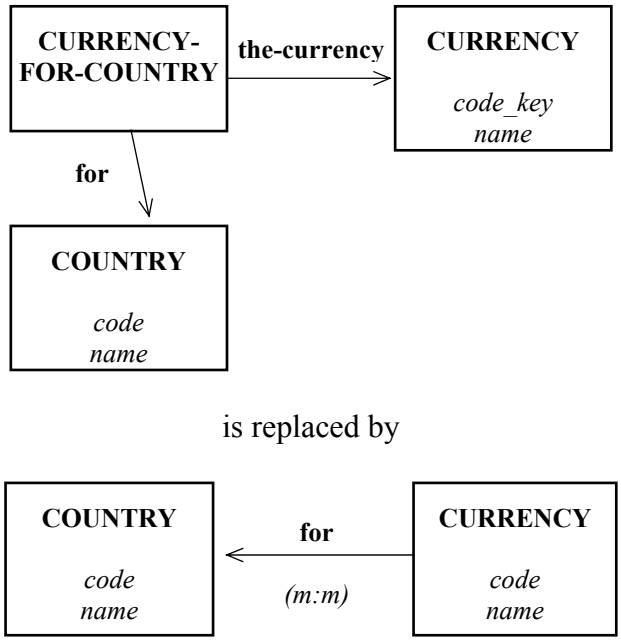
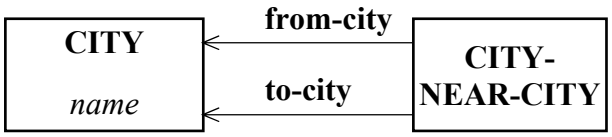


Figure 9. Application of step (5) of the algorithm to Geography database.

With respect to step (7) of the algorithm we also have a good candidate among the categories to work with. As we see on the intermediate semantic view categories COUNTRY and COUNTRY-OTHER-NAME essentially store the same information, the names of the countries. Thus there is little sense to keep two categories on the semantic view with the same information. We may introduce an additional attribute *other-name* to the category COUNTRY and get rid of the category COUNTRY-OTHER-NAME on the semantic view. However, since relation the-country of cardinality many-to-one has COUNTRY-OTHER-NAME as its domain and COUNTRY as its range the cardinality of a new attribute *other-name* in COUNTRY will become one-to-many. Figure 11 illustrates the transformation.

Finally, by employing step (8) of the algorithm we may reach one more level of abstraction on the semantic view of the relational database. We notice that countries,

cities and airports are geographical entities that have a common feature: the objects of the categories COUNTRY, CITY and AIRPORT should have names. This allows us to introduce a supercategory for these categories, which will store this common attribute. Thus we introduce category GEOGRAPHICAL-ENTITY with one attribute name. The respective categories COUNTRY, CITY and AIRPORT will now be the subcategories of the new category and attribute name will be deleted from these three categories. Figure 12 illustrates this step.



is replaced by:

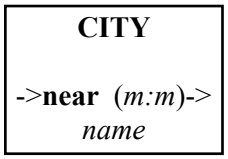
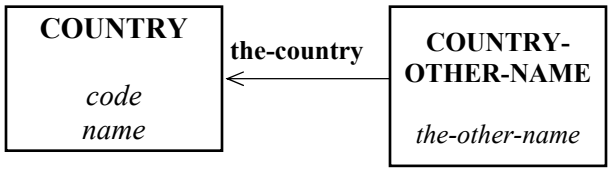


Figure 10. Application of step (6) of the algorithm to Geography database.



is replaced by:

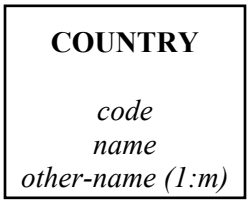


Figure 11. Application of step (7) of the algorithm to Geography database.

2.1.5. Discussion on Semantic Views

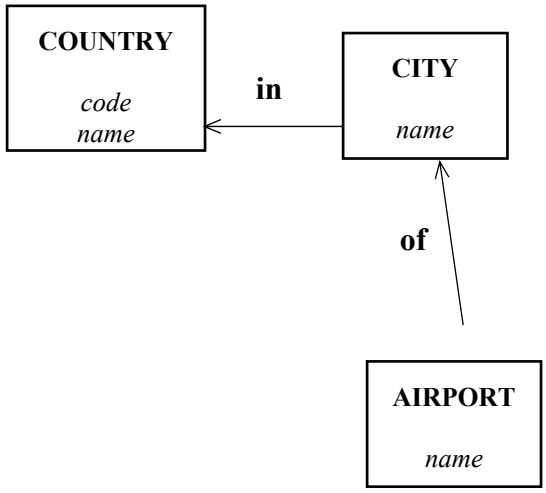
We have learned how to create a semantic view of the relational database, but the main goal is to use the view to query the database. In this section we would like to discuss the questions related to creation, maintenance and use of the semantic views when they are supported by a real-time software system.

We have performed research on how many steps of the semantic view creation algorithm can be automated for a conventional relational database implemented on a popular commercial RDBMS such as Oracle, Microsoft SQL Server, IBM DB2, or Sybase. It is well-known that although commercial RDBMS support the data definition language (DDL) of the SQL standard, they usually enhance the DDL for their respective databases. This allows the RDBMS developers to fully exploit the benefits of their particular implementation of the relational database model, but it makes it harder for our proposed system, Semantic Wrapper, to read the relational database metadata and to use that information to create a default semantic view. We have identified that steps (1)-(4) of the algorithm can be automated for a conventional relational database. Steps (5)-(8) will usually require human intervention.

We have identified that the DBA (database administrator) of the relational database should be the person responsible for creation and maintenance of the semantic views. There are several reasons to support this proposition:

1. The DBA possesses a high level of expertise on the relational database model; she will easily understand the principles of the semantic view creation.

2. The DBA is supposed to be the most knowledgeable person on the data objects stored in a particular database and the relationships between data objects. This will help her to create the most accurate semantic view of the database.



is replaced by:

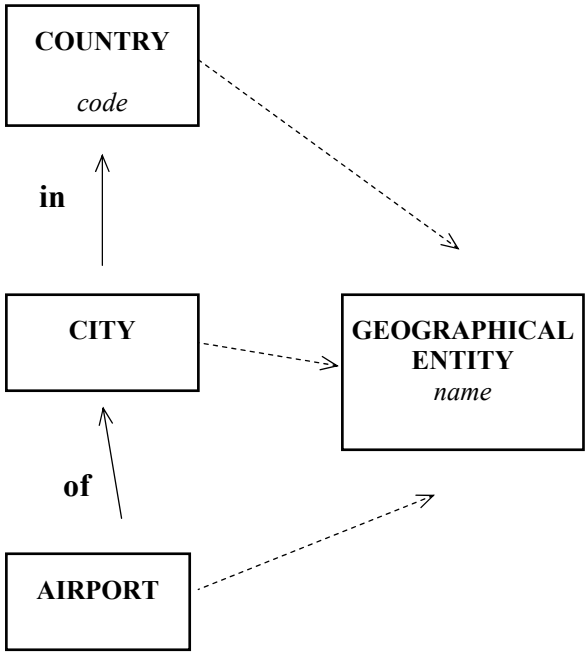


Figure 12. Application of step (8) of the algorithm to Geography database.

3. It is the responsibility of the DBA to create and maintain userviews for different groups of database users. A semantic view of the relational database may be treated as a userview describing the semantics of the whole database. For security or other reasons the DBA may decide to create different semantic views to tailor needs of different database applications or users. If we take the Geography database as an example, a separate userview having only categories CURRENCY and COUNTRY may be created for a currency conversion application. Another userview with only categories AIRPORT and CITY will serve the application, searching nearest airports for the city, and so on.
4. Overall, the DBA is responsible for maintenance and correct functioning of the database tools. The proposed system implementing semantic wrapping technology, namely Semantic Wrapper, may be viewed as an additional database tool in hands of the DBA and the users.

Another problem which needs to be discussed is the extent to which the Semantic Wrapper tool should allow the DBA to modify semantic views. No questions arise when the DBA drops categories and relations from a semantic view of the database, say to create a separate view for a particular group of users. But step (8) of the algorithm allows the addition of new categories to the semantic views in the role of supercategories for the existing categories. This may eventually lead to creation of additional relations on the semantic view as well since the subcategories may have the same relation(s), which will be propagated to the supercategory. Thus, there is a possibility that the DBA will be able to create arbitrary categories and relations that are not related to any data objects in the

relational database. This in turn may lead to incorrect semantic queries against the relational database thus affecting the whole integrity of the semantic wrapping process.

This problem is addressed by enforcing the following rule: At any step of semantic view creation process, Semantic Wrapper keeps the mapping information between the relational database tables and the corresponding semantic view categories/relations intact. If modifications of the semantic view will damage the integrity of the mapping information, it is not allowed by the system.

With respect to the mapping information between the relational database and its semantic view, it is also used to translate semantic queries to the analogue relational queries each time the user or application poses a query against the database. It is natural to have a database or other easily accessible and updateable storage to store this mapping information.

2.1.6. Results

In order to adopt and successfully use the semantic wrapping approach in the relational database environment, the DBA needs to define a semantic view for the database. This effort depends on the degree of complexity of the database and the quality of its existing documentation. If the database is well documented and has a conceptual schema in place, then the effort is relatively small. Otherwise, the DBA needs to reverse engineer the existing database schema. As a byproduct, the reverse engineering effort creates documentation on the database schema, which facilitates database use, improves its reliability, helps in training personnel, and allows the executive managers to better understand their data. With translation from the relational database schema to the

semantic database schema and with Semantic Wrapper we propose an algorithm and tool that mechanizes the reverse engineering and semantic wrapping process to the degree possible, and provides a means for the DBA to make manual improvements.

2.2. Semantic Wrapping Tool for Internet

2.2.1. Summary

Semantic Wrapper is a set of wrapping tools that provides relational databases with a semantic interface based on the Semantic Binary Object-oriented Data Model (Sem-ODM), which is superior to other data models in expressive power and ease of use. This section presents the system architecture of a Web application for accessing relational databases through the Internet via a Semantic Wrapper.

2.2.2. Background

In the past decade, various databases have been developed for commercial and research purposes. As a result, various data resides distributed in different databases that are maintained and developed independently. With the rapid development of distributed computing techniques like Internet computing, the demand for enterprise-level cross system data sharing in a global distributed environment becomes more and more pressing. The research being performed on heterogeneous databases satisfies this need by offering users a global transparent heterogeneous database layer sitting on top of all related distributed databases, thus providing an interface for users who would like to work on what appears to be single database instead of several databases with different data models in a distributed environment [15].

At the High Performance Database Research Center (HPDRC) [16], we undertook the Heterogeneous Distributed Database (HDB) project [17], which aims at integrating information from a variety of distributed heterogeneous data sources including structured data sources (e.g., relational databases, semantic databases) and semi-/un-structured data sources like information from the World Wide Web (WWW or Web). In our solution, the Semantic Binary Object-oriented Data Model (Sem-ODM) [8] developed by HPDRC is selected to build the heterogeneous database layer. Sem-ODM has been shown to be very efficient when compared to relational database on a class of applications that utilize its semantic features [18]. Most popular commercial databases, such as MS SQL server, Oracle, and IBM DB2 are relational databases (RDBMS) developed based on the Relational model [19]. As part of the HDB project, we developed a wrapper (Semantic Wrapper) [15, 20, 21] for relational database systems, which provides an interface to relational databases that is similar to a Semantic Database (Sem-ODB) [22, 23, 24].

Semantic Wrapper is a set of tools developed for accessing relational databases using the Sem-ODM data model and Semantic SQL query language (Sem-SQL) [14]. Our current Semantic Wrapper consists of three major components: Semantic View Constructor, Sem-SQL Server and SDB JDBC/ODBC Driver. We will address each component in the next section.

With Semantic Wrapper, a user views the united database through a semantic schema and accesses it via the standard interface of Sem-ODB. The advantages of such interfaces include friendlier and more intelligent generic user interfaces based on the stored meaning of the data, comprehensive enforcement of integrity constraints, greater flexibility, and substantially shorter application programs [21]. In addition, the semantic

interface supports a client-server architecture, thus providing a client-server interface for accessing a centralized relational database like MS Access.

This section presents the system architecture for a Web application that provides semantic access to relational databases. Users view the relational database through a semantic schema and access it via a standard interface of Sem-ODB, which lets users develop the relational database applications by taking the advantages of Sem-ODB. As part of the system, we implemented the Web Sem-SQL query interface for relational databases.

2.2.3. Web Access to Data Sources via Semantic Wrappers

System Architecture

In this section, we will present the architecture for Web access to relational databases via *Semantic Wrapper* and discuss each module in the architecture.

Figure 13 shows the overall system architecture for accessing relational databases via the Web with Semantic Wrapper. We used a three-tier Web architecture [25] to develop our system.

- The first tier is the Web browser, which serves as our universal client. In the first phase of using the application, an HTML front-end was used for starting the application, which will provide a GUI for user-input and the display of database query results.
- The second tier of the application is implemented with a Web server capable of executing the program serving the server (e.g., Java Servlet program in our system).

- The third tier is composed of our back-end database server, on which Semantic Wrapper facilities are installed for accessing the actual relational database server that stores the information. The query application has to talk to Semantic Wrapper to get the desired information from the actual relational database.

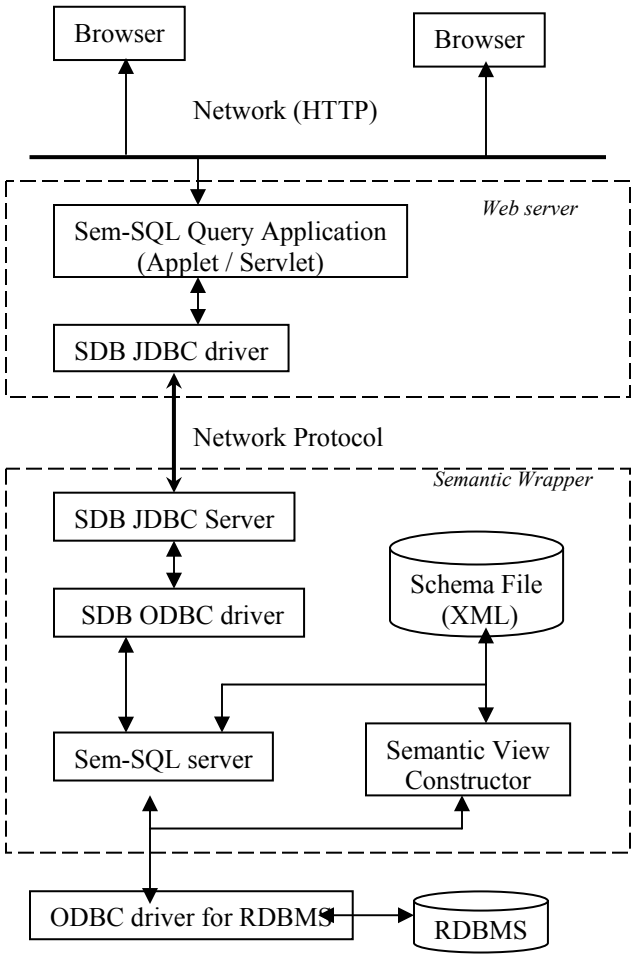


Figure 13. Overall Architecture of Web Semantic Access to RDBMS

We will briefly introduce each module in the architecture as follows.

Browser: A browser serves as the front-end client that loads the Web pages and programs from Web server to take user-input and display the query results.

Sem-SQL Query Application: This module is implemented with Java Servlet/Applet technologies, establishing the connection to database, retrieving the data, and providing a GUI for user-input and the display of query results. In order to retrieve information from RDBMS, this module needs to talk to SDB JDBC server for Semantic Wrapper through standard JDBC interface [26]. We will address this module in detail in the next section.

Semantic View Constructor: Before making a Semantic SQL query for relational database, the user has to create the semantic view (i.e., semantic schema) for the target relational database. Using this constructor, the user can express semantic information for the relational database by creating a semantic schema for it. Complex semantic expressions such as inheritance and m:m relations can be constructed by this component. All the mapping information will be saved in an Extensible Markup Language (XML) [27] file, which will be used to translate the semantic SQL statement into a standard SQL statement later on. This constructor assists the DBA to make intelligent design decisions in creating a complex semantic schema and also keeps the meta-data consistent.

Schema File: The schema file stores both the semantic and relational schemas along with derivation rules for query translation. XML technologies are adopted to save the schema information. By using XML technologies, we were able to easily capture complex semantic information along with the semantic schema from relational databases [28].

Sem-SQL Server: This component is the central processor of Semantic Wrapper, translating Semantic SQL queries (based on the semantic schema) into equivalent relational SQL queries based on the relational schema of the commercial RDBMS. It uses

derivation rules as well as semantic and relational schema information stored in the XML file for this purpose. The relational SQL queries are transmitted to the RDBMS using ODBC interface. The Sem-SQL Server also provides the fundamental programming API for the JDBC/ODBC interface.

SDB JDBC and ODBC: This component provides the standard JDBC and ODBC interfaces, which support the Semantic SQL language, thus giving users the flexibility to develop their own applications on top of either interface. Our JDBC interface is designed to support a server/client structure, which means users can make semantic queries against any remote semantic data source through a semantic JDBC interface. Figure 13 illustrates the server/client structure of the semantic JDBC interface. Using the JDBC/ODBC interface, we provide a competitive solution for developing RDBMS applications while making use of the advantages of Semantic Databases.

Performance

As the data flow shows in Figure 13, in order to map the semantic information into actual relational data, the user application has to go through two ODBC interfaces. One is SDB ODBC interface through which Sem-SQL is translated to standard SQL. The other is the ODBC interface for the targeted database, which is used to retrieve the actual data from the relational database. One problem arises in retrieving information from the relational database: the query execution performance of the application might be lower than that of the application accessing the relational database only through its own standard interface. This means that users can only take advantage of the semantic interface via Semantic Wrapper at the expense of some minor performance losses.

However, the advantages in use and development allow the high performance Semantic Wrapper module to be applied in commercial relational database applications.

In the next section, we will focus on how to develop the Web query application that provides an online Sem-SQL query interface for accessing relational database via our Semantic Wrapper.

2.2.4. Web Sem-SQL Query Application

In this section, we introduce our online query application architecture. We used Java Applet and Servlet technology to implement the system.

Related Work

Applets are Java programs that reside on a Web page and are dynamically loaded into the browser and executed when the Web page is viewed. Applets are discarded when the browser exits the Web page [29]. There are the following advantages in adopting applet technology for the implementation of the query interface program:

- The program enables a Web page to present dynamic content that can interact with the user. In our system, the applet program is used for user-input and displaying results.
- With any operating system, there is no need to install anything on the local machine to run the program.
- To start the program, the user only needs to browse the Web page that embeds the applet program. It is simple to run the program.

On the other hand, there are some security restrictions that restrict applet program access to some local and remote resources [29]. One of the restrictions is that applet

programs cannot normally make a network connection to any host other than the Web server that served the applet [30]. One way of working around this restriction is to use a server application that executes on the applet's host. In our system, we used Java servlet [31] technology to develop the server application serving the applet. The servlet program can make the database connections, send queries to the database server, and retrieve the query results to the Applet.

In the rest of this section, we will address how to develop our query application.

User Interface

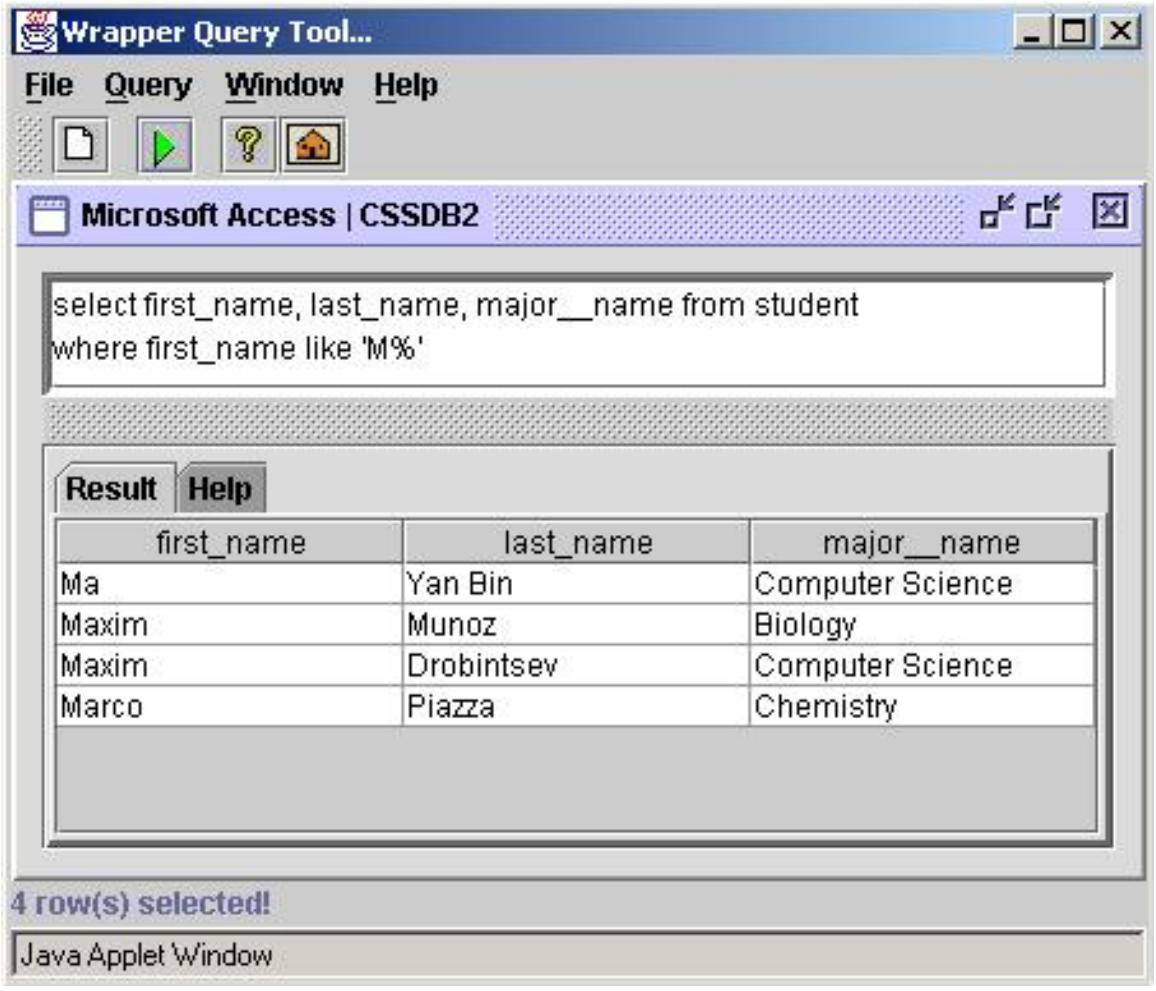


Figure 14. Main GUI for Web Sem-SQL Query Application

Figure 14 shows the main front–end graphical user interface (GUI) for our online Sem-SQL query application, which provides a friendly interface for inputting semantic SQL statements and displaying query result in a table format.

System Architecture

As illustrated in Figure 15, we have implemented the Web query application via two Java programs: the Applet and Servlet programs. Both programs originally reside on the Web server: the Applet program will be loaded into browser and serve as the client side and the Servlet program will run on the server side. We briefly discuss the data flow of the query interface below.

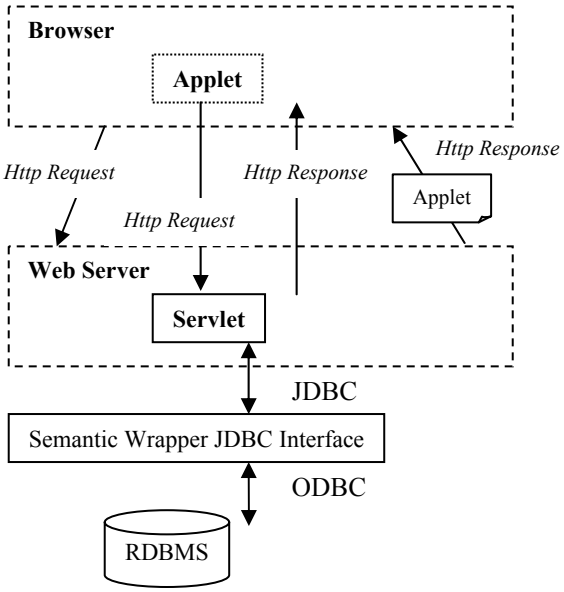


Figure 15. Architecture for Sem-SQL Query Application

1. When the user browses the Web page that embeds the Applet program, the Applet program is loaded into the Browser and is invoked on the client machine.
2. The Applet provides a GUI for user input of SQL statements and the necessary data source information such as the destination IP address, data source name and

username/password. Applet will pack all of the user inputs as a HTTP message and send them to the Servlet program on the Web server.

3. After the message is received from the Applet, the Servlet will unpack the message and execute the query using the Semantic Wrapper JDBC interface. The Servlet will pack the query result as a HTTP message and send it back to the Applet.
4. Finally, the Applet will unpack the message and display the query result.

Performance

Figure 15 shows the architecture to develop a general online query application for accessing semantic wrapped (wrapped with Semantic Wrapper) relational databases via any machine that has access to the Internet. Our implementation of the system shows that the given architecture is practical and allows efficient development of query applications. Our query application can be used as a general query tool to allow Web users to access their relational databases as long as they create a wrapper for their relational database using our Semantic Wrapper.

2.2.5. Results

In this Section, we discussed the architecture for our Web application that provides a query interface for accessing relational databases via the Internet. This interface was developed on top of Semantic Wrapper, which provides friendlier and more intelligent generic user interfaces for relational databases. Our approach of using Applet and Servlet technology together is shown to be a practical and convenient way to develop a high performance query interface.

2.3. Data Extractor Wrapper System

2.3.1. Summary

We describe a Data Extractor system for retrieving data from Web sites. This system represents Web sites as data tables that can be both integrated in a heterogeneous database framework and serve as data sources for standalone applications. We address issues involved in selection and analysis of Web data sources, and construction of wrappers for them. Data Extractor system design and implementation issues are discussed.

2.3.2. Background

The explosive growth of the World Wide Web in recent years has provided users worldwide with unprecedented volumes of information. This wealth of information is, however, underutilized, because mechanisms for accessing it are limited. Users can browse Web pages, search for information and perform a predefined set of transactions on Web sites, but in the majority of cases, generated data is provided only for visual consumption. No convenient mechanisms exist for analysis and processing of the found data, because users can only read what is shown in Web pages. There is no way for users to, for example, create complex queries to a travel agent's Web site—for each such query a special program would have to be implemented inside the agent's Web server. Even if the queries that are available meet user's needs there is no way to work with the data that they return. The stock quotes that are available on the financial Web sites most of the time cannot be imported into a spreadsheet for further analysis. The sites that do provide data in an easy-to-process form, such as XML, are still rare. Although this is expected to

change in the future, large volumes of data are likely to remain in HTML for quite some time.

In this section we describe a Data Extractor system that provides a mechanism for accessing data scattered across Web sites and using it in a variety of applications, such as database management systems, spreadsheets, and analytical tools.

2.3.3. Data Extractor

Heterogeneous database integration

The majority of existing methods for accessing data on the Web specialize in extraction and purification of data, and channeling it to external applications and users. Some researchers ([32, 33]) approach Web data extraction as a part of a bigger problem of heterogeneous database integration. Such integration would let users access resources of multiple databases of different types via a unified interface. It will also allow them to pose queries over a unified schema of multiple data sources.

In our research we are also investigating ways of integrating data extraction into a heterogeneous database system. We developed MSemODB [20], a heterogeneous database management system, whose general architecture is shown in Figure 16.

The system is using the Semantic Binary Object-oriented Data Model (Sem-ODM) [8] for data representation and the SQL query interface for communication between its components. Sem-ODM combines the simplicity of relational and power of object-oriented data models. A major advantage of this model is its ability to use the standard SQL-92 query language interpreted over Sem-ODM schemas (called Semantic SQL) in a variety of relational and object-oriented databases. This feature makes

MSemODB compatible with a number of existing tools developed for standard SQL. The communication between components in the system is done in CORBA, which is an efficient cross-platform and language-independent communication medium.

The main module that controls execution in the system and flow of data is Query Coordinator. Its function is to collect database schemas from all member databases and dispatch user queries to the appropriate database sites. It gives a common user interface to all the databases in the system. Through it, users enter queries using SemSQL and view resulting datasets in a single data model. Query Coordinator consists of Integrator & Knowledge Reconciliator, Schema Catalog and Query Dispatcher. Schema Catalog collects schemas of individual relational, semantic and Web database sites. Integrator & Knowledge Reconciliator coordinates schemas to resolve conflicts. Database administrator can use it to manage and modify Schema Catalog, and to introduce new relations that are not apparent from mere collection of member schemas. Query Dispatcher optimizes and executes queries. It decomposes queries posed by user into sub-queries based on the knowledge stored in the Schema Catalog and dispatches these sub-queries to appropriate sites for execution. When the results are available, it assembles them and presents them to the user.

The database sites are exposed in the system through their individual Semantic SQL and Semantic Schema modules. For the Relational Site a special knowledgebase and a reverse-engineering tool facilitate relational-to-semantic schema translation and storage. The majority of translation tasks are performed automatically. The database administrator can step in and make modifications and enhancements to the schema after the automatic conversion is completed. The Semantic SQL module of the Relational Site

implements an algorithm which automates conversion from Semantic SQL queries to relational SQL queries. With this functionality, virtually any RDBMS available today can be integrated with MSemODB.

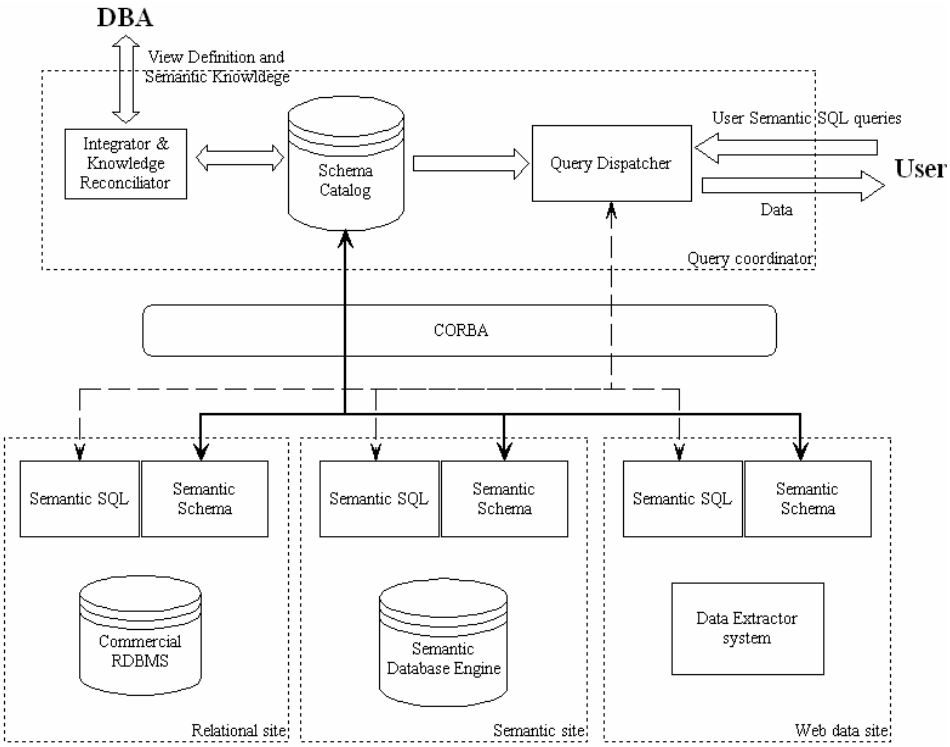


Figure 16. MSemODB Architecture

In the Semantic Site that wraps around Semantic Databases, integration is far more natural. The Semantic Object-oriented Database engine (Sem-ODB) already has Semantic Schema and Semantic SQL query facilities implemented. This database system is a multi-platform, distributed, fully functional client-server implementation that is suitable both for standard database applications and for large-volume data and spatial data applications.

Web Data Site is built around a Data Extractor system and provides a framework for data extraction from the Web.

Data Extractor architecture

The Semantic SQL and Semantic Schema modules access Data Extractor through a standard interface that allows schema discovery, query execution and data retrieval. Together with these modules the system works as an integral part of an MSemODB system, capable of executing SQL queries and returning result datasets. Using Data Extractor external applications pose queries to Web sites and extract data from them. Data Extractor presents extracted data in two-dimensional tables that can be further processed and returned to the user.

Data Extractor system consists of several components (see Figure 17).

- *Wrapper Controller*. This is the main component, whose responsibility is to control and coordinate execution of other parts of the system. It is the entry point for communications with the Data Extractor from the outside. It loads, executes, and controls behavior of wrappers and redirects data that they generate to the user. It accesses the knowledgebase to become aware of the configuration and schema changes in the system.

- *Knowledgebase*. This module stores system configuration information. It contains data on what wrappers are available, where they can be loaded from, what parameters are required to execute them, and what kinds of data they generate.

- *Wrappers*. Wrappers are lightweight modules that execute in response to user requests. They extract data from Web sites and return it to the user in an easy-to-process form.

- *Data Extraction Library*. This library contains extensive network access and HTML processing functionality. This functionality simulates behavior of the Web browser, allowing wrappers to traverse Web sites and extract data from HTML pages.

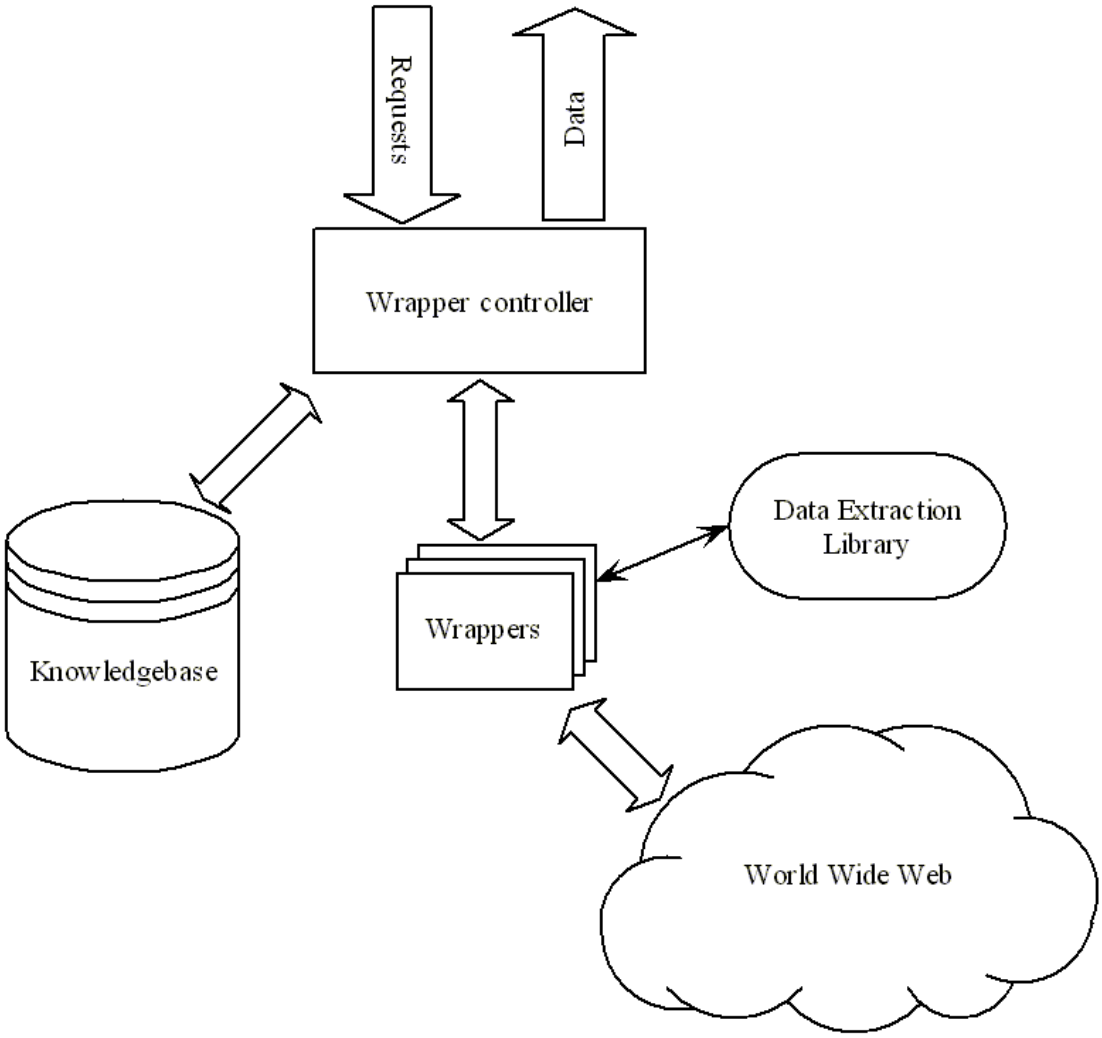


Figure 17. Data Extractor System Structure

Data Extractor is implemented in several versions: *standalone*, *embedded,* or *mobile*. As a *standalone* server it serves clients through a simple browser-based user interface, executes user queries, and delivers raw or processed data directly to the user. When *embedded* inside another application (as is the case with the MSemODB

framework), Data Extractor acts as a data provider for that application. A lightweight *mobile* implementation of Data Extractor is shipped to the client side over the Internet and is executed there on behalf of the user.

2.3.4. Wrapper Construction

Wrappers in Data Extractor execute on behalf of users and extract data from Web sites. Wrappers essentially simulate a user working with the site through the Web browser. They fill out and submit forms, "click" on links, or find data of interest inside of pages. To support this behavior, special functionality was developed that emulates browser interaction with the Web site. It allows us to create and play back a "scenario" of user navigation through the site.

The process of creating a wrapper for a Web data source consists of multiple steps. It includes *source selection, site analysis*, and *wrapper implementation.*

Source selection

Selecting a Web site as a data provider in some applications may become a complex task by itself. Cases when only a single Web site is a source of necessary information are actually quite rare. Stock quotes, airline schedules and weather information—the kinds of information that are needed in business applications—are usually provided by dozens of sites on the Web. As a result, selecting a source of information often becomes the first step in generating a wrapper.

There are many factors, that have an effect on the decision to select a particular Web site as an information provider, including site performance and availability, data

completeness, data distribution (how few or how many Web pages must be retrieved to extract a data set), and others.

Web site analysis

Once a Web site is selected for data retrieval (but before a wrapper is created for it) a thorough analysis of the site must be performed. Good analysis usually makes the subsequent programming effort easier and the resulting code more efficient, less bulky, and easier to maintain. There are many properties and features of the Web site that the analyst has to identify in her research. All of them will influence how the resulting wrapper will behave and how effective it will be:

- *Starting page/deep linking.* Identifying the starting page where the data retrieval process should begin is very important, as it allows the wrapper to skip unnecessary intermediate pages and go straight to data pages, or *deep-link* to them. This will primarily work for *static* URLs that do not change because of context. *Dynamic* URLs (URLs that contain changing parameters) can also be used for deep linking if their structure is well known. The wrapper can substitute parameter values in such URLs to get different data depending on user request.

- *Page identification.* Before the data extraction can actually take place, the wrapper has to make sure that the page containing data was actually retrieved. Sometimes the page that is expected does not arrive. Network and site problems may generate HTTP error pages in response to legitimate requests. Site structure and layout changes could also cause generation of error pages. Finally, the query that the wrapper poses to the

Web site might bring no results. For reliable wrapper execution it is important to check for all of these signs before attempting to perform data extraction.

- *Location markers.* One of the ways to locate data inside HTML is to search for it with reference to some visible *location marker* (or "landmark" –[34, 35, 36]) inside the document. A location marker is any HTML element or group of elements that is uniquely identifiable and that is located close to the data that we are extracting. Once the wrapper finds such a marker inside the document, it could then "move" through the page relative to marker's location in order to find data.

- *Data identification markers.* These markers are unique characteristics of an HTML document that point to data elements and delimit distinct data records. Such markers are usually unique only inside some part of an HTML document and they must be used together with location markers and other search techniques. They can exist in many different forms, but HTML elements that highlight parts of the document are most commonly used for this purpose. Such elements include tags that specify fonts, paragraphs, record breaks, table cells and colors. Some of the HTML elements that are not visible in the browser are also useful for data identification. For example, comments and nonprintable symbols inside text records do not manifest themselves to the user in any way, but the wrapper can split data into records using them as delimiters.

- *Tree search.* One of the most powerful ways of locating data in HTML documents is through searching. We can search for a variety of pieces of information, such as tags, tag attributes, their values, text elements and comments. For the purposes of a

particular application we can search for exact strings or substrings, searches can be case-sensitive or case-insensitive.

In Data Extractor, HTML documents are stored in the form of *trees,* similar to tree structures in [37]. In a tree structure, additional types of searches are possible. Searching can be done in the entire tree or in any of its subtrees. This means that the search can start from the root element or from any element inside the tree structure and only affect descendants of that element. This is a useful property, because it allows us to localize the search to specific logical parts of the HTML document, ignoring the rest. Additionally, we can search *linearly*. In a linear search, the document is treated like a flat stream of HTML elements. The search starts from a given position in the document and continues until its end or until the element is found. This is different from subtree search, because a subtree search finishes when the element is found or when every node in the subtree is visited by the search routine.

The decision to select either type of search depends on application needs. For some applications, locating data inside isolated portions of the document is important (e.g. for searching inside tables). For other cases (like searching for location markers) it is easier to think of the document as a flat file and search linearly.

- *Paths.* The simplest yet the least reliable (in terms of long-term stability of the wrapper) way to locate data inside a page is by specifying a *path* to it. A path is an ordered set of nodes in the document tree that we have to traverse to reach our destination. For example, if, in order to reach a particular node in the tree, we need to start at the root element, go to the second child of that element, then go to its first

child, and, finally go to the third child of that node then the resulting path will be encoded as {2,3,1}.

Unfortunately, this approach is the leading cause of wrapper failure in the event of site changes. If a single node in the path is changed the whole path becomes invalid, requiring wrapper modifications.

It might be tempting to use paths as the only method for locating data inside HTML because of their simplicity. Paths are also much easier to implement in tools that assist the analyst or developer in building wrappers. It is better, however, to concentrate on searching techniques, or to combine searching with short paths that do not originate at the root node, because this will improve wrapper robustness.

- *Multipage data.* The absolute majority of sites that provide large volumes of data dispense it in portions, showing it to the user through sets of linked pages. Data extraction of multi-page results is done continuously. When the first page of results is processed, the wrapper follows the link to the next page. When that page is loaded, data extraction is done and the process repeats. The data that is extracted is returned as a single data table or data stream to the process that executes the wrapper.

- *Parallel page retrieval.* Taking advantage of a *parallel page retrieval* technique can significantly increase wrapper performance, as page downloads are done by multiple concurrent threads of execution. This technique works well when data is distributed across multiple pages and links to several such pages are accessible from a single location, such as an index or summary page.

- *Hidden data extraction.* Data extraction is usually done on textual portions of the HTML page. To be effectively communicated to the user, data has to be highly visible and occupy a prominent position inside the page. There are times, however, when data can be extracted from other, *hidden* parts of HTML. Good candidates for this are comments, tag attributes and their values, URLs, and even scripts. Useful data might include IDs, prices, addresses, phone numbers, and other pieces of information. Extracting this information can save time on unnecessary page retrievals.

- *Scripting simulation.* Web sites are often *script-intensive*—in other words, they make wide use of JavaScript and VBScript for a variety of tasks. Some of these tasks, such as various visual effects, are not of particular interest to data extraction. Others, such as navigation, or form validation and submission, are important. Script execution is beyond the current capabilities of the Data Extractor system, but a wrapper can be programmed to *simulate* some of the actions of the script without having to execute it.

- *Weak binding.* By *binding,* we mean wrapper reliance on particular features of the Web site. In order to identify and extract data from an HTML page a wrapper has to look for certain features and markers inside the page. Such binding has to be *weak* so a wrapper could withstand minor Web site changes without having to be rewritten or corrected.

  This issue is closely related to all of the site analysis and description techniques discussed so far and must be taken into account when applying them. Trying to find a balance between reliable data identification and weak binding has proved to be rather challenging. It is hard to come up with universal recipes on how to do this optimally,

as these two tasks are inherently contradicting. The weaker the binding, the less reliable data identification within the site is. The stronger the binding, the better data identification is, and the greater the chance that the wrapper will not withstand the next site change.

In the absence of a clear-cut solution to this problem, we suggest that the site analyst try to identify the smallest set of site features that will help pinpoint data location inside the site. When selecting these features, the analyst also has to make sure that they are *content-dependent* (could be searched for) rather than *structure-dependent* (are located at specific positions inside markup trees), as the latter is more likely to change. When such set of features is identified, it can be used to create a wrapper that is more tolerant to site changes.

Wrapper implementation

When the Web site analysis has been completed, the next step is to implement the wrapper. In the Data Extractor project, wrapper code is implemented in Java using the functionality of the Data Extraction Library. The results of site analysis that describe steps to traverse the site and acquire data are implemented in Java and debugged using the *wrapper executor*. Wrapper executor is a Java application that, through a simple interface, allows a programmer to specify wrapper parameters, execute them and step through the wrapper code in any Java development environment. Debug information about network communications and returned data is given as feedback to the programmer during wrapper execution. When the wrapper is debugged and tested, it can be integrated into the Data Extractor system through the knowledgebase.

Data output: The data that is extracted from Web sites is either returned directly to the user or fed into the calling application that analyzes and processes it. For a wrapper to be integrated into heterogeneous and other database systems its interface has to act as a mini-database system that produces data in response to queries. Therefore, one of the major tasks of wrapper analysis and implementation is the definition of the structure of the wrapper's output, or *schema*. Schema description and registration is done through the knowledgebase.

We define a schema by specifying names and types for pieces of data, or *fields*, that the Web site provides. The wrapper is then programmed to output data using the field information defined in the schema.

In the Data Extractor system, wrappers can return data both row-by-row and in complete tables. In the row-by-row approach, a row can be output as soon as the data for it is extracted from the Web site. This simplifies concurrent execution of wrapper and applications that process and consume data. As soon as the wrapper has extracted and returned the first record to the calling application, data can immediately be filtered, modified or otherwise processed by that application. Building and returning a table might be the only option in cases when complete data for each row is not available until the end of the extraction process.

There are problems associated with schema definition. Data available on a Web site is usually taken from a data source or database internal to that Web site. Only a small portion of that database is displayed to the user. Knowledge about the database schema is not exposed: no information is given on how data is decomposed into tables internally or what relations exist between tables. Some fields (e.g. internal codes or product IDs) are

rarely shown to the user. Finally, the size of the data set displayed is often less than the one that is stored in the database. All of these factors make schema definition complicated.

Wrappers in the Data Extractor project return data in simple two-dimensional tables similar to the ones used in relational databases. There is a single table defined for each wrapper. In the future we plan to use more complex data structures and generate multiple tables from a single wrapper.

Wrapper parameters: Some of the advantages of wrappers lie in their ability to shield the user from Web site complexity, and to generate data in response to requests made through a simple interface. In order to be truly useful, a wrapper has to be able to execute a class of queries, not just a single query (i.e. a wrapper that extracts weather information for major cities is more valuable than wrapper that can only give weather for New York City).

In the Data Extractor system, wrappers can have *parameters* that let users modify their behavior. In the example above, a wrapper would receive the name of the city as a parameter and then pass that information to the Web site to get weather for that city.

2.3.5. Implementation

Data Extraction Library

The purpose of the Data Extraction Library is to provide Web document retrieval, parsing and data extraction functionality for wrappers, ensuring full interoperability with any Web site and full browser simulation.

The Data Extraction Library provides four main groups of functionality:

- *Page retrieval functionality*. Fast and reliable page retrieval from the Internet is crucial to wrapper operation. In Data Extraction Library page requesting and retrieval is done through sessions. A session is a conversation with a Web server, the result of which is a stream of data. This data can be read as a stream or (when data is actually an HTML page) can be converted into a document. Sessions are more than simple HTTP request/response pairs—they provide rich functionality for building requests and modification of parameters (similar to filling out forms in browsers).
- *HTML processing functionality*. The contents of the retrieved HTML pages are stored internally in the form of a tree that consists of markup, or tags, and text elements. Each HTML page that is retrieved from the Web is automatically converted into an HTML tree. The parsing and document storage functionality was built to parse HTML and any other markup language that is based on SGML. It is, for example, capable of storing and processing XML and other XML-based formats. HTML processing functions give wrappers access to every detail of the document and feature powerful tree search and manipulation capabilities. Special functionality allows the creation of page downloading sessions from HTML forms and links, and automates data extraction from hierarchies of tags and HTML tables.

The HTML parser that we implemented is not a general-purpose parser because it is lacking the validation mechanism. The validation functionality is not necessary in Data Extraction Library, because we only need to store documents in memory, traverse them and search them for information. Tags are treated equally regardless of their type. This makes the parser fast, more robust (because it does not refuse to

process documents that are syntactically incorrect), and able to build a tree for almost any document.

- *Data representation functionality*. When data is extracted from the Web, it is shipped to the consumer. There are two ways to store data in the Data Extraction Library—in rows and in tables, with tables being collections of rows. This model allows us to create wrappers that generate data in either block or stream fashion. For the majority of wrappers, stream is the preferred way to output data because it allows portions of data to be processed by a higher-level application while the rest of it is still being extracted.
- *Wrapper interface functionality*. Wrapper interface functionality provides a simple communication and control mechanism that simplifies the implementation of wrappers and transmission of data. It allows the calling process to fully manipulate wrapper execution, pass input parameters into the wrapper and control the flow of data out of it. Through this interface, a wrapper reports errors to the Data Extractor system, which tells systems administrator that the wrapper needs to be updated.

Challenges

Some difficulties were encountered while implementing the functionality for HTML parsing and page retrieval.

- *Syntax errors*. In the course of building wrappers for a variety of sites we found a large portion of HTML pages to be syntactically incorrect. Shockingly, an estimated 90% of all Web pages on commercial Web sites we have analyzed so far have contained syntax errors. The high quality of modern Web browsers is partially to blame for this. In order to accommodate the widest possible variety of Web sites and

make an effort to display any Web page, no matter how badly structured, the browsers were made extremely forgiving. HTML page authors can miss or mismatch closing tags, put end tags in the wrong order, not close comments or make other mistakes—and browsers will not alert her to the problem.

Syntax errors, however, had little impact on the work of our parsing functionality. A wrapper could not be built due to hopelessly incorrect HTML pages for only one site out of over a hundred we analyzed. For all other sites the parser did a satisfactory job of building markup trees out of pages. Such trees weren't always correct semantic representations of documents, but they provided a data structure that is adequate for traversing and information extraction.

- *Slowness of core network functionality*. Java is an interpreted language and this takes its toll on the performance of time-critical routines in the standard Java libraries. Slowness of the network functionality in Java contributed the most to the overall slowdown of the wrapper operations. In our tests between 40% and 70% of the overall wrapper execution time was spent connecting to Web servers, sending requests for pages and receiving pages. In comparison, only about 5-10% of the time was spent on parsing and processing, and the rest—on operations associated with data extraction and control of wrapper execution.

  Network operations were somewhat sped up when the standard Java HTTP protocol implementation was re-written using sockets and when other optimizations were applied as suggested in [38]. This way a lot of unnecessary operations were eliminated, making the implementation leaner, faster, and more flexible, accommodating redirects, cookies, and other protocols.

In the future we expect the slowness of the network functionality to remain one of the stumbling blocks for successful wrapper implementation. Slowness of such operations is not inherent to Java—the same operations implemented in C++ for comparison purposes performed only insignificantly faster. This decreased performance significantly reduces the usefulness of the applications written using wrapper technology because the speed of data set generation is slow and sometimes insufficient for a satisfactory user experience.

Data Extraction Scripting Language

As the Data Extractor project was progressing, two things soon became apparent. First, the majority of the Web sites which were analyzed and for which wrappers were implemented had a simple structure and did not need the full power of Java for data extraction. Second, maintenance of Java wrappers became cumbersome in some cases, where Web sites would change their structure once in three months or even more often, which in turn required changing the Java source of the appropriate wrapper. These observations initiated the work on a Data Extractor Scripting Language (DESL), a simple scripting language that will make fast definition of wrappers possible for the majority of Web sites.

We followed several requirements when we designed DESL. First, it had to be simple, expressive, and cover only the functionality necessary to extract data from HTML. It was not necessary to create another programming language similar in power to Java. Simplicity and expressiveness of the language improve understandability and reduce code size, thus reducing overall maintenance time. Java can still be used in cases when a site is too complex for a scripting language.

Second, we had to be able to generate scripts in this language using a user-friendly GUI. One of the plans for future development of Data Extractor is to build a GUI environment, where a designer could create and update wrappers quickly, using a WYSIWYG ("What You See Is What You Get") interface. A wrapper would be a result of a "macro recording" of the steps the designer takes through the Web site and the data extraction instructions would be generated in response to the data fields that the designer highlights inside the site. Because of the demands of the GUI, DESL must support "round trip engineering." This means that we should not only be able to generate the script based on designer actions, but also import it into the GUI afterwards and modify it if the need arises.

DESL syntax is currently being finalized. We will report on it as soon as the development and testing of the working prototype is completed.

Related work

Over the past several years, many researchers have studied ways for collecting and processing data available on Web sites. Although a fully automatic extraction and labeling of data on arbitrary sites is currently beyond the capabilities of computer science, assisted, learning-based data extraction has been quite successful in some systems [39, 40, 41].

An alternative approach to data extraction is based on custom wrappers built around Web sites. Wrappers are coded manually or generated through special wrapper-generating browsers, such as the ones described in [42], and [43]. In our research we used a wrapper-based approach because it gives the highest accuracy of results and can be used to cover virtually any problem domain. As many researchers have noted, these

advantages are sometimes offset by the costs of continuous wrapper maintenance. To help reduce those costs, we intend to further develop a two-fold approach to generating wrappers, where wrappers are written in a simple scripting language and wrappers for complex sites are written in Java.

Specialized languages dominate wrapper development, with implementations ranging from Prolog-like predicate logic [44, 45] to SQL flavors [34, 46, 47]. We have not seen wrapper implementations based on a general purpose programming language like Java.

Some of Data Extractor's features can be found in existing systems. These features include, in particular, regular expressions for searching, data extraction and navigation through HTML documents [46, 47, 48], tree representation of HTML documents [46, 35, 49] and form processing [34, 50]. XML query languages (e.g. XSLT [51]) have also influenced our research.

2.3.6. Results

In this section we have described a Data Extractor system that facilitates data retrieval from Web sites. Data Extractor plays an important role as a data provider for the MSemODB heterogeneous database system. The ability to access data not only from relational and semantic databases, but also from the unstructured Web data sources significantly increases the power of MSemODB and extends the range of applications it could be applied to. The solution is portable and can be used both as a standalone data provider and embedded into applications.

We defined custom functionality for implementing wrappers using a high-level programming language and a library of specialized data extractions. A description of a wrapper construction process and a number of techniques and guidelines for site analysis and wrapper design were presented. Data Extractor fully simulates user interaction with a browser and can fill out forms and extract data from complex sets of linked Web pages.

2.4. Mapping from XML DTD to Semantic Schema

2.4.1. Summary

The Extensible Markup Language (XML) is increasingly becoming a popular data exchange and representation format because of the need for enhancing data interoperability and exchangeability over the Web. Different approaches have been investigated on using database technology to store and access XML data. In this section, we explain in detail a unique and complete mapping scheme to map a DTD to a Semantic Schema in Sem-ODM. The key idea is capturing the meta-schemata of both the DTD and Semantic Schema, and then mapping the basic constructs of DTD to their counterparts in the semantic schema, while preserving the structure and semantic information of the DTD. We were able to easily and naturally capture complex semantic information with the Semantic Schema, thus smoothing the mapping process.

2.4.2. Background

As the World Wide Web is gradually becoming one of the most important communication media, the importance of improving the interoperability, easing the access to heterogeneous data sources and integrating data from different applications has

been brought to light. As a result, the World-Wide-Web Consortium (W3C) [52] has defined a language called Extensible Markup Language (XML) [27]. XML is a subset of SGML (Standard Generalized Markup Language), which was originally designed for the purpose of large-scale electronic publishing and is now becoming a popular data exchange and representation format [52].

One of the issues that a lot of researchers and software vendors have been focusing on is XML storage. To date, three different repositories have been used. They are relational databases [53, 54, 55], object-oriented databases [56, 57] and semi-structured databases [58, 59]. Among this work, a great deal of effort has been put on using relational databases because of its mature technology. The general approach of using a relational database as an XML repository is first mapping a DTD/XML-Schema to a relational schema and then loading XML data into relational databases (RDBMS) for further querying. However, no matter what mapping techniques are used, it has proved to be very hard to avoid the common problem of fragmentation when using RDBMS. This is because relational data model is a two-dimensional table and column-based structure, whereas the XML data model is a graph-based structure. In order to fit the graph-based XML data into the table based relational model and keep the mapped relational database in a certain normalized form, fragmentation is inevitable.

In this section, we describe an alternative approach for storing XML, which uses Semantic Binary Object-Oriented Database System (Sem-ODB) as the underlying XML repository. Sem-ODB was developed at the High-Performance Database Research Center (HPDRC) [16] and is based on a conceptual data model, the Semantic Binary Object-Oriented Data Model (Sem-ODM [8]). It has been successfully deployed for

highly complex applications such as applications intended for storage and processing of large amounts of earth science observations and the TerraFly Geographic Information System (GIS) [60].

Sem-ODM has features of Object-Oriented data models (such as inheritance, oid, explicit description of relationships, and a high-level data model) while maintaining the simplicity of relational data models in the sense of using simple constructs. It is more natural to preserve structure as well as semantic information using Sem-ODM than the relational model, as illustrated in section 4. In addition, set-valued attributes and nested structures of XML can be easily represented using relations (like associations in OO model) in Sem-ODM, thus avoiding the common problem of fragmentation when using RDBMS to store XML. Furthermore, by using relation and user defined oid (surrogate) concepts in Sem-ODM, semantic information such as element and document order in XML is more suitable to representation in a semantic schema than in a relational schema.

In this section, we describe in detail our approach of mapping DTDs to Sem-ODM semantic schemas. The basic idea is capturing the meta-schema of both DTD and the semantic schema, and then mapping the basic constructs of a DTD to their counterparts in a semantic schema, while preserving the structure and semantic information of the DTD. The mapping information is not hard-coded; rather it is generated dynamically during the mapping process and kept in a Sem-ODB for future query translation and reconstruction phases. The approach of capturing the meta-schemas of different data sources and storing the mapping information for future processing has been applied successfully in our SemWrapper project [20] and SemAccess project [21].

2.4.3. Overview of DTD and its Meta-Schema

2.4.3.1. XML and DTD

XML is a subset of the Standard Generalized Mark-up Language (SGML). It is a tag language, but users have the freedom of customizing the tags. Thus it can be used to define other languages. An XML document is composed of a stream of text nested within pairs of matching open and close tags. The tags are called Elements. Each element may have attributes describing the element and contain sub-elements in its body. Each sub-element can have cardinality of only one, at most one (?), at least one (+), or many (*) to describe how many times that sub-element can appear within the body of its parent element. In addition, it can be specified whether the sub-elements should appear in some order (denoted by a "," between sub-elements) or not (denoted by "|"). In this way, an ordered, nested and hierarchical document can be formed.

The structure and constraints of XML documents are described by a Document Type Definition (DTD) [61]/XML Schema [62]. Figure 1 is a DTD example that was extracted from [54] and is used as a running example in this section. The basic components of a DTD are elements and attributes. They are declared in the following form, respectively.

<!ELEMENT element_name element_content_type>

<!ATTLIST element_name attribute_name attribute_type default>.

DTD Simplification Assumption

To ease the mapping process, we transform complex DTDs to simpler, but equivalent ones before performing the real mapping. In the subsequent sections, we will assume DTDs have been simplified by rules in [54, 63].

In the following section, we will review the components of DTD and present the meta-schema that we have devised to describe a DTD.

```
<!ELEMENT book (booktitle, author)>
<!ELEMENT booktitle  (#PCDATA)>
<!ELEMENT article (title, author*, contactauthor)>
<!ELEMENT title  (#PCDATA)>
<!ELEMENT contactauthor EMPTY>
<!ATTLIST contactauthor authorID IDREF #IMPLIED>
<!ELEMENT monograph (title, author, editor)>
<!ELEMENT editor (monograph*)>
<!ATTLIST editor name CDATA #REQUIRED>
<!ELEMENT author (name, addr)>
<!ATTLIST author id ID #REQUIRED>
<!ELEMENT name (first?, last)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT last  (#PCDATA)>
<!ELEMENT addr ANY>
```

Figure 18. DTD Running Example

2.4.3.2. DTD Constructs

Note that notations of the Sem-ODM are used to illustrate DTD and Semantic Schema components throughout the rest of this section (see Section 2.4.4 for basic Sem-ODM concepts). Graphically, the rectangles represent categories. The dashed-arrow represents ISA links (inheritance/super-category subcategory relationships). The dashed-arrows point from sub-category to super-category. The attributes of a particular category are placed in the respective category rectangle with ranges placed after the ":" (semi-

colon). The thick arrows (i.e. non-dashed) represent relations between categories. The cardinalities and constraints of relations are represented inside parentheses.

The basic constructs of a DTD are elements and attributes, which are depicted in Figure 19 in the form of a Sem-ODM Semantic Schema. Therefore, mapping can be considered from these two perspectives, element-related and attribute-related mapping.

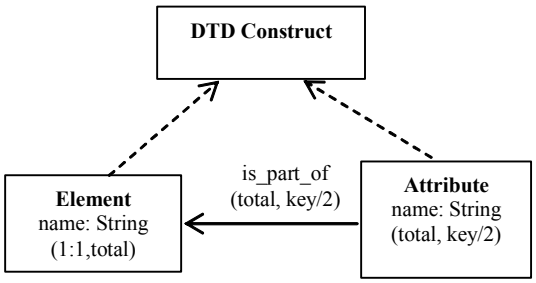


Figure 19. Sub-schema Representing DTD Construct

2.4.3.2.1. Meta-schema of Element

According to the content type of elements, elements can be classified into EmptyContent, AnyContent, PCDataElement, Mixed, and Sequence, as illustrated in Figure 20. Recall that we simplify a DTD so that it does not contain choice content elements.

An EmptyContent element does not have any text between its start-tag and end-tag. An example of such an element is contactauthor in the running example. Elements of AnyContent may contain any content, which might be an element. For instance, element addr is declared as ANY content in Figure 18. Thus in the XML document, addr can have arbitrary XML fragments. In our classification, we use PCDataElement to differentiate elements declared as #PCDATA type and #PCDATA in a mixed content declaration. Elements first and last in our DTD are examples of PCDataElement. Category PCDATA in Figure 20 is used to represent the character data in a mixed content. Mixed content

elements may contain character data as well as elements. For instance, the following example declares a mixed content element paragraph.

<! ELEMENT paragraph (#PCDATA* | figure*) >

<! ELEMENT figure (#PCDATA) >

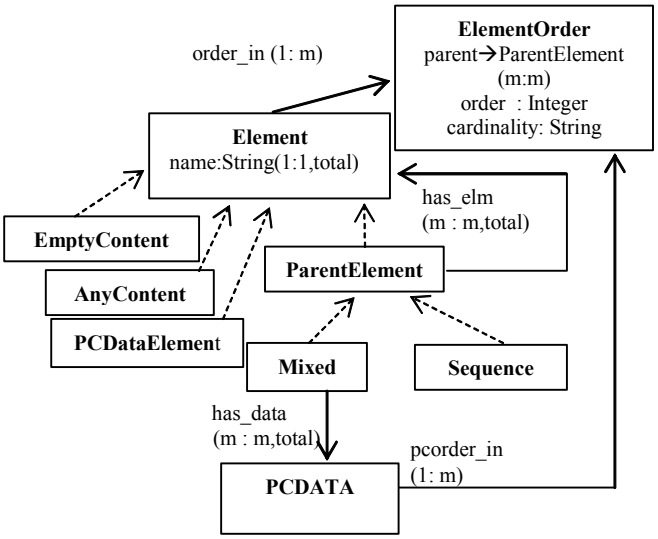


Figure 20. Sub-schema Representing DTD Element Meta-Schema

By our definition, element figure will be regarded as PCDATAElement and #PCDATA within element paragraph is an object of PCDATA category. They will be treated differently in the following mapping process.

A Sequence element is an element which contains child-elements and an ordering among the sub-elements. For example, element book in the above DTD example consists of booktitle and author. Booktitle must appear in front of author in any XML document that conforms to this DTD. Because mixed content and sequence elements may have sub-elements, we call them ParentElements in our categorization.

Category ElementOrder in Figure 20 is used to capture the ordering and cardinality of an element in its parent elements. Attribute parent keeps the information about the parent element, thus has the range of ParentElement.

2.4.3.2.2. Meta-schema of Attribute

An element may have a set of attributes associated with it. According to their types and defaults, attributes in DTDs can be further refined, as shown in Figure 21. An attribute can be of type CDATA, Tokenized, or Enumerate. Attributes can have REQUIRED, IMPLIED, DEFAULT, or FIXED declared as a default value.

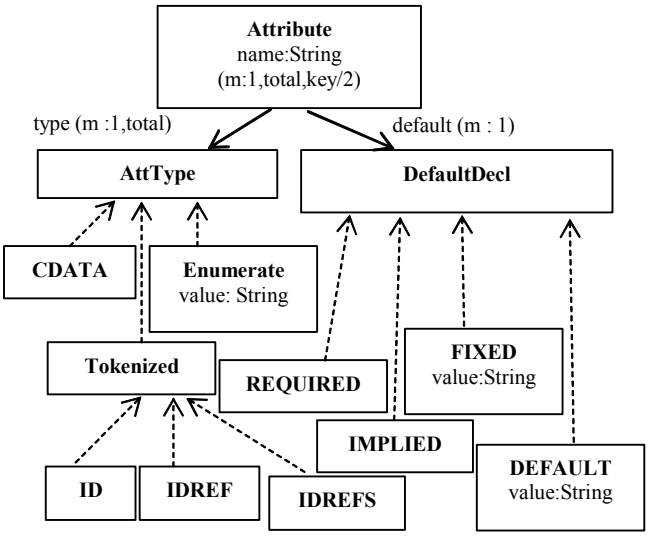


Figure 21. Sub-schema Representing DTD Attribute Meta-Schema

CDATA stands for character data, which is a string type. The REQUIRED option means that a value must be provided for the attribute for each element that this attribute belongs to. IMPLIED means no default value should be provided for this attribute. An attribute may have a default value in its declaration, which means if no value is provided for that attribute in the XML document, the default value will be assumed as its value.

The FIXED option determines that the corresponding attribute can only have a fixed value when it appears in XML documents.

Two special attribute types worth mentioning are ID and IDREF(s). The former is used to uniquely identify each element. It is much like the key concept in the relational model, except that each element can only have one ID attribute and no multiple attributes are used to identify one element. An example is attribute id in element author. Each value of id can uniquely identify an author. An attribute of IDREF must have a value matching the value of some ID attribute. The attribute authorID of element contactauthor is of IDREF type.

2.4.4. Overview of Sem-ODB Constructs

There are two constructs, category and relation, used to describe a Sem-ODM. The categories and relations in Sem-ODM model tables and the foreign key relationship between tables in relational databases, respectively. Categories are like Entities in the Entity Relationship (ER) model, except that the Attributes in the ER model are represented as relations in Sem-ODM. A Category can either be a Concrete Category or an Abstract Category. Concrete Categories are categories like String, Number, and Boolean, etc. Abstract Categories are categories composed of abstract objects; an example of an sbstract category is person or book. There can be binary relations from an abstract category, which is called the Domain of the relation, to another category, which is called the Range of the relation, in a Semantic Schema. Relations from an abstract category to a concrete category in Sem-ODM are called attributes in ER model. Relations from an abstract category to an abstract category are just like associations in OO model.

The constructs and relations between the constructs of a Sem-ODM can be illustrated in Figure 22.

With the help of relations in Sem-ODM, the nested structure of XML can be represented in a Semantic Schema. Sem-ODM also has the concept of Cardinality of relations, which can be used to capture the same concept in DTDs. There is no ordering concept in Sem-ODM. We would have to extend the relations in Sem-ODM with ordering to support this feature of XML. This is illustrated in the Relation category in the above figure in which an order attribute is added. In addition, an XMLType category is added as a Concrete Category in the above figure. This category is used to represent ANYContent in DTDs.

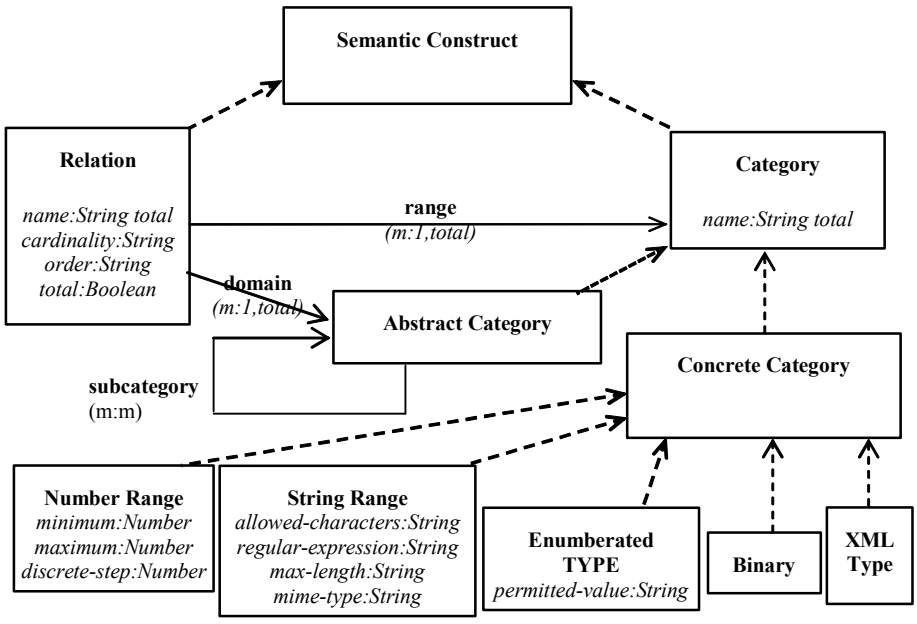


Figure 22. Subschema Representing Sem-ODM Meta-Schema

2.4.5. Structure and Semantic Mapping

In order to store XML in Sem-ODB, we need to map the constructs of a DTD to their counterparts in a Semantic Schema. During the mapping process, we preserve the

structure and semantic information defined in the DTD for future query translation and result construction. By doing structure mapping, the elements and attributes are mapped to categories and relations in Sem-ODB. Performing the semantic mapping ensures that the mapping process is complete, i.e., all the semantic information such as cardinality, nullable, and reference is set in the Sem-ODB.

2.4.5.1. Structure Mapping

Three kinds of structure mapping are performed: element, relationship, and attribute. To make the algorithm easier to understand, we introduce the following naming conventions:

1. Each element E in the DTD is represented by its name (e.g. element "book" is represented by book).
2. The category, which corresponds to element E in the DTD is represented as E in the Semantic Schema$_c$ (e.g. the category that's mapped from element "book" is represented as book$_c$).
3. Relations which are mapped from some attributes or elements E in the DTD are represented as E_R in the corresponding Semantic Schema.

In addition, for simplicity, we use the following notation to represent a relation r with domain category D, range R, cardinality c and totality t: r:D→R (c,t).

2.4.5.1.1. Element Mapping

An element of the DTD will be mapped to a category or inlined as an attribute of a category according to its content type and whether or not it is a child element. The following is the detailed description.

1. For every Element E of Sequence, EmptyContent, and Mixed type is mapped to a Category E_C in the Semantic Schema. For example, book in Figure 18, which is a sequence element, is mapped to a category called $book_c$ in the semantic schema.
2. For every Element E of AnyContent type,
   a. If E has only one parent element P and the cardinality of E in P is not * or +, then E is mapped to a relation E_R from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to XMLType, i.e., E_R: E_P →XMLType.

      For example, element addr is mapped to addr: author→XMLType.

   b. Otherwise, E is mapped to a Category E_C with a relation R from E_C to XMLType, i.e., R: E_C→XMLType

      For example, element test declared in <!ELEMENT test ANY> will be mapped to a category called test with relation data: test→XMLType.

3. Similar to (2), for every Element E of PCDATAElement type,
   c. If E has only one parent element P and the cardinality of E in P is not * or +, then E is mapped to a relation E_R from the category E_P, which is the category corresponding to the element P in the Semantic Schema, to String, i.e., E_R: E_P →String.

      For example, element first is mapped to an attribute of category name (first: name→String).

   d. Otherwise, E is mapped to Category E_C with a relation R from E_C to String, i.e., R: E_C→String.

For example, element title is shared by element article and monograph; therefore, it is mapped to a category called title with relation data: title→String.

The idea behind (2) and (3) is if E is shared by many elements, or it may appear in its only parent more than once, it should be mapped to a category and let all the parents set up a relation with it. Otherwise, let it be an attribute of its parent category.

4. For each PCDATA, map it as a relation from the category corresponding to the mixed element to String. For example, the #PCDATA in <! ELEMENT paragraph (#PCDATA| figure) *> will be mapped to data: paragraph→String(m:m).

2.4.5.1.2. Relationship Mapping

Some important structure information in a DTD lies in the implicit relationships between elements and their sub-elements and between elements and their attributes. Unlike DTD to relational schema mapping, in which this kind of information is implicitly expressed by the key and foreign key relationship or table-column relationship, a semantic schema explicitly represents them as a relation.

The mapping is described in the following.

1. If any of the above rules can be applied, then skip the following. For example, booktitle is a sub-element of element book. Since it's a PCDATAElement, it is mapped as an attribute of category book according to rule (3) in section 2.4.5.1. The following two rules are not applicable.
2. Every sub-element E_i of element E of Sequence type is mapped into a relation from E_c to E_{ic}, where E_c is the category corresponding to E and E_{ic} is the category

corresponding to E_i. For example, the relationship between elements book and author is mapped as a relation called book_author (see Figure 23)

3. Similarly, every sub-element E_i of element E of Mixed type is mapped into a relation from E_c to E_{ic}, where E_c is the category corresponding to E and E_{ic} is the category corresponding to E_i.

Note that in the above sections 2.4.5.1.1 and 2.4.5.1.2, if a relation is created in the Semantic Schema, its cardinality and totality have to be decided according to the rules in section 2.4.5.2.1.

2.4.5.1.3. Attribute Mapping

Attributes in DTDs are mapped to relations, whose domain is the category corresponding to the element that this attribute belongs to and whose range is a Concrete Category. The mapping rules are generalized as follows.

1. For <! ATTLIST E att CDATA>, where E is an element and att is its attribute of type CDATA. Attribute att is mapped to a relation att_R: E_C → String(m:1) (e.g. attribute name of element editor, see Figure 23).
2. For <! ATTLIST E att eval1 | eval2 | eval3 >, where E is an element and att is its attribute of type Enumerate. Attribute att is mapped to a relation att: E_C → Enumerate(m:1).
3. For <! ATTLIST E att ID>, where E is an element and att is its attribute of type ID. Attribute att is mapped to a user defined oid of category E_C when we don't consider the document order of XML. However, if we want to consider that order, we have to

map it to a relation att: $E_C \rightarrow$ String (m:1). (e.g. attribute id in element author, see Figure 23)

4. For <! ATTLIST E att IDREF>, where E is an element and att is its attribute of type IDREF. Attribute att is mapped to a relation att: $E_C \rightarrow E_{id}$ (m:1), where E_{id} is coresponding to the element that att is pointing to in the DTD. For example, attribute authorID of element contactauthor is of IDREF type and is mapped to a relation between contactauthor and author, i.e., authorID: contactauthor → author (m:1). Note that since IDREF(s) is untyped, the program could not tell which element that this IDREF attribute is pointing to. We have to explicitly set up the above relation.
5. For <! ATTLIST E att IDREFS>, where E is an element and att is its attribute of type IDREFS. Attribute att is mapped to a relation att: $E_C \rightarrow E_{id}$ (m:m), where Eid is coresponding to the Element that att is pointing to in the DTD.

Furthermore, if the default type of the attribute is #REQUIRED, the totality of the relation is total.

2.4.5.2. Semantic Mapping

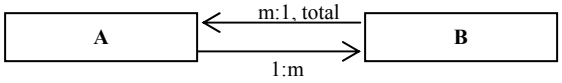
2.4.5.2.1. Cardinality Constraints

The cardinality of a sub-element in a DTD represents the occurrence of the sub-element within its parent element. To our knowledge, most known DTD-relational mapping algorithms either did not explicitly explain the mapping of cardinality or did so rather complicatedly as in [64]. It is rather easy in DTD to Sem-ODM mapping, since there is the cardinality concept, such as m:1, 1:m, m:m, 1:1, defined in the Sem-ODM.

For only one and at least one semantics in a DTD, we use total concept to represent the semantics.

The mapping of sub-elements cardinality is as follows:

1. Cardinality "only one" is mapped as a relation with m:1, total (e.g. element last)
2. Cardinality ? is mapped to m:1 (e.g. element first in Figure 18 has ? cardinality within its parent element name, thus it is mapped to first: name→String (m:1))
3. Cardinality * is mapped to m:m except when there is a cycle between two elements that have the parent and child relationship and the cardinality in one of them is only one or at most one. In the latter case it is mapped to 1:m. For example, suppose we have <!ELEMENT A (B*) > and <!ELEMENT B(A) > in a DTD. In this situation, we will have



   The reason that we map * to 1:m in this case is as follows. When only one or at most one semantics appears, we would derive a conflict if we still mapped * to m:m. For instance, suppose we map the above DTD to A→B (m:m) and B→A (m:1, total), then by B→A (m:1) , we derive B→A is 1:1, however A→B (m:m) tells us that it is m:1 instead. An example shows this mapping in Figure 18 is between element editor and monograph (see Figure 23)
4. Cadinality + is mapped to m: m (total)

2.4.5.2.2. Inclusion Constraints

The IDREF and IDREFS in DTDs actually represent the foreign key relationship, which is a kind of inclusion constraint, in relational database systems. In Sem-ODB, we

do not have this concept, but we can map this constraint to a relation that points to the category which has the ID attribute, as we explained earlier in section 2.4.5.1.3. This mapping is more accurate than key and foreign key mapping as seen in the relational model, because a key could be a composite key. However, in XML ID by itself uniquely defines an element. For example, the id of element author is of ID type, it is mapped to a uid of author, if we do not consider document order. The authorID in element contactauthor is of IDREF type. It is mapped to a relation which points to author.

2.4.5.2.3. NULL and not NULL

As discussed previously, Attributes in DTDs may have default options. We map the #REQUIRED and #IMPLIED concept to total or not total and #FIXED and #DEFAULT to default value in Sem-ODM. The mapping rules about the default options are generalized as follows:

1. Attribute *A* with #REQUIRED default option: the corresponding relation is total (i.e., not null)
2. Attribute *A* with #IMPLIED default option: the corresponding relation is normal (i.e., nullable)
3. Attribute *A* with #FIXED default option: set the fixed value as default for this relation in Sem-ODB
4. Attribute *A* with #DEFAULT default option: set default value for this relation in Sem-ODB

2.4.5.2.4. Document Order and Element Order

One important feature of XML is that it may be viewed as an ordered document, i.e., a *document order* exists among the elements of an XML document. In addition, there's an *element order* concept in XML, which is captured in the DTD/XML-Schema. XML documents can root at any element as long as the relationship between elements and sub-elements is kept valid. To our knowledge, most of the papers on XML mapping schemes either did not consider the ordering issue or simply mentioned that it is easy to enhance their scheme with ordering concept. However, in most DTD→relational schema mapping approaches, in order to keep such information, at least two more columns need to be created in the relational tables to represent the parent-child relationship (element order) and ordering among tuples (document order) in each table representing an element, as in [54, 65]. In Sem-ODB, these concepts are much easier to capture. The element order is first kept in the ElementOrder Category in the meta-schema in Figure 20, and is later mapped as an attribute of each relation in Sem-ODM. As for the document order, since in Sem-ODB we can provide a uid to each abstract or binary object, it can be represented by the ordering of such uids, as long as we relax the mapping of attributes of ID type as mentioned in Section 2.4.5.1.3.

Based on the above mapping rules, we can transform the DTD in Figure 18 into the Semantic Schema shown in Figure 23. Note that in Figure 23, the relations are enhanced with ordering which is denoted by an ordering number along with the relation name. And the default cardinality in Sem-ODM is m:1.

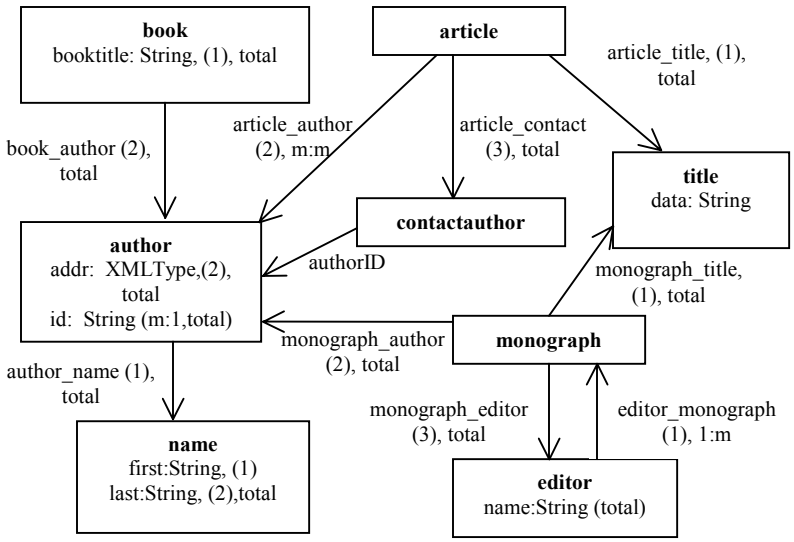


Figure 23. Semantic Schema Representation of the DTD Example

Through the transformation, the structure and semantic information of a DTD are more vividly represented in the Semantic Schema. Nested structures are split and distributed into categories and relations. Relationships between elements and sub-elements are expressed explicitly. ID and IDREF concepts are seamlessly integrated into Sem-ODM. Semantic information such as cardinality and default value are mapped to corresponding concepts in Sem-ODM. We were able to easily capture complex semantic information with the semantic schema.

2.4.6. Related Work

Many XML to relational model mapping schemes have been proposed in recent research projects [53, 54, 55, 64, 65] or applied in commercial products [66, 67]. [55] viewed XML as an ordered and labeled graph and derived relational tables according to the edges between nodes in the XML graph. This approach does not need a pre-existing DTD to help with the generation of relational tables. However, because we believe it is beneficial to have a DTD to describe XML documents, and many applications which use

XML document as exchange format require the presence of a DTD, our approach assumed the existence of DTDs for XML documents as did the approaches used in [54, 64]. However, [54, 64] generated relational tables according to the description of the DTD and the normalization requirement of the relational model. Our approach first captured the meta-schema of a DTD and a semantic schema, which is similar to the approaches used in [53, 65], and then performed the actual mapping and stored the mapping information in a Sem-ODB. [65] proposed a generic relational schema to store the structure information about any XML document so that relational data sources can be described as views over the generic schema. X-RAY in [53] introduced meta schemas for both DTD and relational schemas, defined the mapping between them, and stored the mapping information for resolving schema heterogeneity. Most of this work, except [64], only explained the mapping from a structural point of view, not the semantics of the mapping.

Similar to that taken in [64], our approach described the semantic mapping as well as structural mapping in detail. However, since we are using a different and more powerful data model, Sem-ODM, the semantics can be more vividly and naturally represented in our approach. Our mapping is easier compared to the one in [64], since some semantic concepts of DTD can be found in Sem-ODM. We also provided a complete solution for handling document order and element order. In addition, by capturing the meta-schemas of both DTD and Semantic Schema and storing them as well as mapping information in Sem-ODB, the autonomy of both data sources is preserved.

2.4.7. Results

In this section, we explained in detail a unique and complete mapping scheme to map a DTD to a Semantic Schema in Sem-ODM. The mapping scheme is complete in the sense that both structure and semantic information is mapped. The mapping is natural because Sem-ODM has very similar concepts to XML.

As an extension to the current work, we plan to implement the above mapping scheme to further evaluate its efficiency in generating the meta-schema, storing mapping information, and loading XML document into Sem-ODB. In addition, we are interested in translating XML queries into SQL and evaluating the mapping through query translation and optimization.

2.5. XML Schema Modeling Using Sem-ODM

2.5.1. Summary

XML Schema, proposed by the World Wide Web Consortium as an XML schema language, has numerous enhancements compared to DTD, a previous schema protocol for XML. However, XML Schema lacks the capability of modeling complex relationships among entities in the real world. It is desirable for users to model an XML Schema via some conceptual modeling method, preferably one that has a graphical interface. In this section, we propose to model an XML Schema using the Semantic Binary Object-Oriented Data Model's (Sem-ODM) Semantic Schema. We present a mapping scheme to transform a Sem-ODM Semantic Schema to an XML Schema. Sem-ODM, being a semantically rich data model, has expressive means to graphically model XML schemas

and is well set to be applied in the design of multimedia, distributed, and intelligent information management systems.

2.5.2. Background

With XML (eXtensible Markup Language) [68] defined by W3C [52] gaining popularity, more data is expected to be encoded in XML or exported from traditional databases as XML for easier exchange among different applications that use heterogeneous data formats. An XML Schema may be associated with XML documents to describe structure and content of the XML documents.

Compared to DTD [68], which is also a schema language for XML documents, XML Schema [62] has significant enhancements. For instance, it supports more data types, inheritance, type definition by extension and restriction, etc. However, features like these make it more difficult to write and understand a full-fledged XML Schema. In addition, though XML is a good candidate for creating structurally complex documents, its schema language, XML Schema, by itself lacks the capability of modeling the real world and the complex relationship among entities like the Entity-Relationship (ER) model does [69]. It is more desirable for users to model an XML Schema via some conceptual modeling method, and preferably through a graphical interface. The reason lies in the fact that conceptual modeling methods generally have the advantages of being easy to use in communication with customers and a high level of abstraction, while a graphical interface is easier for human beings to comprehend because its visualization is closer to the way people envision the real world.

In this section, we propose to model an XML Schema using Sem-ODM's (Semantic Binary Object-oriented Data Model) [8] Semantic Schema and provide a mapping scheme for this transformation. Sem-ODM has rich semantic modeling features and may be used to model various element content types, attribute types and constraints in an XML Schema. Sem-ODM also has a graphical representation and a schema editor tool, the Semantic Schema Editor [70], which helps to visualize the modeling process. In this section, we also offer solutions to modeling those special components of XML Schema that do not have direct counterparts in Sem-ODM.

2.5.2.1. Overview of XML Schemas

DTD, XML Schema is a schema language defined by W3C and is used to describe the content and structure constraints of XML instances. However, DTD was originated to be used for semi-structured documents and thus has very limited data types and constraints. Unlike DTD, XML Schema is designed in such a way that not only semi-structured data, but also structured data such as data in databases can be defined and restrained.

The basic constructs that an XML Schema defines are Elements and Attributes. In addition, the XML Schema introduces the concepts of simpleType, ComplexType and type definition by extension and restriction as well as some other features. There are more primitive types in the XML Schema than in the DTD. Primitive types are simpleTypes. A simpleType can be used to describe element and attribute types, while in DTD, the basic data type for elements is only PCDATA. Hence, elements and attributes can be described more precisely via appropriate data types in an XML Schema.

Furthermore, simpleTypes and ComplexTypes can be refined by either extension or restriction in an Object-Oriented style. By introducing simple/complexType, data types may be reused in a schema. Figure 24 shows an XML Schema snippet and its corresponding DTD description.

```
<xsd:element name = "course_enrollment" type ="course_enrollmentType"/>
<xsd:complexType  name = "course_enrollmentType" >
<xsd:sequence>
    <xsd:element ref = "student"  minOccurs= "0" maxOccurs= "unbounded"/>
    <xsd:element ref = "course_offering"  minOccurs= "0"maxOccurs="unbounded"/>
</ xsd:sequence>
<xsd:attribute          name = "final_grade"   type = "xsd:integer" />
</xsd:complexType>
```

(a)

```
<!ELEMENT course_enrollment (student*, course_offering*)>

<!ATTLIST  course_enrollment  final_grade  CDATA>
```

(b)

Figure 24. (a) XML schema snippet (b) DTD

2.5.2.2. Overview of Sem-ODM and Enhanced Sem-ODM

Sem-ODM (Semantic Binary Object-Oriented Data Model) is a high-level data model that can mirror the real world enterprise scenarios naturally as the ER (Entity Relationship) model does. In addition, it has advantages of the Object-Oriented data model, such as inheritance, object identifiers, and explicit relationships among objects.

The basic constructs in Sem-ODM are *Categories* and *Relations*, which are like *Entities* and *Relationships* in the ER model, respectively. There are two kinds of categories in the Sem-ODM, *Concrete Categories* and *Abstract Categories*. Concrete Categories are atomic data types such as *String*, *Number*, and *Boolean*, among others. Sem-ODM also provides a way in which users can specify their own data types, composed of atomic types defined in the Sem-ODM. For example, user can specify *Enumerate* types. In addition, regular expressions can be used to constrain the format of a data of string type. Abstract Categories are categories composed of abstract objects, for

example, categories such as person and book. Graphically, they are represented by rectangles. A binary relation, represented by a solid arrow, can be created between two categories in a Semantic Schema. The category, where the arrow starts, is called the *Domain* of the relation and must be an abstract category. The category, where the arrow ends, is called the *Range* of the relation and could be a concrete category or an abstract category. A relation from an abstract category to a concrete category is called *attribute* in a Semantic Schema, and it is represented inside its domain category rectangle with a colon (:) delimiting the attribute's name and type. *Cardinality* and other constraints (such as *totality*, default value and the list of values in case of Enumerate type) of a relation are placed alongside its name in parentheses. *Inheritance* is represented by dashed arrows from sub-categories to super-categories.

Figure 25 shows a Semantic Schema example for a university. *Person* is a super-category, which has two sub-categories: *Student* and *Instructor*. *Person* has a *non total, m:1* attribute called *name* with a range of Concrete Category *String*. The category *Student* has two relations, *major* and *minor,* pointing to the category *Department*. Note that in a Semantic Schema, relations that do not specify cardinalities and totality have *m:1* cardinality and non-total totality by default. Please refer to [8] for a thorough discussion on Sem-ODM.

The rich semantics of Sem-ODM is manifested by its ability to support generalization, explicit relationships with different cardinalities/totalities, *1:m* attributes, various data types and constraints on the data format of the data types. This allows users to model real world scenarios using Sem-ODM.

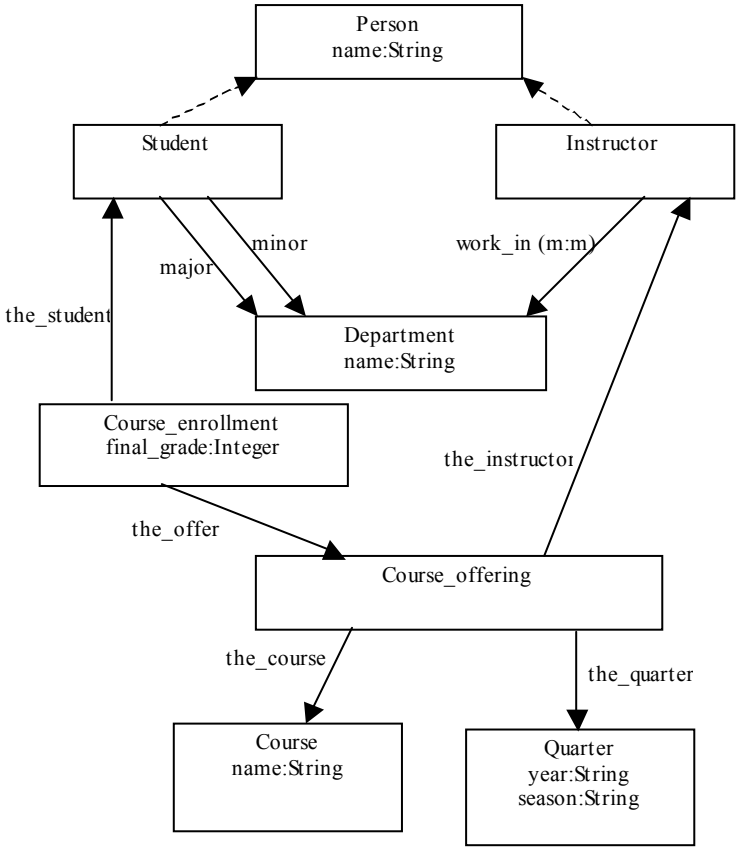


Figure 25. University Semantic Schema

2.5.3. Mapping a Schema from a Sem-ODM Semantic Schema to an XML Schema

The basic idea of the mapping scheme is to map the *inheritance* concept of Sem-ODM to the type inheritance concept of XML Schema, to map *categories* into *elements*, to map *m:1* or *1:1 attributes* into *attributes*, and to map *relations* into sub-*element relationships*. The mapping steps can be described as follows (A complete XML Schema description generated from the Semantic Schema in Figure 25 by following the steps described below can be found in Figure 33).

1. Inheritance

a. For each super-category *C*, create an element "*C*" of complex type "*CType*", i.e.,

<xsd:element name = “C” type=”CType”>

<xsd:complexType name = “CType”>

…….

</xsd:complexType>

The content of *CType* will be completed after the following step (b) and (c).

For example, the category *Person* in Figure 25 can be mapped as the following XML schema snippet:

<xsd:element name = “Person” type=”PersonType”>

<xsd:complexType name = “PersonType”>

…….

</xsd:complexType>

b. For each relation *r* of *C*, where *r: C→R (c, t), R* is the range of *r, c* and *t* are the cardinality and totality of *r*, respectively.

- If there exists at least one more relation *r’* of *C*, where *r’*: *C→R* , i.e., both *r* and *r’* are relations from *C* to *R*, then create two child elements “*r*” and “*r’*” of type “*RType*”, i.e.,

  <xsd:sequence>

    <xsd:element name = “r” type = “RType” >

    <xsd:element name = “r’” type = “RType” >

        ……..

  </xsd:sequence>

  Here we map two relations of *C* into its sub-elements and the definition of *RType* will be provided when the category R is visited during the process.

- Otherwise (r is the only relation of C with range R), create a child element with ref ="R" inside the complex type "CType", i.e.,

```
<xsd:sequence>
    <xsd:element ref = "R" >
        ……..
</xsd:sequence>
```

Here, we use *ref = "R"* to indicate the element is referencing an element called "*R*". Later on, when the category R is visited, a definition for element "*R*" will be provided.

According to the value of *c* and *t,* we can append the "minOccurs" and "maxOccurs" facets to the previously shown <xsd:element…> definition inside *CType* definition.

i) *c = m:m or 1:m, t = total*

minOccurs= "1" maxOccurs= "unbounded"

ii) *c = m:m or 1:m*

minOccurs= "0" maxOccurs= "unbounded"

iii) *c = m:1 or 1:1 , t = total*

minOccurs= "1" maxOccurs= "1"

iv) *c = m:1 or 1:1*

minOccurs= "0" maxOccurs= "1"

For instance, suppose the category *Person* has two relations called *edit* and *publish* to another category *Book* as illustrated in Figure 26.

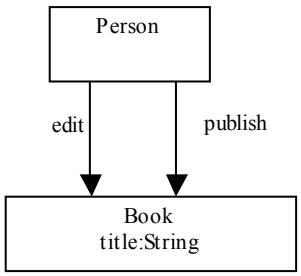


Figure 26. Super-category Person with two relations

It will be mapped as the following XML Schema snippet:

```
<xsd:element name = "Person" type="PersonType"/>
<xsd:complexType  name = "PersonType">
  <xsd:sequence>
     <xsd:element name = "edit" type = "BookType"minOccurs= "0"/>
     <xsd:element name = "publish" type = "BookType"minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>
```

c. For each attribute *a* of *C,* where *a*: C→*T(c,t)*, *T* is the range, which is a concrete category, of *a, c* and *t* are the cardinality and totality of *a*, respectively.

- If *c* = *m:m* or *1:m*

  Create an element named *a* inside *CType* as the following:

  - if *t* = *total*

    <xsd:element name = "a" type=

    "xsd:Ttype" maxOccurs= "unbounded"/>

  - otherwise

    <xsd:element name = "a" type=

    "xsd:Ttype" minOccurs= "0" maxOccurs= "unbounded"/>

- Otherwise (i.e., *c* = *m:1* or *1:1*)

  Create an attribute named *a* inside *CType* as the following:

  - if *t* = *total*

<xsd:attribute name = “a” type=

“xsd:Ttype” or “uType” use= “required”/>

- otherwise

<xsd:attribute name = “a” type= “xsd:Ttype” or “uType”/>

In most of the cases, the *Ttype* in the above declaration is a primitive type allowed in the XML Schema. It is the counterpart of the Concrete Category *T*. For example, if *T* = *integer*, then *Ttype* = *integer*. There is an exception to this rule. When *T* = *Enumerate*, we need to provide a user-defined simple type indicated by “uType” above to represent *Enumerate* in the XML Schema. For instance, suppose we have a *total* attribute called *gender* inside a *person* category. It has enumerated value of “*female*” and “*male*”. We need to provide a simple type called “genderType” to represent the enumerated values. The representation in an XML Schema is as follows:

```
<xsd:attribute  name = “gender”type = “genderType” use = “required”>
    <xsd:simpleType  name = “genderType”>
        <xsd:restriction            base = “xsd:string”>
            <xsd:enumeration value = “female”/>
            <xsd:enumeration value = “male”/>
        </xsd:restriction>
    </ xsd:simpleType>
</xsd:attribute>
```

Since Sem-ODM provides a means to specify the Enumerate type, which is a common data type in most programming languages, and is also supported by the XML Schema, it is natural to model such a data type in the Semantic Schema. In addition, it is worthwhile to mention that Sem-ODM is equipped with some other useful features. For example, for the integer type, users can specify the

minimum and maximum number of the data of this type. This feature can be used to model the "minInclusive" and "maxInclusive" facets. Suppose we are only interested in person whose birthyear is after 1940(inclusive) and before 2100 (inclusive). We can specify it in a Semantic Schema shown in Figure 27.

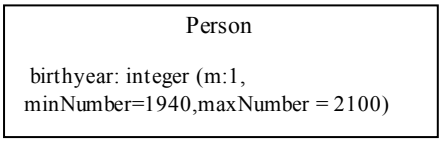


Figure 27. Integer Type with MaxNumber and MinNumber

This can be transformed into the following XML Schema snippet.

```
<xsd:attribute          name = "birthyear">
  <xsd:simpleType>
      <xsd:restrictionbase = "xsd:integer">
          <xsd:minInclusive value = 1940 />
          <xsd:maxInclusive value = 2100/>
      </xsd:restriction>
  </ xsd:simpleType>
</xsd:attribute>
```

In the XML Schema, regular expressions can be used to restrict the format of data used in an instance document. For example, "\d{3}" indicates a value that consists of three digits. This can be easily modeled by Sem-ODM, because Sem-ODM offers a similar feature in its String Category. For example, suppose there is a *SSN* attribute of *String* type in category *Person*, whose format should conform to a regular expression "\d{3}-d{2}-d{4}".

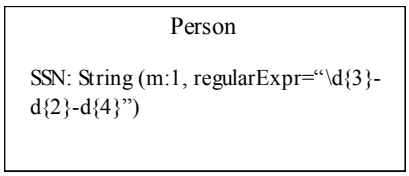


Figure 28. String Type Using Regular Expression Restraining Form

This may be mapped to the following XML Schema snippet.

```
<xsd:attribute          name = "SSN">
    <xsd:simpleType>
        <xsd:restriction      base = "xsd:string">
            <xsd:pattern value = "\d{3}-d{2}-d{4}"/>
        </xsd:restriction>
    </ xsd:simpleType>
</xsd:attribute>
```

All the attribute definitions of *CType* should come after the element definitions of *CType*.

d. For each sub-category *Ci* of *C*, create an element "*Ci*" of complex type "*CiType*" by extending the base type "*CType*", i.e.,

<xsd:element name = "Ci" type="CiType">

<xsd:complexType name = "CiType" base =

"CType" derivedby = "extension">

Then we can follow the above (b), (c) to complete the definition of *CiType* by analyzing the relations and attributes of *Ci*.

2. For any unvisited category *C*, create an element "*C*" of complex type "*CType*", as described in 1-(a) and then follow the rules in 1-(b) and 1-(c) to complete the definition of *CType*.
3. Root Element

   Finally we define a root element "*root*" of complex type "*rootType*", which consists of references to all the elements that are mapped from categories in the original Semantic Schema, i.e.,

```
<xsd:element name = "root" type = "rootType" >
   <xsd:complexType name = "rootType">
      <xsd:sequence>
         <xsd:element ref = "C" minOccurs= …maxOccurs= "unbounded"/>
         <xsd:element ref = "C1" minOccurs=…maxOccurs="unbounded"/>
            ……
      </ xsd:sequence>
   </xsd:complexType>
</xsd:element>
```

4. Other modeling strategies

   So far, we have only discussed the modeling capabilities of Sem-ODM from the perspective of the Semantic Schema. We can map the concepts in a Semantic Schema to the counterpart of XML Schema through the above described steps. Can we model concepts of XML Schema in a corresponding Semantic Schema? This is the question we want to answer in this section.

   a. Ordering

      It is quite often the case that we need to specify an ordering among sub-elements of a parent element in an XML Schema. This is done via the <xsd:sequence> tag. In the previous section, we imply an ordering among relations of a category while we are mapping the relations into sub-elements of the category. This order is decided by the mapping algorithm. However, what if we want to specify a particular ordering during our modeling process? In this case, we need to provide a specific way to help users to do so. We propose to denote this ordering in the name of relation *r* by appending the ordering number *i* to *r*, i.e., we use *r_i* to represent the *ith* sub-elements of a parent element. For example, to model sub-element *major* should occur before *minor*, we can use the following representation shown in Figure 29:

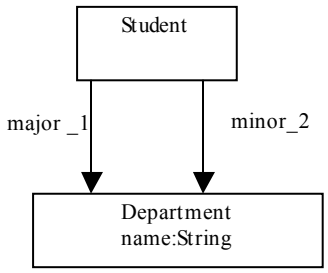


Figure 29. Ordered Relations

This will be mapped to the following XML snippet, similar to the one illustrated in (1)-b).

```
<xsd:element name = "Student" type = "StudentType"/>
   <xsd:complexType  name = "StudentType"        >
      <xsd:sequence>
         <xsd:element name = "major" type = "DepartmentType"
            minOccurs= "0"       />
         <xsd:element name = "minor" type = "DepartmentType"
            minOccurs= "0"       />
     </ xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

b. Choice

Sometimes we want to express the idea of choosing one item from a group items in an XML Schema. This can be conveyed via the <xsd:choice> tag. For example, the following XML schema snippet indicates that the *body* of a *book* can contain one or many *chapter* or *part* sub-elements in any order.

```
<xsd:element name = "body" >
      <xsd:complexType >
         <xsd:choice     maxOccurs= "unbounded">
            <xsd:element ref = "part" />
           <xsd:element ref = "chapter" />
         </ xsd:choice>
     </xsd:complexType>
</xsd:element>
```

We model this concept using the same idea as modeling ordered sub-elements. We use *contain_ori* to represent that the relation *r* should be mapped as a part of a choice content. Here *i* actually does not imply any ordering and is just used to

differentiate among different relations of the same category. For example, the above XML Schema can be modeled in the following Semantic Schema:

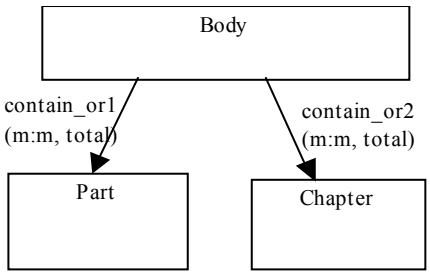


Figure 30. Relations Representing Choice

c. Any order

It is well known that XML schemas, unlike DTD which has to use a very clumsy code to do the same job, can specify any order concept by introducing the <xsd:all> tag. For example, to specify that a book element can have a title, author, and ISBN sub-elements in any order, we can use the following snippet:

```
<xsd:element name = "book" >
   <xsd:complexType >
      <xsd:all>
         <xsd:element ref = "title"  type = "xsd:string"/>
         <xsd:element ref = "author" type="xsd:string"/>
         <xsd:element ref = "ISBN" type = "xsd:string"/>
      </ xsd:all>
   </xsd:complexType>
</xsd:element>
```

We use *contain_alli* to represent that the relation *r* should be mapped as a part of an *all* content. For example, the above XML Schema can be modeled in the following Semantic Schema:

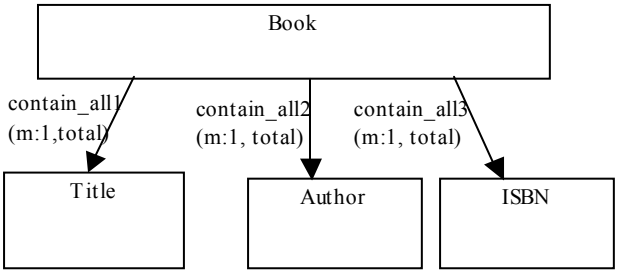


Figure 31. Relations Representing Any Order

d. Empty Content

Elements of *Empty Content* can be modeled by a category with no outgoing relations. For example, to model the following XML Schema snippet, we use the Semantic Schema in Figure 32.

```
<xsd:element name = "image" >
   <xsd:complexType >
      <xsd:attribute name="href"type = "xsd:string" />
   </xsd:complexType>
</xsd:element>
```

Figure 32. Category Modeling Element of Empty Content

3. Distributed Databases

This chapter explores methods to enhance distributed databases. It explores the optimization of queries by exploiting features of intelligent networks, such as ISDN or Internet provisioning of varying levels of Quality of Service.

This work was published as:

N. Rishe, S. Graham. "Database Query Distribution over Intelligent Networks." Air Force Research Laboratory Information Directorate Final Technical Report, AFRL-IF-RS-TR-2001-65, 2001. 132 pp.

```
<xsd:element name = "university" >
    <xsd:complexType >
        <xsd:sequence>
          <xsd:element ref = "person" minOccurs= "0"      maxOccurs= "unbounded"/>
          <xsd:element ref = "student" minOccurs= "0"     maxOccurs= "unbounded"/>
          <xsd:element ref = "instructor" minOccurs= "0"  maxOccurs= "unbounded"/>
          <xsd:element ref = "department" minOccurs= "0"maxOccurs= "unbounded"/>
          <xsd:element ref = "course_enrollment" minOccurs= "0"    maxOccurs= "unbounded"/>
          <xsd:element ref = "course_offering" minOccurs= "0"      maxOccurs= "unbounded"/>
          <xsd:element ref = "course" minOccurs= "0"      maxOccurs= "unbounded"/>
          <xsd:element ref = "quarter" minOccurs= "0"     maxOccurs= "unbounded"/>
        </ xsd:sequence>
    </xsd:complexType>
</xsd:element>

<xsd:element name = "person" type = "personType"/>
<xsd:complexType  name = "personType">
        <xsd:attribute      name = "name"      type = "xsd:string" />
</xsd:complexType>

<xsd:element name = "student" type = "studentType"/>
<xsd:complexType  name = "studentType"base = "personType"derivedby="extension">
        <xsd:sequence>
       <xsd:element name = "major"          type = "departmentType"      minOccurs= "0"/>
       <xsd:element name = "minor"          type = "departmentType"      minOccurs= "0" />
        </ xsd:sequence>
</xsd:complexType>

<xsd:element name = "instructor" type = "instructorType"/>
<xsd:complexType  name = "instructorType"      base = "personType"derivedby="extension">
        <xsd:sequence>
       <xsd:element ref = "department" minOccurs= "0"     maxOccurs= "unbounded"/>
        </ xsd:sequence>
</xsd:complexType>

<xsd:element name = "department" type = "departmentType"/>
<xsd:complexType  name = "departmentType"     >
        <xsd:attribute      name = "name"      type = "xsd:string" />
</xsd:complexType>

<xsd:element name = "course_enrollment" type = "course_enrollmentType"/>
<xsd:complexType  name = "course_enrollmentType" >
        <xsd:sequence>
       <xsd:element ref = "student" minOccurs="0"maxOccurs= "unbounded"/>
       <xsd:element ref = "course_offering" minOccurs= "0"maxOccurs= "unbounded"/>
        </ xsd:sequence>
        <xsd:attribute      name = "final_grade"type = "xsd:integer" />
</xsd:complexType>

<xsd:element name = "course_offering" type = "course_offeringType"/>
<xsd:complexType  name = "course_offeringType" >
        <xsd:sequence>
       <xsd:element ref = "instructor" minOccurs= "0"      maxOccurs= "unbounded"/>
       <xsd:element ref = "course" minOccurs="0"maxOccurs= "unbounded"/>
       <xsd:element ref = "quarter" minOccurs="0"maxOccurs= "unbounded"/>
        </ xsd:sequence>
</xsd:complexType>

<xsd:element name = "course" type = "courseType"/>
<xsd:complexType  name = "courseType">
        <xsd:attribute      name = "name"      type = "xsd:string" />
</xsd:complexType>

<xsd:element name = "quarter" type = "quarterType"/>
<xsd:complexType  name = "quarterType">
        <xsd:attribute      name = "year"      type = "xsd:string" />
        <xsd:attribute      name = "season"    type = "xsd:string" />
</xsd:complexType>
```

Figure 33. University XML Schema

Portions of this report were refined to include additional networking concepts and published as:

> N. Rishe, O. Wolfson, S.Graham, W. Sun. "Database Query Distribution over Intelligent Networks." Workshop on Database Technologies at the Beginning of the New Millennium (Softcom 2001). v. 1, pp. 109-116.
>
> O. Wolfson, N. Rishe, S. Graham, W. Sun. "Database Integration over Hybrid Networks." 2001 International Conference on Software, Telecommunications, and Computer Networks (Softcom 2001). v. 2, pp. 935-942.

3.1. Database Query Distribution over Intelligent Networks

3.1.1. Summary

The main objective of a Query Distribution Algorithm is to find efficient ways to evaluate global queries. In traditional distributed database systems based on proprietary WAN infrastructures, efficiency of query processing is typically measured by query response time: the time it takes to finish a query. The monetary cost of network usage is not explicitly considered in the cost model since the overall dollar cost of maintaining the proprietary network is a fixed figure. This has placed the focus on how to reduce necessary data transfer across the network as much as possible for each query (and thus reduce transfer time), in order to maximize the effective use of the available network bandwidth. The adoption of intelligent networks in a distributed database system reveals two fundamental changes from a traditional one based on proprietary networks. First, the connections among the distributed sites are dynamic rather than static. In other words, the network links are established (through call setup) only when needed and the bandwidth can be allocated/de-allocated on-demand. Second, the dollar cost of the connections is charged on a per-connection basis. These new factors make the direct

adoption of traditional distributed query processing and optimization techniques unsuitable as they do not consider network dollar cost (which can vary between any two sites) and query response time in an integral manner. This section presents a Query Distribution Algorithm that takes these factors into consideration.

3.1.2. Background

The problem of query processing and optimization in a distributed database environment has attracted much attention from the database research community, and numerous publications have been devoted to this subject. The most expensive database operation is the join operation. Considerable research effort has been applied to minimize the cost of join queries. A common technique to reduce the communication cost in a distributed join is to use semi-join. The semijoin strategy was justified by the assumption that network transmission cost is the dominant component in overall query processing cost.

There have been many theoretical studies in database integration. [71, 72, 73], and a number of prototype systems such as Information Manifold [74], Multibase [75], MRDSM [76], Pegasus [77], Carnot [78], SIMS [79], TSIMMIS [80, 81], and SEMHDB [20, 21] each represent a different methodology. One common assumption in the previous research work is that the ``communication cost rates are constant and equal to transmission rate" [82] and thus the monetary cost of network usage is usually not explicitly considered. This greatly simplifies the problem of distributed query optimization as there is only one optimization metric: time. This assumption and therefore the various techniques developed based on it do not apply to our proposed

distributed database environment using intelligent networks. First, it is clear that constant inter-node bandwidth cannot be assumed in networks made up of, for example, ISDN links, where a varying number of ISDN links may be established between two nodes. Second, with ISDN being charged based on usage, we now have two objectives for minimization: query response time and ISDN cost (in real dollars). Our proposed Query Distribution Algorithm (QDA) aims to close this gap and advance the query processing and optimization technologies to the new horizon of intelligent networks.

3.1.3. System Overview

Our research is based on a heterogeneous distributed database system interconnected via an ISDN network. The techniques are applicable to other intelligent network topologies. We adopt the concentrated management mode. AMIP, the Application Management Interface Program, is the concentrated manager server of whole system. LIP, the Local Interface Program, is the client program offering a user interface. The heterogeneous DBMS at each local site communicates with AMIP and LIP through ODBC. Figure 34 depicts the system's physical architecture. Further details on both AMIP and LIP can be found in [83].

The user queries the global schema using standard SQL statements. To reduce communication cost, the SQL parser module resides in AMIP. LIP only sends the user's input query to AMIP. It is up to AMIP to parse the queries based on SQL syntax, generating a query tree. If the syntax generates a parser error, AMIP will notify LIP to ask the user to check the query syntax and reenter the query. AMIP then produces an execution plan for the query, based on the global schema. This procedure is the focus of

this section. AMIP then executes the EP, transferring the resulting table to a dedicated LIP site if necessary and notifies the originating LIP to display it. According to the execution plan, the coordinator module will control LIPs to execute sub-queries and synchronize the data transfer from LIP to LIP. This query processing model is depicted in Figure 35.

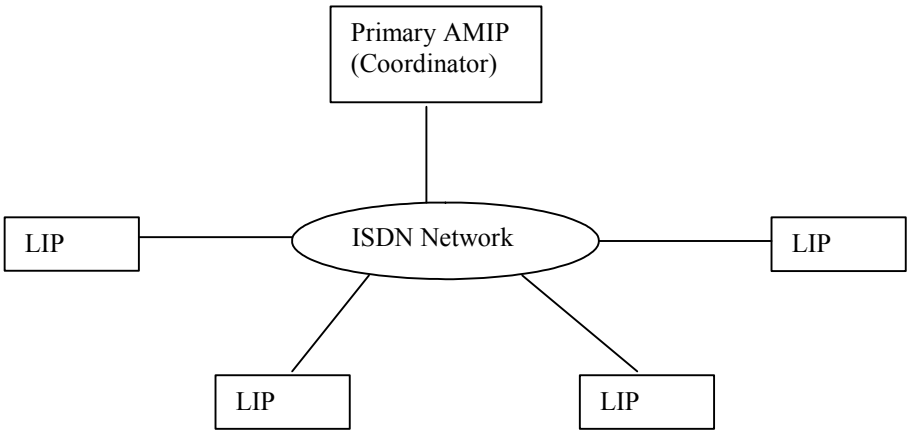


Figure 34. Physical Architecture of QDA

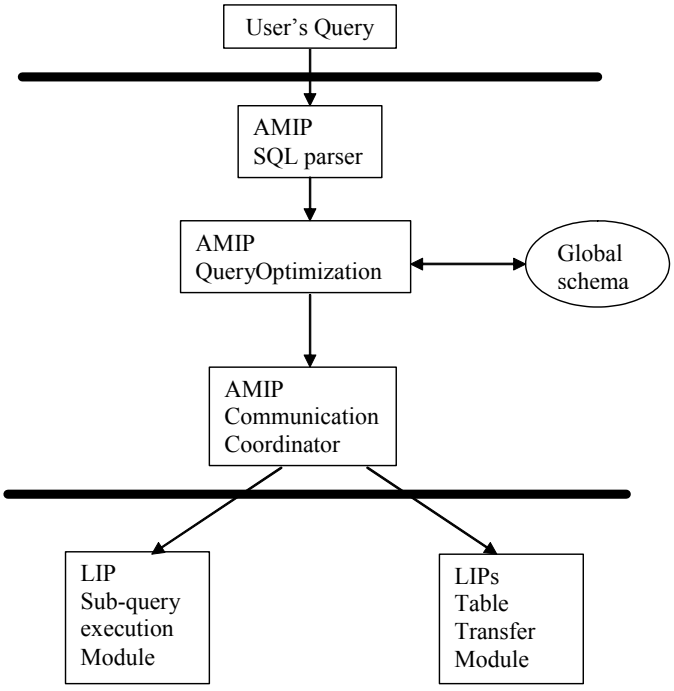


Figure 35. Query processing

3.1.4. Query Execution Plan

A query execution plan is a sequence of operations that retrieves data from databases and composes the answer set for a query. In our distributed database system, which provides access to heterogeneous data sources, an execution plan is a sequence of mixed sub-query executions and table transfer operations. A sub-query is a query that is to be executed against a certain data source at a certain LIP site; a table transfer operation sends a table from one LIP site to another. The goal of the Query Distribution Algorithm is to construct a cost-effective execution plan for a distributed query. A widely adopted rule of thumb when constructing an execution plan is to perform selections and projections as early as possible. This is usually referred to as the "push-down" strategy for selections and projections. Reflected in our system, we should perform the query's selections and projections present for the involved tables at their local sites before sending them over the network for more processing. The rationale behind this is to reduce the size of the tables in order to save communication cost.

We now look into the issues pertinent to finding a cost-effective execution plan for a query. We divide the discussion into two types of queries: (1) *selection queries* - which access only one table, and (2) *join queries* - which combine tuples from more than one table. For a selection query, the selection must be performed at the local site first (according to the push down rule). The only communication cost incurred is for sending the restricted table to the destination site where the result of the query is to be placed. To minimize the cost, we need to find the least-cost ISDN call path to transfer the data. We maintain an up-to-date graph of the network topology and potential network topology with costs for each link. The problem of finding the least-cost path can be formulated as

the classic "Shortest Path Problem" for which an efficient algorithm is well known. In addition to minimizing communication cost for selection queries, the shortest path algorithm is also useful in constructing execution plans of low communication costs for join queries, which will be discussed next.

The execution plan for a distributed join query comprises a sequence of two-way (2-way) joins, each of which joins two tables from different sites. Several decisions need to be made to select appropriate strategies while constructing the execution plan.

- *join method*: Two join methods, pure join and semi-join, are widely adopted in traditional distributed database systems. A pure join between tables *A* and *B* is performed by sending one of the tables (subject to restriction by selections and projections) to the other site for a regular join. We use *A* ||X| *B* to denote a pure join where *B* is the table being sent. Notation *A* |X|| *B* can be understood similarly. A semi-join, denoted *A* |X *B*, performs the following steps sequentially: (1) send *A*'s join column(s) to *B*, (2) restrict *B* by selecting only those tuples whose join attribute values have a match in the join columns received from *A*, (3) send the restricted table of *B* back to *A*'s site and perform a regular join with *A*. Semi-join *A* |x *B* requires less data transferred than a pure join when the size of the projected join column(s) of *A* consists of only a small portion of the total data transferred and table *B* is highly restricted by *A*'s join column(s).
- *join order*: When joining two tables (either by pure join or semi-join), there are two directions to perform the join (*A* ||X| *B* or *A* |X|| *B*, *A* |X *B* or *A* X| *B*). When joining more than two tables, the order of performing the individual 2-way joins also has an impact on the final communication cost. For example, to perform a three-way join (*A*

join *B* join *C*), we can do either ((*A* join *B*) join *C*) or (*A* join (*B* join *C*)), with each two-way join subject to the options of pure join or semi-join.

- *data transfer paths*: There could be multiple paths to transfer a table from one site to another. The least expensive one must be sought in order to save communication cost. Note that the least cost path between two sites need not to be the direct connection (namely the edge connecting the two nodes). It could be an indirect path which travels through other node(s) to relay the data (namely, a path that consists of more than one edge). For example, if a semi-join *A* |x *B* is to be performed, the least cost paths to send data from *A* to *B* and from *B* to *A* must be found. And the paths can go through a third site *C*.
- *assembly sites*: Assembly sites are the places where the intermediate or final query results are assembled. Some algorithms only allow one assembly site, i.e., the destination site, and defer the assembly until the last stage. In those strategies, no intermediate join results, except for semi-join restricted tables, are allowed to be generated. Some allow more flexible choices on the sites to place intermediate join tables.

Given all the options in constructing execution plans, the number of possible execution plans for a join query increases exponentially with the number of involved tables. Thus, it is cost-prohibitive to perform an enumerated search on the entire space of execution plans to find the optimal one. In the next section, we describe an algorithm that finds an efficient execution plan from a selected subset of the search space.

3.1.5. Query Distribution Algorithm

3.1.5.1. Description

QDA is the kernel part of the AMIP. As shown in Figure 36, it consists of two parts: pre-order traverse and post-order traverse. The former receives the SQL objects (a data structure of query trees) produced by the AMIP's SQL parser, which processes SQL syntax, checks it, and produces SQL objects. Then pre-order traverse processes the query tree, makes a global semantic check of each table, does some preparation work such as assigning each source table a global exclusive internal table name and processing horizontally fragmented tables. We call this procedure "pre-order traverse query tree" or simply "pre-order traverse" because it traverses the query tree from root to leaves. The second step is post-order traverse, which accepts the result of pre-order traverse as input: a modified query tree and a global table schema object. Post-order traverse generates a distributed query Execution Plan (EP) and executes it by sending the command messages to LIP one by one to control and complete the query process. We call this procedure "post-order traverse query tree" or simply "post-order traverse" because it traverses the query tree from leaves to root.

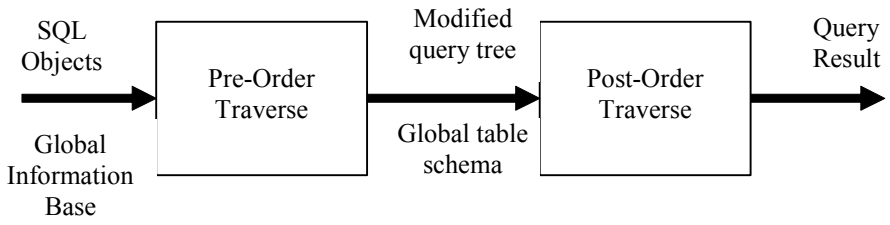


Figure 36. QDA Procedure

3.1.5.2. QDA Algorithm

The QDA is the central module of the distributed system. The main task of QDA is processing complicated join queries. QDA decomposes chain queries into sub-queries according to least weight cost of communication cost and response time. Join decomposition adopts LPT (Linear Processing Tree) or Bushy Tree. The search space complexity of the former is O(n^3) and the latter is O(n^5) (where n is number of tables in the join). The dynamic channel allocation of ISDN is also considered in QDA as an important parameter in the shortest path search. To compress the search space and decrease the complexity of QDA processing, we adopt LPT search, and many heuristic rules are also introduced. The dynamic channel allocation of ISDN is considered as an important parameter in the shortest path search used by QDA. Last, but not least, the optimizing object function is decided by a weighted total of communication cost and response time. This weighted cost offers users a simple way to optimize the query processing to the user's preference.

When AMIP receives a query statement from LIP, it first calls upon its SQL parser to analyze the query statement and generate a query tree. The QDA process at the AMIP module performs three steps:

A. Pre-Order traverse the query tree and perform four tasks:
    1. For each node in the query tree, generate the corresponding result table schema and insert it into EPGblSchema. (EPGblSchema records the schema of all tables included in a SQL query, depending on a special thread.)

2. For each SELECT node of the query tree, perform a semantic check on the tables and their related attributes according to the information stored in the knowledge base.
3. For each SELECT node of the query tree, if there is an OR operator in the WHERE list, convert OR to UNION ALL and modify the query tree accordingly.
4. For each table in the FROM list of a SELECT node, generate the corresponding table and insert its schema into EPGblSchema.

B. Post-Order traverse the query tree and perform nine tasks:
1. For each UNION, UNION ALL, and BRACKET node of the query tree, generate the result table according to the EPGblSchema.
2. For each local query of a SELECT node, process the query locally then transfer the results to the destination site.
3. For each distributed query of a SELECT node, shrink each table in the FROM list by executing another query that is relatively simpler and that is only related to the local table itself and the projections.
4. For the tables generated in step 3, produce new SELECT nodes with only GlobalName tables.
5. From the new SELECT nodes, generate a new Join Graph.
6. From the new Join Graph, call the QDA kernel algorithm and generate a new Execution Plan (EP).
7. For the new EP, call EPInterpreter to execute the EP and produce the resultant table on the destination site.

8. With the resultant table and the queries of the original SELECT node, make a new query with all operations at the local site.
9. Evaluate the new local query, generate the final result table, and insert the result table's schema into EPGblSchema.

C. After the Post-Order traverse, we can produce the final result table with the query tree and EPGblSchema. We then delete all temporary tables on each local site.

The following provides further detail on how an Execution Plan is generated from a Join Graph.

For optimization, a Shrink Table procedure is introduced to do some simple condition query and projection operations to minimize the size of tables to be joined. Meanwhile some accessory work is done for the upcoming processing of the distributed join query. With the pre-processed simplified internal table produced above, we can focus on processing the join operation (including Cartesian product) and producing a Join Graph, each node of which indicates the internal tables taking part in join operation and each edges of which indicate join relations between two tables.

The next step is to produce a linear tree that can be turned into an execution plan. We can get a great number of linear trees from a Join Graph. If we consider the Cartesian product as a join relation between two vertices of the Join Graph, the number of linear trees produced is C(n,2)*(n-2)!=n!/2. When n is not a small number, the search space is very large. To compress the search space, we use some heuristic rules, which are described below:

- Select the minimal size table as the first table in Linear Tree.
- The priority of join is greater than Cartesian product.

- When there exists more than one join relation table (Adjacent Vertex list has more than one element) or Cartesian product table, select the table that has minimal size.

From a Linear Tree we get to know the order of tables in a join sequence and the operations between two tables: join(|X|) or Cartesian product(X). Now we have to decide the join method (semi-join or pure join) and the result table location in each join operation step. Since the search space is still very large (2^(n-1) if there are n source tables and we do not consider the join method), we must introduce some heuristic rules to optimize the whole search process. The first rule is called destination pruning, and the other is called of k-stage decision.

The destination pruning technique is a pre-processing step toward the production of an execution plan. The pruning technique is based on the fact that in each join step we can decide the result table location immediately if the location of one of the operand table is equivalent to the destination site. Another aspect is that we also do not have to search the intermediate result table location, since we always put the intermediate table on the destination site.

K-stage decision is introduced to offer users the option to limit the depth of the binary search tree according to practical conditions, such as computer's processing ability or a user's need for optimization accuracy, to compress the searching space from 2^(n-1) to 2^k. When variable parameter k (k<n) is set, the searching space is compressed to 2^k. At the end of the k^{th}-stage, we can get a partial optimal conclusion to decide the intermediate table location and join method of each join step in the stage. When k>=n-1 we scan the whole searching space and can acquire the optimal EP.

The introduction of the k-stage decision method compresses the search space greatly when n is not a small number. But under a multi-query concurrent environment the search space can still be excessive and very difficult to compute even if n is not too large. In practice we adopt the "k=1 local optimal" rule to achieve the trade-off. When processing one distributed query we have to predict the variation of the ISDN Network Topology Graph (NTG) to get the k-stage optimal cost. When there exist multiple concurrent distributed query requests, a simple solution is to set the current real NTG as the reference NTG and to not generate any predicted NTG to estimate local optimal cost. To acquire a k-stage optimal cost with concurrent query requests, we must adopt a dynamic prediction method to predict the real variations of the NTG in a multi-threaded environment.

3.1.6. Results

We have presented the outline of a Query Distribution Algorithm that is optimized for databases distributed across intelligent networks. Traditional query distribution algorithms do not take the costs associated with setting up and maintaining network connections into account. Since these costs were typically fixed on a monthly or yearly fee schedule, instead of a per-minute or per-second schedule, they did not need to. With today's intelligent networks, these issues do become factors in any optimization method for query distribution. Our Query Distribution Algorithm has been implemented in a prototype system and has proven to be effective, allowing optimizations to be made for both response time and communication cost. Future work along these lines includes optimizations for hybrid networks, and for varying Quality of Service levels.

3.2. Database Integration over Hybrid Networks

3.2.1. Summary

The use of intelligent network technology and hybrid networks that include intelligent networks has created new challenges for those who wish to integrate heterogeneous databases distributed across these networks. Advances in public network technology enable the implementation of new distributed database systems that require improved query distribution and optimization algorithms. Reducing response time has typically been the key focus of distributed query execution over traditional networks. The new intelligent and hybrid networks allow optimizations to be made to minimize the communication costs needed to execute queries. In this section we extend our database integration methodology to hybrid networks by simulating ISDN networks over TCP/IP.

3.2.2. Background

Network technology is continuously developing to meet the requirement of information sharing among customers and business organizations. New versions of these networks are expected to become available to businesses and consumers since users expect to be able to retrieve information across networks at a faster speed and at less cost. Intelligent public networks, such as ISDN for example, have become widely available in North America during the past few years. These networks can be used to provide distributed computing to those who have found it too expensive in the past by providing a cost savings over traditional distribution techniques such as leased lines. ISDN lines are also available to a wider range of customers than newer technologies such as DSL lines. Intelligent networks are significant to the database research community because they

render many assumptions and parameters of conventional distributed database systems obsolete. As such, database researchers need to pay more attention to the new challenges brought about by these newly emerging technologies.

There have been many theoretical studies in database integration. [71, 72, 73], and a number of prototype systems such as Information Manifold [74], Multibase [75], MRDSM [76], Pegasus [77], Carnot [78], SIMS [79], TSIMMIS [80, 81], and SEMHDB [20, 21] each represent a different methodology. This previous work was all performed with traditional distributed database environments, where databases are connected by privately owned networks, in mind. By making use of intelligent network technologies, such as ISDN, distributed databases will be able to communicate through public networks provided by the telecommunication companies. This transition has changed the nature of the networks from static to dynamic. Using these new technologies, network resources can be dynamically allocated and manipulated directly by the database systems to allow them to achieve better network utilization and overall performance. These changes require database integration techniques, including distributed query optimization techniques, to be revisited since the assumptions that were typically made about the networks in previous studies are no longer valid.

In [83], we presented an overview of our approach to integrating databases over intelligent networks, which was first proposed in [84]. This section addresses our efforts to extend our approach to include hybrid networks.

3.2.3. Query Distribution Method

Our Query Distribution Algorithm (QDA) is a heterogeneous distributed database system based on ISDN network. It adopts the concentrated management mode. AMIP, the Application Management Interface Program, is the concentrated manager server of the whole system. LIP, the Local Interface Program, is the client program offering a user interface. The heterogeneous DBMS in each local site communicates with AMIP and LIP through ODBC. The system's physical architecture is depicted in Figure 34.

Our system maintains a collection of essential knowledge, including global schemata and mapping information, to resolve the possible heterogeneity [85], determining schematic and data conflicts between the global schema and local schemata [86]. Usually, such conflicts occur when semantically equivalent data is represented in different ways. Our prototype focuses on the four major types of conflicts that are most often cited in the literature. They are:

- *Structural heterogeneity.* Logical data structures (i.e. the number of relations and the foreign key joins between the relations) may vary across databases that contain similar data. This is due to different preferences on how data should be organized and due to variations in database contents.
- *Abstraction heterogeneity.* This class of heterogeneity arises when two databases use different levels of generalization and aggregation and retain different levels of information detail.
- *Naming heterogeneity.* Naming variations can appear in two levels: the relation level and attribute level. Differences in the names of two similar relations or

attributes will make them appear different to the database system, thus requiring semantic reconciliation.

- *Domain heterogeneity*. Even for same-named attributes, the underlying domains may be different.

Our system makes use of intelligent network features to optimize for interconnection costs as well as response time. We have identified techniques to take advantages of ISDN's tariff structure. These include the following:

- Intelligent connect/disconnect strategies: For long-distance ISDN connections, the charge is based on usage increments, regardless of whether or not one uses up all the available bandwidth during the entire increment. Thus, the connection must be kept until the current increment expires so that the remaining increment can be used for another possible data transfer without additional cost.
- Dynamic channel allocation: Weighing between query response time and ISDN monetary cost, techniques are developed to determine how many channels should be allocated to achieve a maximum weighted cost.
- Least-cost data transfer paths: To minimize the ISDN monetary cost incurred, the least-cost path must be found for each data transfer. We have modified Dijkstra's shortest path algorithm to take into account both ISDN cost and transfer time. The new algorithm not only determines the path via which the data should be routed, it also determines how many channels should be allocated for each direct edge along the path.

3.2.4. Intelligent Network Prototype

We have implemented a prototyping system based on the methodology introduced in Section 3.2.3. This section presents a brief introduction to our prototype system.

Figure 37 depicts the basic architecture of our loosely coupled multi-database environment. Each site in the figure runs an autonomous DBMS. One site among them is designated as the coordinator whose responsibility is to receive inter-database query requests from the other sites and generate the corresponding global execution plan. There are two different kinds of software components in the system: the Application Manager Interface Program (AMIP), which runs on the coordinator site, and the Local Interface Program (LIP), which runs on the remaining sites. We use ODBC to resolve the potential heterogeneity among the different kinds of DBMSs. The LIP is a specific ODBC driver. An application program submits ODBC compliant SQL queries to an LIP by means of the ODBC manager. If the queries only relate to the local database, the LIP will access the local database by using the ODBC manager to select another ODBC driver that is provided by the local DBMS. If the queries need to access one or more remote databases, the LIP will send the queries to the AMIP. The AMIP parses and decomposes the received queries into a global execution plan that is composed of a set of sub-queries. The AMIP coordinates the processing of each sub-query by brokering connections among the involved local sites. To reduce ISDN costs, with AO/DI, queries and control messages are communicated through D channels while the query results are routed through B channels.

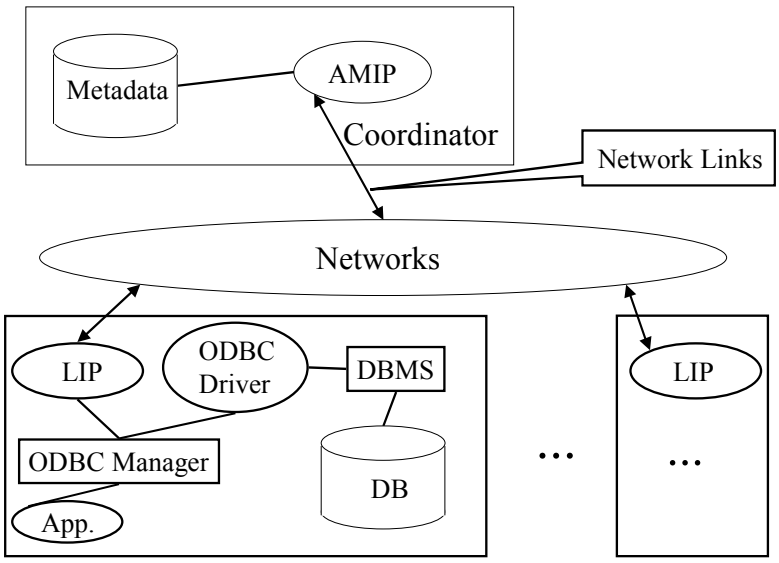


Figure 37. System Architecture of the Prototype System

3.2.5. Extending to Hybrid Networks

A distributed environment might consist of a diverse set of networks that connect a number of computers. For example, within an enterprise, computers may be connected via Local Area Networks (LANs) or Wide Area Networks (WANs), but different enterprises may choose ISDN lines to communicate with each other. It is, in fact, quite common for users to have to face a hybrid network infrastructure that is, for example, a combination of traditional networks and intelligent networks.

We have proposed a query distribution strategy for such hybrid network infrastructures based on the fundamental principles presented in section 3.2.3. Our query distribution strategy works well with traditional TCP/IP based networks as well as ISDN.

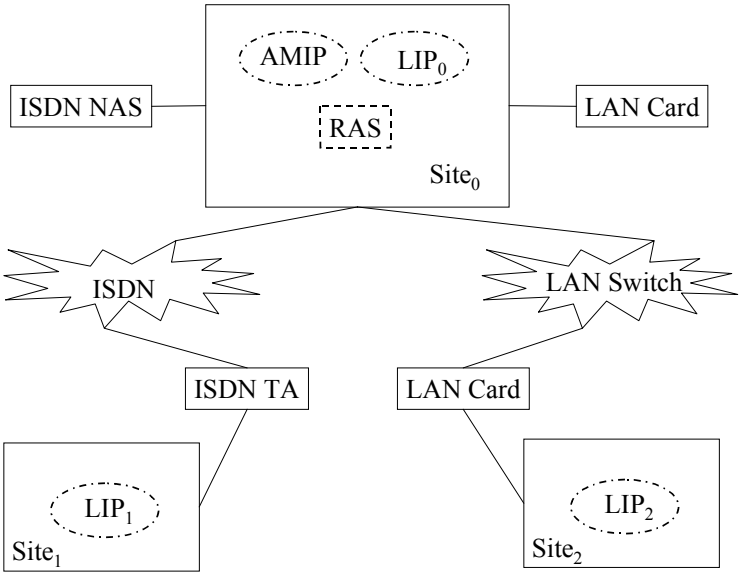


Figure 38. Example Hybrid Network Infrastructure

Figure 38 depicts a typical example of a hybrid network infrastructure. Assume that there are three sites, $Site_0$, $Site_1$, and $Site_2$, in the system. $Site_0$ and $Site_1$ are connected via ISDN lines and $Site_0$ and $Site_2$ are connected via a LAN. For this topology, we choose $Site_0$ to be the system coordinator, so the AMIP runs at $Site_0$.

A RAS server is required to run on $Site_0$ to enable the LIP running at $Site_1$ to communicate with the AMIP running at $Site_0$ through ISDN lines via Dial-up Networking. At the same time, another LIP on $Site_2$ communicates with the AMIP at $Site_0$ via the LAN. Technically, to realize this hybrid-networking version of QDA, we have designed an IP-Site map table so that all LIP sites are known to one another. The number of LIP sites dialing in through ISDN network is limited by the capacity of the ISDN NAS.

Simulating ISDN networks with WinSock

In order to add this flexibility to our prototype system, we have implemented another communication module based on WinSock. This module allows the prototype system to retrieve information from databases across a LAN or WAN using TCP/IP protocol, as well as from databases across the ISDN networks.

The WinSock-based module was designed to allow us to simulate ISDN networks via TCP/IP networks. This allows the low-level communication module to be separated from the high level QDA program and LIP program.

Without losing generality, we can place the AMIP and one of the LIPs, for example LIP1, at the same computer in our prototype. In such cases, the communication between the AMIP and LIP1 does not, therefore, use the ISDN network, but instead uses shared system memory. In fact, it uses a kind of inter-process communication, which can be implemented using a File Mapping technique. Using TAPI, we have implemented a module, MONITOR, whose responsibility is to monitor incoming calls, receive messages through the ISDN network, and to put them into the shared memory for the AMIP or the LIP. We have also implemented a dynamic linking library (DLL), Dcomm, for sending messages through the simulated "D-channnel," which is actually a socket. The WinSock simulation communication module for the computer that runs both the AMIP and a LIP is shown in Figure 39.

The arrows in the figure depict the transmission of messages. The thin solid arrows represent the transmission of the incoming messages. The MONITOR monitors line status. When an incoming message appears, the MONITOR will receive it, and then

put it into the AMIP's or the LIP's shared memory as appropriate by referring to the header flag embedded in the message. The thick solid arrows represent the transmission of outgoing messages. The AMIP and the LIPs are able to send a message to the network by calling the *SendMessage* function in the Dcomm library. The dashed arrows represent the transmission of messages between the AMIP and the LIP, which reside on the same computer in this example.

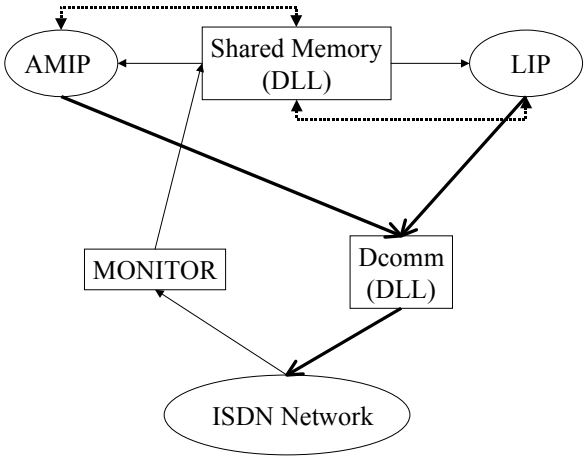


Figure 39. Winsock Communications Module

To better simulate the ISDN communication, a special data structure for table mapping is needed to describe the relationship between bound B-Channel connections and real B-Channel handles or WinSock handles. We can easily migrate the QDA program from an ISDN communication environment to a TCP/IP network environment by changing the appropriate field in this table—simply replace the B-Channel file handle with the WinSock handle .The data structure of the mapping table is depicted in Figure 40.

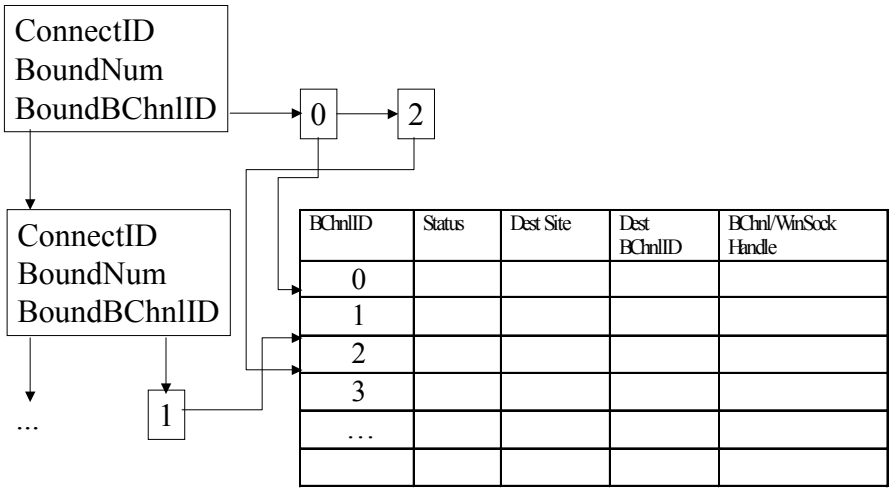


Figure 40. Communication Mapping Table

Additionally, there is another table mapping BchnlID and BChannel phone number at each LIP site. When a B channel connection needs to be set up, the LIPs find the relevant B channel number via this mapping table and then dials the number. In the WinSock version, there is no need for such a mapping table.

3.2.6. Results

Our prototype implementation has shown that our methodology is very scalable. Our query distribution methodology has no theoretical limit to the number of the participating sites. Practically, however, the scale of the system is limited by hardware capacities and, in particular, the speed of the computer at the AMIP site. Since all query requests are first sent to the AMIP site, a large number of concurrent transactions can become a great burden to the AMIP site. Regarding our query distribution strategy, the number of participating sites is not the key criteria for scaling. The key point is how

many sites will actually be involved in each particular distributed query. Our experience in database application development has shown us that the number of sites participating in each query is likely to be less than 5. As a result, even BRI service, which has only two B-Channels, can most likely meet the basic requirement for the execution of all distributed queries. Therefore, as long as the hardware components are powerful enough, the scalability of our prototype system is quite large.

We feel that several technical points could benefit from continued research to improve the performance, extensibility, adaptability, consistency, and security of our methodology:

- The hybrid network structure, which combines an ISDN network and a conventional LAN together, could be further investigated with the aim of increasing the extensibility and adaptability of this system.
- A dynamic query optimization strategy, which enables the system to change the generated execution plan dynamically based on some changing factors could be explored. Examples of factors that could be considered include the available memory size, the number of the active processes, etc.
- Encryption techniques for data transfer could be explored in order to increase the system's security against intrusion since the security of information is constantly becoming more and more important.
- The introduction of some type of dual AMIP architecture may increase the reliability and durability of this system. This, and other ways to bring fault tolerance to the system, could be explored.

4. Database Applications

This chapter describes several practical applications of the database research explored in the previous chapters. These applications revolve around the management of geospatial and environmental data using Semantic database technologies.

The first item describes the use of Sem-ODB to store the attribute data needed for implementation of a Geographic Information System (GIS). It was published as:

> T. Ho, E. Alvarez, N. Prabhakaran, D. Barton, N. Rishe, S. Graham. "Integration of a GIS and a Semantic Database System." First International Conference on Geospatial Information in Agriculture and Forestry. v. I, pp. 629-636.

The next two items describe the use of Sem-ODB to manage the storage of remotely-sensed imagery, as well as applications to visualize the stored imagery. They were published as:

> N. Rishe, D. Barton, N. Prabhakaran, M. Gutierrez, M. Martinez, R. Athauda, A. Gonzalez, S. Graham. "Landsat Viewer: A Tool to Create Color Composite Images of Landsat Thematic Mapper Data." First International Conference on Geospatial Information in Agriculture and Forestry. v. I, pp. 529-536.
>
> M. Gutierrez, N. Rishe, S. Graham, G. Rocha, X. Ye, O. Wolfson. "A Web-Enabled System for Storage and Retrieval of GOES-8 Meteorological Data from a Semantic Database." 5th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2001). v. I, pp. 62-65.

The final two items in this chapter describe the use of Semantic modeling as a design tool for creating the relational databases used to store environmental observation data collected by researchers at the Everglades National Park. These items were published as:

> N. Rishe, D. Barton, M. Chekmasov, K. Medhyanapu, S. Graham, M. Chekmasova. "Everglades Data Integration using a Semantic Database System." First International Conference on Geospatial Information in Agriculture and Forestry. v. I, pp. 567-573.

N. Rishe, M. Chekmasov, M. Chekmasova, S. Graham, I. DeFelipe. "Cape Sable Seaside Sparrow Environmental Database." 7th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2003). v. XVI, pp. 17-21.

4.1. Integration of a GIS and a Semantic Database System

4.1.1. Summary

Database Management Systems (DBMS) are implemented to allow for storage, and quick retrieval of large amounts of information. With the great demand for Geographic Information Systems (GIS), an effective database is in great need. Choosing an effective database for the GIS is very important because the overall efficiency of the GIS is affected by the database. A GIS consist of two types of datasets: the spatial data, and the attribute data. The attribute data consists of textual strings, which can be related to the spatial dataset. It is here, for the two datasets where the databases can be used. In commercial GIS the spatial data are mostly stored in an internal database. The attribute data can be stored either internally or through an external database. This section will demonstrate how the attribute data will be stored in the Semantic Database (SDB). Furthermore, a schema has been designed that can store spatial data in the SDB. Possible methods on integrating this SDB schema with a GIS will be explored. The successful integration of a GIS with the SDB would benefit users who currently incorporate GIS with relational databases.

4.1.2. Background

There has been a remarkable surge of environmental, agricultural, scientific, and academic interest in Geographic Information Systems (GIS) since their graphical nature allows planners to easily visualize the data, which aids in decision making. A GIS is a

sophisticated computer based mapping and information retrieval system, consisting of three primary components: a powerful computer graphics program, a set of analysis tools, and one or more databases which serve as the data repository. All these components must be tightly integrated in order to establish an efficient system. Of the three components, the database subsystem is the component that can be interchanged readily by the GIS designer. Choosing a database subsystem is very important because the overall efficiency of the GIS package can be increased with the selection of a suitable database for the specific need. In today's GIS application, one frequently encounters the manipulation of a large dataset. Furthermore, simultaneous multi-user access and updating is becoming more common. This section describes some of the database approaches available, and explains why an Object Oriented Semantic Database Management System would provide very good performance as a data server for GIS software.

4.1.3. Database Subsystems of GIS

As mentioned above, the database subsystem chosen can drastically affect the performance of a GIS project, as such, the GIS designer should carefully weight the different databases available. Before choosing a database for a GIS project, the designer must consider the data storage efficiency, speed of data retrieval, user visualization of the data, data security, scalability, and many other aspects of the database.

Before we discuss the merit of different databases, it would be useful to understand the data storage requirement of GIS first. In most GIS packages, there are two separate datasets. The first set is the spatial data, which are usually represented using

a non-standard data structure. These spatial data are used to store the coordinates of points, which can be combined to form arcs, polygons or other more sophisticated GIS vector data. The second set is the attribute data, used to store textual data that can correspond to different spatial data.

4.1.3.1. The Spatial Dataset

ArcInfo is an extremely powerful GIS package with the capability of opening many formats of spatial data. Also, ArcInfo has also been accepted as the GIS package of choice by the industry. Thus, we have adopted ArcInfo's spatial model into this section, which is based on the notion of vectors. Other models exist, such as those derived from the raster model. However, vector data are preferred, as it is easier for the computer to "understand" the data and manipulate them more efficiently. Furthermore, vector data model represents geographic feature similar to the way maps do.

A vector can be defined as directed line, and in the computer model, it is stored as a pair of coordinates. Thus, the basic element in the spatial dataset is a point, known in the ArcInfo jargon as a node or a vertex. The nodes and vertices are then joined together to form a line, known as arc. The difference between nodes and vertices are that the former are points that serve as end points of an arc, while the latter are intermediate points of the arc. Notice that each arc contains two or more points, thus the arc need not be a straight line, but it can be a curve of any kind. Optionally, arcs in turn are grouped to form *polygons*, which encloses an area on the map. Finally, all the spatial data can be grouped into layers called coverage.

ArcInfo uses a variety of data structures to stores this spatial data. The arc attribute table (AAT) describes arcs, nodes attribute table (NAT) describes nodes, and the polygons' and points' data are described by polygon attribute table (PAT). In addition to these feature attributes data, ArcInfo also employs the Arc-Node data structure to stores the X, Y Coordinates of all the points that make up an arc. Furthermore, Tic (TIC) and Boundary (BND) tables are used to define the geographic extent of a coverage. From these basic data structure, ArcInfo can build the database it needs to create digital maps, which would serve as the foundation of a Geographic Information System. For more information on how ArcInfo structures the spatial data, refer to [87] and [88].

4.1.3.2. The Attribute Dataset

The second dataset used by ArcInfo is the attribute data. They are stored in the attribute table, and consist of textual data describing the geographic features in the spatial data. In contrast to the spatial data, where ArcInfo defines the structure of the data structure, the structure of the attribute dataset is in the complete control of the GIS project designer. All that is required is that a link is established between the attribute data and the spatial data. In relational databases this is accomplished by storing the key that identifies the spatial data in the attribute dataset, thus allowing a database join command to be performed. As an example, one can have the following table to store the information about soil-type and vegetation-type of a polygon.

COVER-ID	SOIL-TYPE	VEGETATION-TYPE
60	Marl	Brazilian Pepper
. . .		

In this table, the *COVER*-ID provides an identifier that can be used to find out which polygon the record is referring to, thus a join can be established between each record and the GIS polygon with the same *COVER*-ID. The other two fields are attribute datum used for description purposes. This is because spatial data alone does not describe a real world object. Attribute data must be used to compliment the spatial data so that GIS users can effectively model their area of interest.

4.1.4. External Databases for GIS

By default, the attribute data is handled by the ArcInfo database subsystem, which uses a relational database model. If the user desires, these attribute data can be stored in an external database. ArcInfo can connect to a multitude of commercial relational database, such as Oracle, Sybase, Informix, and many others. This connection allows users to connect to their existing database and relate their attribute data to the spatial data. Even for users who design a new GIS database, the ability to store the attribute data in an external database has many advantages.

Typical commercial databases such as Oracle or Sybase offer a suit of data management tools that help to maintain data integrity and accuracy [89]:

- Concurrency mechanisms-that allow different users to query the database while other parts are being updated by other users.
- Transaction-based update where either all or none of the updates are committed.
- Schema integrity is always maintained, since only the Database Administrator (DBA) can modify the schema of the database.
- Backup and recovery mechanism to restore the database into a consistent state.

- Availability of data for other programs. The data maintained by the DBMS can be accessible by a multitude of programs, such as ArcView, or custom program that display the data over the web.

These and many other advantages are offered by a typical commercial relational DBMS. Florida International University's High Performance Database Research Center, has developed an efficient Semantic DataBase (SDB) with an object-oriented framework. In addition to the above benefits, SDB provides many other benefits that typical relational databases do not provide. Some salient features of SDB are [8]:

- Optimum storage space requirement
- Efficient query processing
- Indexing on all attributes of the database
- A rich collection of GUI-based visual tools to create, load, and maintain a schema, as well as to query the database.
- High level query language support (SDB queries are simpler than SQL queries)
- Unlimited precision of numerals and fractions
- All physical aspects of the representation of the information are transparent to the users. This creates a greater potential for optimization of internal representation.

The SDB provides a standard Open DataBase Connectivity (ODBC) interface; thus this project sets out to interface ArcInfo with the SDB. We will configure ArcInfo to store all the attribute data in SDB, thus providing a way for ArcInfo's user to break away from the relational database and reap the benefits of SDB.

4.1.5. Integration of SDB and GIS

As mentioned before, all GIS software depends on a database to store its data. In the case of ArcInfo, there are several ways to do this. ArcInfo defaults on using the integrated database subsystem, INFO, to manage all the attribute data. ArcInfo can be interfaced with an external database via ODBC. (In the Unix environment, Database Integrator is used instead.) There is a multitude of commercial databases that supports ODBC: Oracle, Sybase, Access, etc. Once the connection is established between ArcInfo and the external database, the Arc RELATE command allow the users to join spatial and attribute data that is accessible like a single coherent database.

Using the same approach, the integration of ArcInfo with SDB is done via ODBC. Furthermore, the SDB provides a relational interface so that ODBC compliant software can query the SDB. Although SDB uses a semantic model, no special understanding is required from the ODBC client. As such, the Arc RELATE command continues to work as if the SDB was a relational database. While accessing the data stored in the SDB, ArcInfo's users will only see the same differences as migrating from INFO to other commercial database such as Oracle. In short, the integration of SDB and ArcInfo will largely be seamless and transparent to the user; thus the benefit of SDB can be obtained with little changes in work-approach.

After the connection is established, the database must be created so that the attribute data related to the GIS project, can be stored. In the current version of SDB, the SQL CREATE command is not supported via the ODBC interface. This means that the user cannot create a table in the SDB from within ArcInfo. However, this is not a serious draw back. As anyone who has created a GIS project can testify, database table creation

using commands alone is not easy, to say the least. Many would rather use visual tools, instead of word-descriptions, to create database tables and establish links between them (relations). The SDB provides a user friendly, GUI based visual tool to design and implement the database tables for use in the SDB and GIS. Figure 1 shows this visual tool in action. Once the database schema is designed, the correct database in SDB can easily be created. There is no SQL or other coding involved. Overall, the integration of SDB and ArcInfo provides a better database management environment than the traditional platform.

Another advantage of SDB is the improved speed of processing queries. This is accomplished by decomposing the query into non-redundant atomic queries that can be executed concurrently. Thus, these requests take close to the minimal number of disk accesses required to retrieve the data [10].

4.1.5.1. Image Storage in SDB

Modern database can store more than textual data. Many commercial database can store Binary Large Objects (BLOBs). The implication is that raster images, which are often used as backdrop of GIS vector data, can also be stored in the database. Current usage does not usually exploit this feature of the database, as many see the raster image just as a file instead of a collection of records that are the realm of the database domain. However, the fact is, images can be stored in the database, and there could be several advantages to this approach. First, the images will be stored in a consistent, centralized location. Second, many descriptive data about the image that is usually not stored (or stored sporadically as files where users cannot access them easily) can now be

maintained by the database. For instance, if a variety of satellite and/or aerial images are used to complement the vector data of a GIS project, many descriptive data about the image would be useful in cataloging them. Information such as date of data acquisition, coordinates of the images, data provider, description of the image, plus others information can become very handy in allowing the user to judge the usefulness of an image for a particular task at hand.

It is in binary storage where SDB has a real advantage over traditional relational databases. The object-oriented framework in which SDB was designed allows a very efficient data storage and manipulation [16]. For instance, automatic compression can be applied to the image at the database level. This will reduce the storage space requirement, and the compression can be performed independent of the image format. This is advantageous because smaller files means reduced hard disk access. This, plus the optimized query from SDB, allows large images to be retrieved faster than many commercial databases. Preliminary tests at the center showed very promising results.

4.1.5.2. External Representations of Spatial Data

Another aspect that is natural for a database to handle is the spatial data. To create a vector based map, the GIS software needs to obtain the coordinates of a large number of points. Furthermore, the GIS package needs to know how these nodes are connected to form line segments (arcs), and how these arcs in turn form polygons. In this section, a SDB schema that can also be used to store this spatial dataset will be discussed.

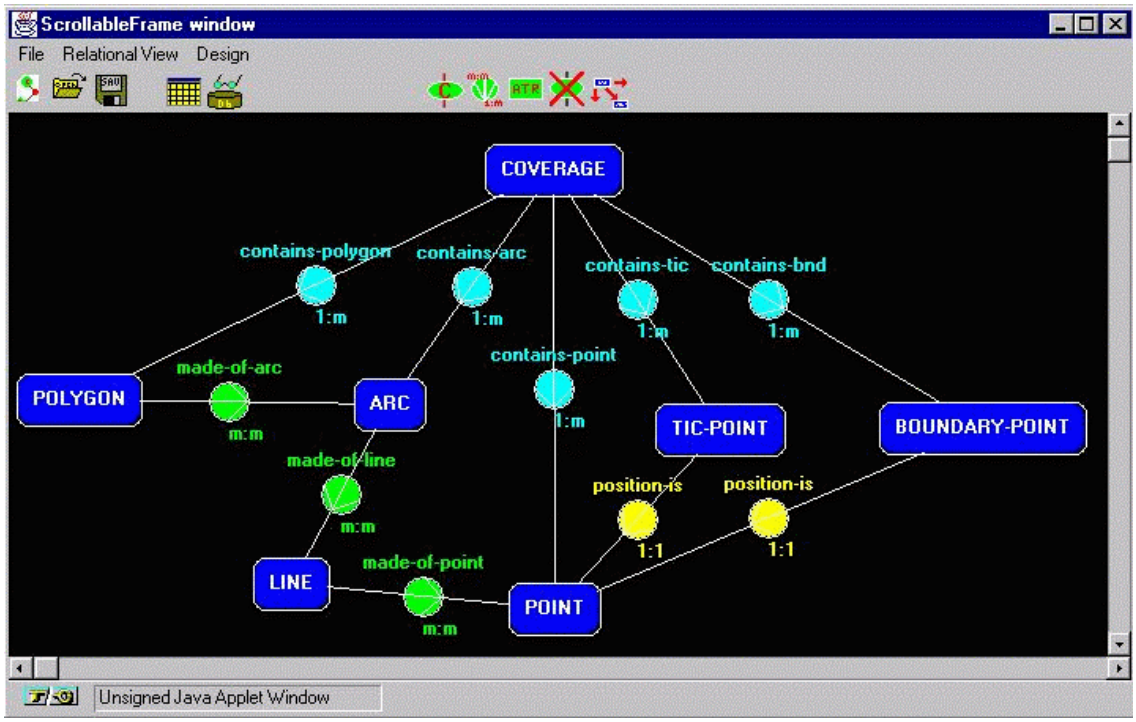


Figure 41. SDB Schema That Can Store Spatial Data

Figure 41 shows the schema necessary for storing spatial data in the SDB. Categories are shown as rectangles, while relations between the categories are shown as circles. The attributes of any category can be retrieved with a double click on the corresponding rectangle (see Figure 42). The diagram makes it very easy to understand how the SDB will store the spatial data and how the data is related. The diagram shows us that POINTs are grouped to form LINEs, which in turn form ARCs, and POLYGONs. TIC-POINT and BOUNDARY-POINT are categories derived from POINT; they have two extra attributes added to them: latitude and longitude, which give them geographic dimensions and thus identify the coverage to a particular area on the globe. If different geo-coordinate is desired, they can also be easily changed using the visual Semantic Editor. For more information of SDB design, refer to [8].

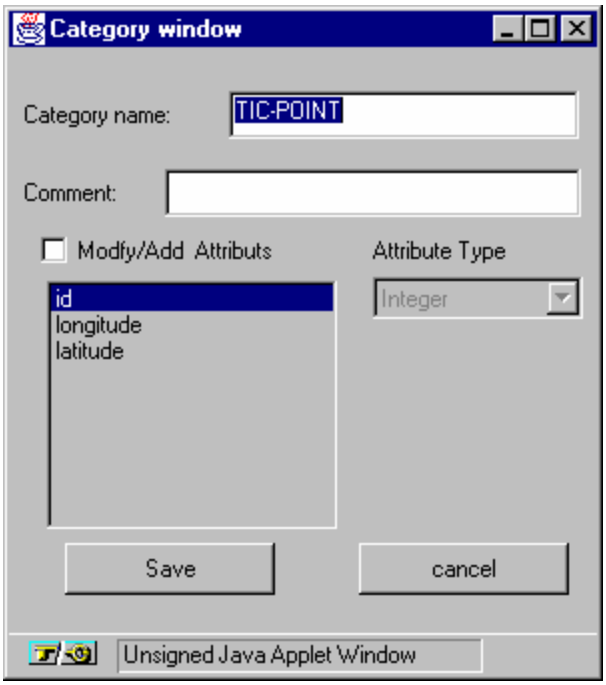


Figure 42. Attribute Data of TIC-POINT

4.1.6. Applications

The integration of GIS and SDB can benefit several communities, including agriculture, environmental and forest management, which are currently using GIS for their research and studies. For example, the Database Center has a project with the Everglades National Park. One of the research projects in the park is the control of Brazilian Pepper, an unwelcomed exotic plant that has many adverse effects on the flora and fauna of the Everglades. This forestry related management part of the project calls for a database schema that requires, among other information, plot of lands that have been invaded by the Brazilian Pepper, fields where herbicides or fire has been used to control this exotic plant, as well as the effect on the land by such actions.

The experience gained by the staff in the center is that the visual database schema designer provided by the SDB is a very effective tool to create a database. The

traditional relational database was hard for non-expert to understand; thus the participation level from the client was usually minimum. However, when we used the highly visual SDB schema design tool in a project with the Everglades National Park, they found it extremely easy to understand, and participated actively in the design of the database.

While this Everglades project was not born because of the GIS requirement in data analysis, it still nonetheless provided a good example of how the GIS user can benefit from the use of SDB. In fact, an extension of the current project is being evaluated, where geographic information will be added to the current text-only database, effectively converting it into a GIS project. This extension will naturally call for a modification of the current database. However, since the schema is in a highly visual and easy to understand format, such task is much easier than it had been traditionally. Furthermore, the park staff will not hesitate to perform changes, since they completely understand the project.

An additional advantage of the SDB is its flexibility. Users can use the semantic high level language to perform the queries, or they can use SQL. The semantic database browser tool allows relational-oriented users to view the schema of the semantic database as relations and attributes. Also, they can write SQL queries to retrieve all textual, spatial and image data similar to any standard relational database queries. Once the query is submitted, this tool translates the SQL query into the semantic language and provides the result to the users. But this process is transparent to the users making it very easy for users not familiar with the semantic model.

4.1.7. Results

The SDB provides a very robust way of storing both conventional (textual data) and non-conventional data (spatial data and images). Furthermore, the visual tools provided by SDB provides a user-friendly interface that allows even non-experts to create an efficient database, as well as query and display the data. This current project of SDB-GIS integration extends the above-mentioned benefits to the GIS users, who are usually locked into the traditional relational database.

4.2. Landsat Viewer: A Tool to Create Color Composite Images of Landsat Thematic Mapper Data

4.2.1. Summary

Data from Landsat Thematic Mapper (TM) sensors detect reflected radiation from the Earth surface in the visible and infrared wavelengths. The characteristics of the TM bands can be selected to maximize their capabilities for detecting and monitoring different types of the Earth resources. The ground area covered by one Landsat scene is over 34,000 square kilometers and represents about 260 MB of data.

This section provides a description of a web page interface to a Landsat TM Semantic Database, being developed at the High Performance Database Research Center (HPDRC) at Florida International University. The web interface allows the user to graphically select areas of the Earth to be examined showing latitude and longitude coordinates. The user can further choose the size of the Landsat scene or quad and the color composite image to view based on the seven available sensors. The color composite images are generated in real time in 24-bit color and are subject to various user

selected picture enhancement algorithms before being recomposed and exported to the client program in a standard image format. This Landsat Viewer facilitates image processing from the Internet.

4.2.2. Background

It has taken the database community some time to recognize the impact of the uniqueness of spatial data. In the 1970's and 1980's, the database community lumped every kind of data other than fixed-format records (including spatial data) into a heterogeneous group called 'non-standard data'. It was tempting to extend relational database technology, with this simple conceptual structure, to handle all kinds of data. However, relational data is not just a way to represent data, it also implies or suggests certain access algorithms that are particularly efficient on data naturally represented by rows and columns. It is true that much business data is of the tabular form and that this data lends itself to such regular access patterns. However, if we force spatial data into tabular form, for example by introducing relations faces, edges and vertices, this may have harmful consequences. Geometric proximity is not reflected by proximity in memory. For example, all vertices no matter how far apart in space are stored contiguously in the same relation, whereas a vertex and its incident edges and faces are scattered all over storage. This may have grave consequences when data is stored on disk, where instead of accessing one entire object as a unit we may have to gather bits and pieces of this object in many separate disk accesses [90].

4.2.3. Semantic Database

The HPDRC's Semantic DBMS is based on the Semantic Binary Model. In the Semantic Binary Model, the information is represented by logical associations (relations) between pairs of objects and by the classification of objects into categories. The Semantic Binary Model is the most natural and convenient way of specifying the logical structure of information and for defining the concepts of an application's world. It is represented in the form of a semantic binary schema [8].

4.2.3.1. Description

The Semantic Database models are potentially more efficient than the conventional models for two main reasons. The first is that all the physical aspects of the representation of information by data are invisible to the user and the second is that the system knows more about the meaning of the user's data and about the meaningful connections between such data. The first reason creates a potential for optimization by allowing more changes without affecting the user programs. The second allows this knowledge to be utilized to organize the data so that meaningful operations can be performed faster at the expense of less meaningful operations [8].

The efficient retrieval and updates are a requirement of the semantic database. Requests are maximized by decomposing queries into atomic retrieval operations and each atomic retrieval request normally requires only one disk access. A transaction is composed of a set of facts to be deleted from the database, a set of facts to be inserted and additional information needed to verify that there is no interference between transactions of concurrent programs. A transaction can be generated by a program fragment

containing numerous update commands contained among other computations. However, until the last command within a transaction is completed, the updates are not physically performed but instead are accumulated by the DBMS. Once the transaction is completed, the DBMS checks its integrity and performs the update. This insures the consistency of the database, with regard to applications and users. Until the transaction is completed, its effects are invisible [8].

The Semantic Database is perceived by its users as a set of facts about objects. These facts can state that the objects belong to a category, they can state that there is a relationship between objects or they can be fact relating objects to data, such as numbers, texts, dates, images, etc [8]. HPDRC's Semantic DBMS contains semantic facts and inverted semantic facts. This fact inversion scheme assures efficiency of queries including range queries and content access and also exhibits low entropy of data blocks, which facilitates compression.

- Compression will not negatively affect the efficiency of a search since the logical B-tree order between data and index blocks will not be altered by such compression.
- Compression will reduce the amount of I/O time required for long sequential data transfers which are commonplace in scientific database applications.
- In a distributed system, compression will significantly decrease the amount of communication overhead.
- Data compression will be transparent to the user.

Even without special compression algorithms, the semantic DBMS is very storage efficient. However, contemporary compression techniques can help achieve significant savings.

The mathematical abstraction of the relational model has allowed the introduction of powerful and easy -to-use languages for retrieval and updates of databases. The semantic model however, offers a higher degree of abstraction, which results in more concise user programs, speedier processing (due to optimization), and a wealth of other features. Relational databases are good for general conventional database applications. However, in situations where the structure of information is complex, or where greater flexibility is required (objects with unknown identifiers, or objects moving from one category to another, etc.), or where non-conventional data is involved (spatial data, long text, images, etc.), semantic databases need to be considered.

4.2.3.2. Landsat Semantic Schema

We used a Semantic Binary Database for the storage of Landsat TM data. The first step involved in creating the database is the design of the schema. HPDRC has acquired some Landsat TM data of scenes and quads observed by Landsat 4 and 5. These spatial data along with its meta-data are integrated to the database by the schema design. Figure 43 shows the current schema design for the Landsat TM database.

Description of schema:

- LOCATION—category (A catalog of locations defined by path and row which is the coordinate system used for Landsat observations)

- OBSERVATION—category (A catalog of observations which refers to observations made by a Landsat satellite for a particular date)
- QUAD—category (A catalog of quads which is a particular area of observation specified by the quadnr)

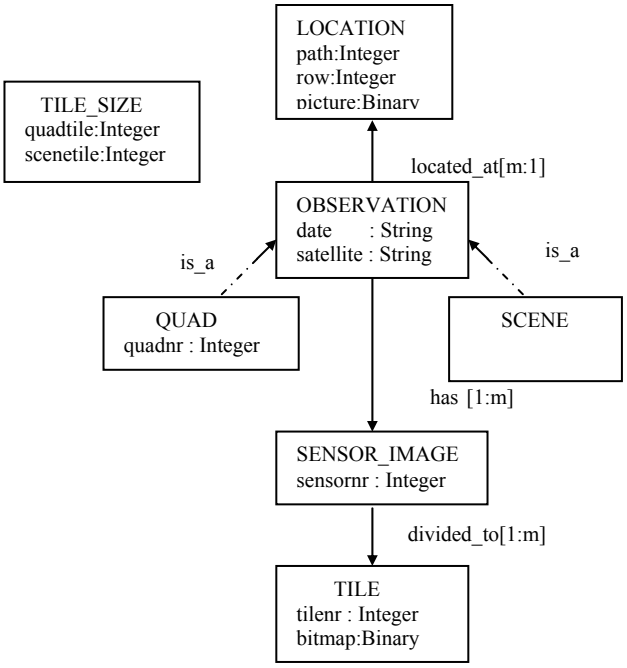


Figure 43. Schema for Landsat Thematic Mapper database

- SCENE—category (A catalog of scenes which is the area of observation)
- SENSOR_IMAGE—category (A catalog of images observed by a sensor on-board Landsat satellite sensor specifies the sensor which made the observation)
- TILE—category (A catalog of tiles which are segments of a sensor image)
- TILE_SIZE—category (A category of TILE_SIZE which contains the sizes of quad and scene tiles)

- located_at –relation from OBSERVATION to LOCATION (m:1,total) (An observation must have a location that it observes. There are many observations with the same location)
- divided_to–relation from SENSOR_IMAGE to TILE (1:m) (A sensor image is divided in to smaller segments called tiles)
- has–relation from OBSERVATION to SENSOR_IMAGE (1:m) (An OBSERVATION consists of many sensor images observed by different sensors in the satellite)
- path – attribute of LOCATION of type Integer (The path number of the coordinate system)
- row – attribute of LOCATION of type Integer (The row number of the coordinate system)
- picture – attribute of LOCATION of type Binary (The image depicting the region covered by scene)
- date – attribute of OBSERVATION of type String (The date when the observation was made)
- satellite – attribute of OBSERVATION of type String (The name of satellite which made the observation)
- quadnr – attribute of QUAD of type Integer (The quad number which specifies the area of observation)
- sensornr – attribute of SENSOR_IMAGE of type Integer (The sensor number which specifies the sensor that observed the image)

- tilenr – attribute of TILE of type Integer (The tile number which identifies the tile)
- bitmap – attribute of TILE of type Binary (The binary data observed by the sensor on-board the satellite)

The LOCATION category specified by a path and row number gives a specified location for an OBSERVATION category. An OBSERVATION object has a relation located_at, of cardinality [m:1], with LOCATION specifying that there are many observations for a particular location. Since the relation is total, every OBSERVATION must have a particular LOCATION. Picture attribute contains an image of the area covered by the scene for the particular location . OBSERVATION is described by a date specifying the date of observation, and the name of satellite, which made the OBSERVATION. An OBSERVATION object could be either a SCENE or QUAD (specified by a quad number). Every OBSERVATION can have up to 7 sensor images observed by the seven instruments on board Thematic Mapper instrument. This is specified by a has[1:m] relation and SENSOR_IMAGE category. Sensor_Nr specifies the sensor that made the observation. Each sensor image is divided into smaller sub-images called tiles. A scene's sensor image is divided into 930 tiles and a quad's sensor image is divided into 378 tiles. The justification is that in most applications only a very small percentage of the sensor image is used for viewing by the user. Hence, when a request is made to view a particular part of the image, queries to the specified tiles can be made and the image created, instead of retrieving the whole image at once and selecting out regions to view. This methodology will enhance the performance in most cases when only a small number of tiles are retrieved. This is captured by a divided_to [1:m] relation

from the SENSOR_IMAGE to TILES category. TileNr specifies the tile and Bitmap attribute is used to store the observed data.

4.2.4. Design of the Application

The overall structure of the application consists of design and implementation of 3 main components :

- The design and implementation of a storage-retrieval medium for Landsat images. In our case, it is a Semantic database developed at HPDRC. This is described in the previous section.
- The design and implementation of a client program that acts as the front-end of the application.
- The design and implementation of a server program that queries the database to fulfill the requests of the client.

The client program will interact with the user to compose a query using easy-to-use Graphical User Interface (GUI). It will send the user's request to the server for processing, and will display the results that are received from the server for a particular request.

The server program acts as the back-end of the application interacting with the database to fulfill the requests of the clients. This program will query the database to obtain for the client's requests and communicate with the client to send and receive results

4.2.4.1. Client Program

The main task of the client program is to obtain the user's query using an easy-to-use GUI. The client shows a map like Figure 44 (map of US) from which the user select a state (for example Florida, see Figure 45).

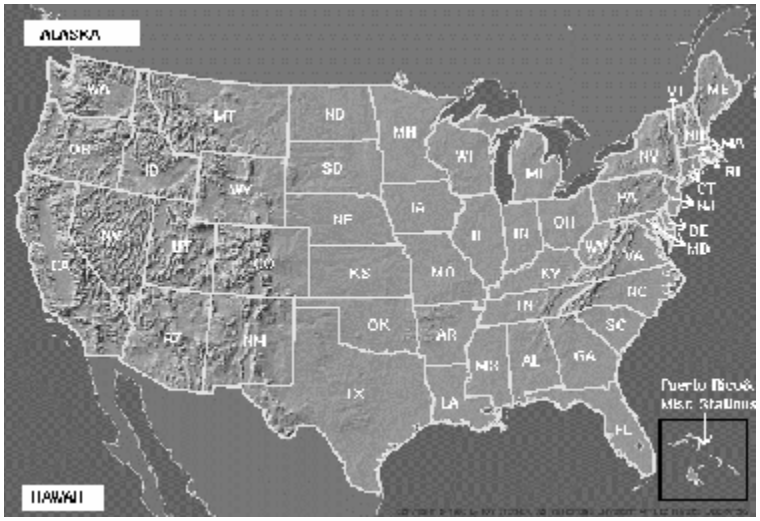


Figure 44. Map of the US from which the user selects a state of interest

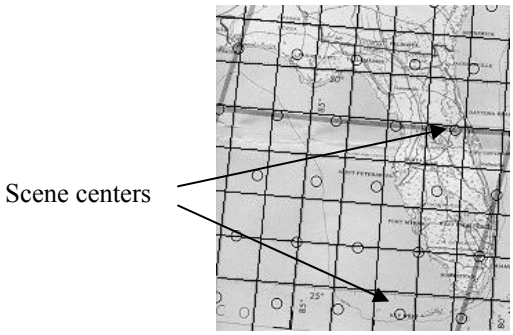


Figure 45. Map of the state of Florida with the scene centers marked by circles

The user in most cases requires only seeing a small area of this region. The map of Florida is marked with scene centers. The user selects a scene center of interest, which gives the client program a particular path and row number that the user requires. This information is transmitted to the server program as Query1. The results of Query1, which

are received by the client program, contains meta-data on all the Landsat Thematic Mapper images present in the database for the selected path and row. Also, a name of a picture file, which contains the region covered by the scene for the selected path and row, are received. The picture file depicting the selected region is displayed from which the user selects a smaller region of interest.

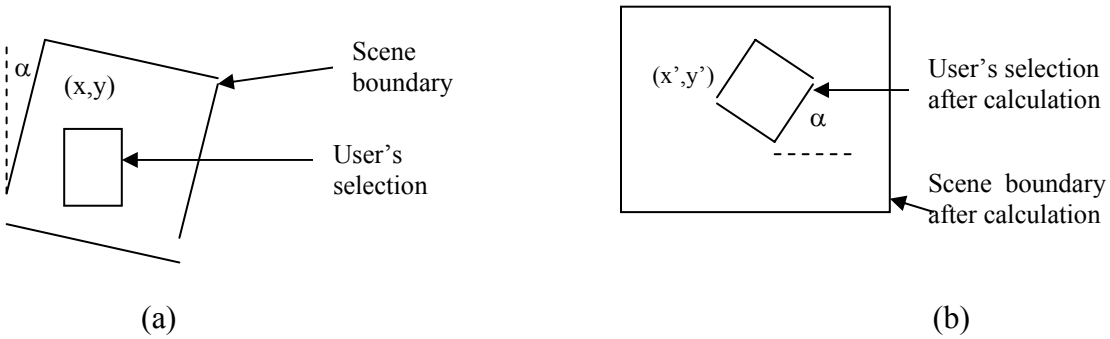


Figure 46. (a) The alignment of an angle α to the horizontal axis in the scene boundary

(b) The scene boundary and user's selection

The client program then calculates the tiles for quads and scene involved in the selected region. There is an alignment in the scene boundaries of the displayed picture (see Figure 46 (a)) when compared to the user's selection rectangle. This alignment is due to the path Landsat satellite takes as it observes the Earth. Thus, we need to find a rectangle, engulfing the user's selected rectangle, and whose sides are parallel to the sides of the scene boundaries to calculate the tiles required to be queried for the user's selection. This calculation is performed using the following method.

Assume that there is an alignment angle of α as shown in the Figure 46 (b). We move each vertex (x,y) of the user's selected rectangle by an angle of α to calculate (x',y'). This is performed by the following equation:

$$x' = x \cos \alpha - y \sin \alpha$$
$$y' = y \sin \alpha + y \cos \alpha$$

The same calculation is performed for the scene boundary co-ordinates.

The engulfing rectangle's, shown by the dashed lines in Figure 47 (a), are calculated as points A, B, C, D

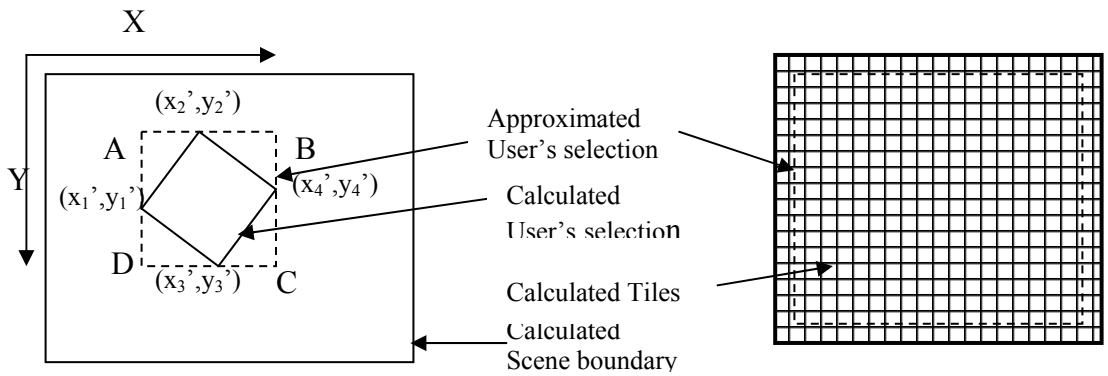


Figure 47. (a) Approximated user's selection (b) The selection of tiles to the approximated user's selection

With these co-ordinates we are able to approximate the user's selected rectangle and calculate the tile numbers for the scene and quad that need to be queried (Figure 47 (b)). Once the tiles required by the user's selection are calculated, it is checked whether the tiles are present in the database. Since the results of Query1 contain all meta-data of Landsat images present in the database for the particular path and row, this can be easily performed. Next, the dates available for the selected region are displayed from which the user selects a date of the observation he/she prefers to view the image. Next, the sensor images available are computed and displayed for the user to select to produce color-composite images. Note that we require sensor images to be computed because the user's selection may span across multiple quads and it is possible that the database may not

have sensor images of a quad selected. This is resolved by finding the intersection of sensor images present in database for the user's selected quads. Also, image enhancement could be done to the resultant image by applying a filter. Next the selected tiles, date, sensor numbers and filters are composed into a query (Query2) and transmitted to the server program to process. Finally, the result of Query2 is an image created by applying the selected sensors to red, green and blue and the selected filter. The client program will display this image.

The client program is implemented as a JAVA applet running on a WWW browser. The states (e.g. florida.html) where the user clicks on a preferred state to view an image, is implemented in HyperText Mark-up Language (HTML) with path and row numbers embedded as parameters. The connection between the client and the server is handled by a TCP/IP (reliable byte stream) connection using socket implementation.

4.2.4.2. Server Program

The main method or controlling body of the server program provides two major functions. It opens the Landsat database and waits for a client to contact. When a client does make a connection, it creates a thread or process and lets the client communicate with the new process for its future transactions as it continues to wait for more clients.

The client requests for two different types of queries from the server.

1. Query1: For a given path and row number, provide meta-data on all the Landsat quads and scenes present in the database for the particular location along with a picture of the region.
2. Query2: For a given path, row and sensor numbers, query the bitmaps for the selected tiles.

The server waits for query from the client. On receiving a request, checks whether it is of type Query1 or Query2 and perform the necessary tasks accordingly. It then sends the results for the queries to the client and waits for another query. The server program is implemented in C++. It uses the C++ interface developed for the Semantic Binary Database at HPDRC to query the database. The server runs on a Sun Sparc station using Solaris as the underlying Operating System. Note that the server side is kept simple intentionally so as to make the application easily portable between different schemas of the database. The main components that require to be modified are the two queries.

4.2.5. Image Enhancements and Color Composite Images

The application gives the capability of producing color-composite Landsat images by applying any of the different sensors images to the RGB color model. Applying filters can further enhance these resultant images. Landsat viewer has the following filters:

1. Linear Contrast Enhancement [91]
2. Low Frequency Filtering - mean filter (LFF5,out) [91]
3. High Frequency Filtering (HFF5,out) [91]
4. Linear Edge Enhancement : Vertical [91]
5. Linear Edge Enhancement : Horizontal [91]
6. Linear Edge Enhancement : NE Diagonal [91]
7. Linear Edge Enhancement : SE Diagonal [91]
8. Emboss East [91]
9. Emboss NW [91]
10. Compass Gradient Masks - North [92, 93]
11. Compass Gradient Masks - NE [92, 93]
12. Compass Gradient Masks - East [92, 93]
13. Compass Gradient Masks - SE [92, 93]
14. Compass Gradient Masks - South [92, 93]
15. Compass Gradient Masks - SW [92, 93]
16. Compass Gradient Masks - West [92, 93]
17. Compass Gradient Masks - NW [92, 93]
18. Compass Gradient Masks - vertical [94]
19. Compass Gradient Masks - horizontal [94]
20. Compass Gradient Masks - Mask R, diagonal [94]

21. Compass Gradient Masks - Mask S, diagonal [94]
22. Laplacian Convolution Masks - Mask T [92, 95]
23. Laplacian Convolution Masks - Mask U [92, 95]
24. Laplacian Convolution Masks - Mask V [92, 95]
25. Nonlinear Edge Enhancement - Sobel edge detector [91]
26. Nonlinear Edge Enhancement - Robert's edge detector [91]

Items 2 - 26 are categorized as *Spatial Convolution Filtering*. Items 4 - 24 are filters used in *Linear Edge Enhancement*. We used 3 X 3 convolution masks in applying these filters.

Each color in the RGB color table has intensities of up to 255 colors, providing for 2^{24} different colors. (24 Bit Color) Each of these Bands has a very limited range in the RGB Color Table. Putting the Three Band Combinations to the Screen in RGB is not enough to fully see the entire image in detail. A Landsat Image combining three bands without any further image processing will produce an image in very low contrast, some features may be hard to see with the human eye. As a solution to this, we use an image processing algorithm called "Linear Contrast Stretching" that stretches the colors in the image to use the entire color. This algorithm must be done with each band used in the RGB Combination. For each band, you find the maximum digital number, and the minimum digital number. With these two numbers, you can apply this formula for each pixel's digital number (DN) in the band:

$$DN' = (255*(DN - Min))/Max\text{-}Min$$

Using this formula, the image is utilizing the entire Color Table, making it easier to view the image in full color. To explore the full information content of Landsat images, higher order image processing algorithms such as Histogram Equalization, and Band Ratios can be used.

4.3. A Web-Enabled System for Storage and Retrieval of GOES-8 Meteorological Data from a Semantic Database

4.3.1. Summary

Meteorological data is a major component in the study and understanding of environment behaviors and their impact on society. Examples of the effects of hazardous weather conditions range from mild structural damage and agricultural and commercial losses to massive death and destruction. Meteorological data enables the prediction and analysis of weather phenomena and its possible life-threatening and property-damaging effects.

The Geostationary Operational Environmental Satellite (GOES) system provides timely global weather information, including advanced warning of developing storms. Using GOES imagery, meteorologists are able to better study and understand atmospheric circulation patterns and behaviors. Capable of generating over 18 GB of meteorological data per day over the eastern United States, the GOES-8 is the first of this new generation of Earth-observing satellites.

The High Performance Database Research Center (HPDRC), in conjunction with the NASA Regional Applications Center (RAC), has developed a web-enabled system which houses several gigabytes of GOES-8 meteorological data during each hurricane season. This system uses semantic database (Sem-ODB) technology to efficiently store and retrieve GOES-8 atmospheric data. Combining newly developed technologies and algorithms with the broadcast capabilities of the World Wide Web, HPDRC and NASA

RAC make this vast source of information available to the general public and fellow scientists around the world.

4.3.2. Background

Understanding atmospheric behaviors aids meteorologists and weather forecasters in providing improved advanced warnings of thunderstorms, flash floods, hurricanes, and other severe weather conditions. Accurate forecasts save lives, preserve property, and benefit agricultural and commercial interests.

Aided by the rapid evolution of remote sensing technologies, the usage of meteorological satellites has become a major method for the acquisition of weather imagery and data. Data acquired by these satellites or satellite systems provides highly accurate and detailed information facilitating the study, analysis and understanding of weather phenomena. One such system, the GOES, is a basic element of US weather monitoring and forecast operations and a key component of the National Weather Service (NWS) modernization program. The GOES system is an essential cornerstone of weather observations and forecasting used for short-range warning and "now-casting."

Seeking to encourage further research advancements and development, HPDRC has designed a web-enabled system housing large amounts of GOES-8 weather and atmospheric data. This system uses semantic database technology to achieve efficient storage and retrieval of large amounts of data. Provided with a Web-interface, this database system makes queries of weather measurements, imagery and data available to a vast area of the scientific community.

4.3.3. A Web-Enabled Weather System

Aided by semantic database technology [8], our innovative weather system efficiently acquires, stores, organizes and provides access to GOES-8 data enabling its use for the investigation of oceanographic and meteorological phenomena. Four major components comprise this web-enabled system. These components are: a vast archive of GOES-8 meteorological data, a database system capable of efficiently storing and manipulating large amounts of data, an ingest system constantly acquiring up-to-the-minute imagery and measurements, and a World Wide Web interface.

4.3.3.1. GOES-8 Meteorological Data

Spacecraft and ground based systems work together to accomplish the GOES mission of providing weather imagery and quantitative sounding data for weather forecasting and related services. GOES satellites provide the continuous monitoring necessary for intensive data analysis. They circle the Earth in a geosynchronous orbit, providing a constant vigil for atmospheric "triggers" of severe weather conditions such as tornadoes, flash floods, hailstorms and hurricanes. When such conditions develop, these geostationary meteorological satellites monitor storms and track their movements.

Provided with a five-channel imager and a nineteen-channel sounder, the GOES meteorological satellites acquire high-resolution visible and infrared data as well as temperature and moisture measurements of the atmosphere. Furthermore, they introduce two new features. The first, flexible scanning, allows small-area imaging; a feature that allows the focus of localized weather trouble spots to improve short-term weather forecasts and now-casts. The second feature is simultaneous and independent imaging

and sounding. This feature allows the usage of data from both, imager and sounder instruments, to increase the accuracy of weather analyses and forecasts.

In this manner, data provided by the GOES-8 satellite, also called GOES-EAST, includes detailed weather measurements, frequent imagery, and new types of atmospheric soundings. Located at 75°W, the GOES-8 satellite generates data covering most of the eastern US and South America. Furthermore, this meteorological data, combined with information from new Doppler radars, aids water resource managers as they make critical decisions about allocating precious water resources.

4.3.3.2. The Database System

Based on Sem-ODB technology, our weather database is primarily used for monitoring of severe tropical weather phenomena during hurricane season. Specifically, our Hurricane Season 2000 database archives over 26GB of GOES-8 Continental United States (CONUS) imagery and data. A high growth potential is observed due to the constant acquisition of imagery. Implementation of the semantic database model and an efficient schema design enable the fast querying and efficient manipulation of such considerable amounts of data.

The database is categorized by hurricane season, which in turn is categorized by date. Figure 48 presents the semantic database schema.

The schema contains four Categories: HURRICANE_SEASON, SDATE, SCANLINE, and DATA. The HURRICANE_SEASON category contains the year of the hurricane season, and the starting and ending Julian day for the season, this distinguishes one hurricane season from another and allows storage of multiple seasons in one

database. The HURRICANE_SEASON category contains many SDATE, which describes the month, day, and time of the data set. Each DATA object may contain many channels and many instruments. Each Channel can have many SCANLINEs, which contain data of each pixel in a scanline.

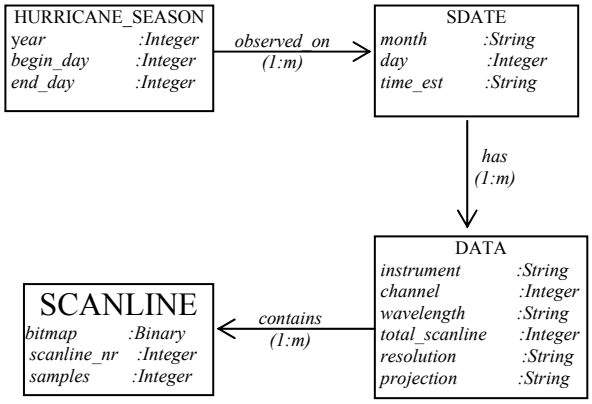


Figure 48. Hurricane Database Semantic Schema

For GOES-8 GVAR, each CONUS data file is approximately 65 MB in size and is stored by scanlines. Storing the data by scanlines permits efficient scaling down of the image during retrieval. To scale an image that is stored by scanline with the scale factor N, only every Nth scanline needs to be retrieved, and then every Nth pixel within each scanline is picked up. It efficiently reduces retrieval time for other applications that need information about only specific area or scanlines in a similar way. This scheme improves the efficiency and flexibility of database retrieval for related applications.

4.3.3.3. Ingest System

The automatic ingest system uses UNIX scripts to communicate with the ingest machine, provide WEB access, and automatically load data into the hurricane database.

This system is composed of four main modules: (1) File Downloading, (2) Data Processing, (3) Image Generating, and (4) Database Updating.

Figure 49 shows the control and data flow of the whole system. The ovals in the figure contain mainly processes the rectangles contain data and the arrows represent data/control flows.

The functionality of each module can be described as follows:

(1) File Downloading: A list of the current files in the system is downloaded via FTP from the ingest machine to the processing server. From this list, the latest file is selected. This file is then downloaded to the server for further processing. This is a timed process and varies depending on the requirements.

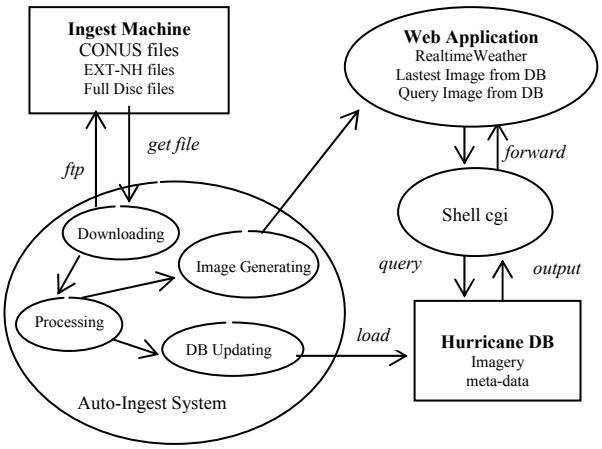


Figure 49. Auto-Ingest System Structure

(2) Data Processing: The GOES-8 ingest system broadcasts the data in GVAR (GOES Variable Retransmission) format. This defined format consists of twelve distinct blocks numbered 0 through 11 including 10-bit pixel data and the detector information arranged in correct order. [96] When the GVAR image is processed, the data is organized into five

bands and each of the five bands are output to a file in BSQ format for easier manipulation.

(3) Image Generating: The BSQ file is processed and an image is generated. This image is scaled down, converted into JPEG format and then put on the web page (in the latest data section). The web site is constantly updated with the latest weather image of the Eastern United States.

(4) Database Updating: The image is loaded into the hurricane database at set intervals. The method used for storing remote sensed digital imagery is directly related to the performance obtained at retrieval time. The decision of which method to use depends on the size of the image, the characteristics of the data and the related applications. There are several ways to store the data. For low resolution images, or images were algorithms will be performed, it is efficient to store the data by scanline. For high resolution images, or images where only a certain portion will be viewed, it is efficient to store the data by tiles. A combination of the two techniques can be used to store the data by swaths. For this system, we chose to store the data by scanline since this is a low-resolution image and the data attributes are relative to each scanline.

4.3.3.4. Web Interface

The web interface represents a gateway to our vast archive of meteorological imagery. It displays the latest weather image and facilitates the querying of archived GOES-8 data. The Web interface is divided into three frames, as shown in Figure 50.

The first is the “Latest image” frame. This frame displays the latest CONUS image acquired from our GOES ground station and processed by our ingest system. The “Latest image” is updated from the main automatic ingest system once every 30 minutes.

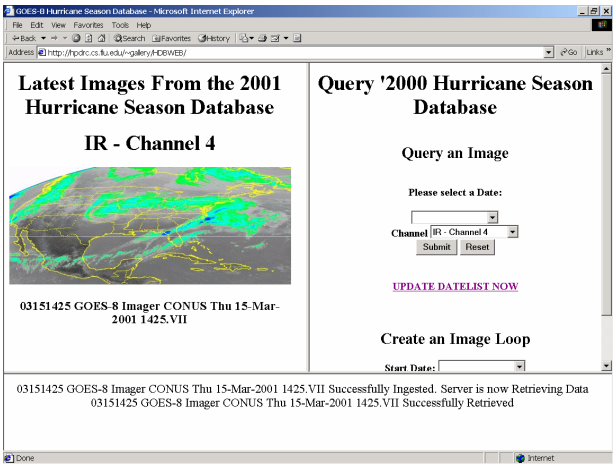


Figure 50. GOES-8 Web Interface

The second frame shows the Hurricane Database list. This frame provides links to available Hurricane Databases for the current as well as previous years. Following a database link provides access to the database query form. This form allows the user to query CONUS images of 12-hour intervals corresponding to specific dates and channels. In addition, image loops may also be created by specifying a loop start and end date. Image loops resemble the animations used during evening newscasts. They allow the observation of past atmospheric patterns and behaviors over a given time period.

The third frame is the console. It displays the server’s current status as well as the date and time associated with the last ingested image. Server error messages are also displayed in the console frame.

4.3.4. Results

The usage of remote sensed satellites, such as the GOES-8, has made major contributions and advancements to the study and monitoring of atmospheric and oceanographic phenomena as well as to the detection of forest fires and extreme changes in the environment. As the demand for meteorological data continues to grow, a specialized system that efficiently handles large amounts of detailed data produced by remote sensing devices is required.

The merger of Sem-ODB technology with Web resources has enabled the development of a reliable and efficient source of meteorological imagery and data. This system not only fulfills the need for fast data manipulation, but also facilitates access, regardless of possible limitations on connections speeds. Real time and historical data is then made available and can further be used in conjunction with other types of data or meteorological tools.

4.4. Everglades Data Integration using a Semantic Database System

4.4.1. Summary

The NASA Regional Applications Center at Florida International University's (FIU) High Performance Database Research Center (HPDRC) is involved to the acquisition, storage, and dissemination of all usable remotely-sensed and other data collected about the Everglades National Park (ENP) since the 1920's, as well as data collected for ground truth verification of current satellite observations. The purpose of this effort is to increase the use of remotely sensed information for monitoring the status of the ENP and the contiguous region. This section describes a prototype of an

environmental database that is being constructed to include all data related to the South Florida region. This data will be stored in a random-access semantic database for immediate use via the Internet. Methods already developed at HPDRC will perform the various data processing activities including data fusion, ingestion, compression, archived data retrieval, querying and mining, data publishing and dissemination, image rendering and visualization.

4.4.2. Background

There are several major reasons why the Everglades National Park attracts researchers involved in environmental studies. It not only has a unique and picturesque landscape, it is home to many endangered species. This area is highly affected by the human agricultural activities in the surrounding region and rapid urban development in South Florida faces problems such as supplying fresh water for the growing population. Thus, the Park indicates the environmental health of the whole area.

The overall project on constructing the Everglades Environmental Database (EEDB) is obviously a huge undertaking. This data came from different sources such as ground-based observations, aerial photography, satellite remote sensing together with demographic and cartographic information. In addition, the data is present in different textual and graphical formats. Even part of the project's goal of collecting the spatial data that is available for the ENP and the conservation area south of Lake Okeechobee is, by itself, a difficult task.

All the data should be integrated in such a way that a comprehensive database is available to researchers in a convenient, useful and timely manner. To solve the problem

of accurate storage and management of the data collected, we employ the semantic modeling approach to the design and implementation of the database. This technique has been developed at the HPDRC and has proven to be more efficient than the widely used relational approach for storing and retrieving large binary objects such as satellite images.

There are several aspects of systems research arising from this project aimed on extending the basic functionality of the database. Among the important goals are integrating of commercial Geographic Information Systems (GIS) with the EEDB and making the data available through the Internet.

4.4.3. Types of Data for the Database

For the last seventy years the government and scientific organizations have performed a large number of environmental studies within the boundaries of the ENP. These studies have resulted in the collection of massive amounts of valuable data from various sources. The information from the following resources is to be populated in the EEDB: historical data (i.e., data collected about the Park since 1920's), ENP aerial survey data, satellite data of the terrain for the last 20 years, and data collected for the ground truth verification of the current satellite observations.

HPDRC is now completing the design and implementation of a large relational database for the Everglades National Park. This task is accomplished using Oracle relational database management system (RDBMS). More precisely, the relational database is a set of 22 databases, which store the results of environmental surveys performed within the Park during the past several decades. Some of these historical data

sets will form an important part of the present semantic project and we now turn to a brief discussion of this data.

Data recording for a typical survey is performed as follows. The researchers have a set of printed forms, which they fill in with the data collected during fieldwork. In the office, the recorded information is entered into the computer and stored in the plain text files of a predefined format. This format is carefully described in the documentation. Nowadays the rapid development of computer technology allows using laptop computers during field observations for direct recording of measurements.

The files resulting from one survey logically form a data set. The largest data set stores the physical parameters of the area. These parameters include water depth, salinity, pH, temperature, wind direction and speed, air temperature, rainfall, conductivity, evaporation as well as others. A set of monitoring stations has been established within the Park to automatically record the physical data on a 10-minute, hourly or daily basis. All this data is populated into the Physical database. This data set is considered to be the most comprehensive and complete of the data sets available.

The other data sets resulted from vegetation and animal species studies. The Creel Census data set contains data on the fishing activities within the boundaries of the Park. Florida Bay, in particular, is an essential part of the Park and a vast region for exploration. The Shrimp data set focuses on the investigation of the life of invertebrates, but also contains data on marine vegetation observations, specific methods and stations. The aerial manatee surveys are found in the Manatee data set and a separate data set contains data on the study of Florida panthers' behavior. Several aerial surveys were conducted to study alligators' nests, wading birds and white tail deer location and

activities. These studies formed the Systematic Reconnaissance Flight data set. Smaller data sets contain information on Fire Ecology, Pine Demographic, East Everglades Exotic, Mitigation and other surveys.

Another source of information for the EEDB will arise from a joint effort between the Everglades National Park and the University of Georgia. This project is aimed at providing aerial photography of the Park region. This will make available a set of high-resolution images, which completely cover the Park's territory. These images are planned to be primarily used for geographic location, topographic and vegetation study purposes. The images can be scanned into the computer to be placed into the database.

Presently, satellite images are a valuable source of information that is used in decision making. Thus, this data should be populated in the semantic database. At present a low-volume satellite dish is installed at the FIU campus. It will be initially used to receive geostationary/geosynchronous operational environmental satellite (GOES) data for the EEDB. In the process of scientific exchange with other governmental and educational institutions, we are planning to receive images from other satellites. However some data still has to be acquired on the open market.

The variety of satellite images available is quite large depending on the characteristics of the corresponding remote sensing systems. The characteristics of these systems can vary in a number of ways. One way they can vary is in terms of spectral resolution. This refers to the number and dimension of specific wavelength intervals in the electromagnetic spectrum to which a sensor is sensitive. They can also vary in terms of spatial resolution. This is a measure of the smallest angular or linear separation between two objects that can be resolved by the sensor. Temporal resolution of a sensor

can also vary. This refers to how often it records imagery of a particular area. Finally, radiometric resolution can vary. This defines the sensitivity of a detector to differences in signal strength. To highlight specific features of the terrain, filters may be applied to the image. This usually results in a set of supplementary images to be stored in the database in addition to the original one.

Other source of information to be populated in the EEDB is ground-truth verification of current satellite observations. The goal of these investigations is to verify whether conclusions made about biophysical variables that are based on the study of satellite images are true or false. This data can be obtained by comparing data collected by biologists in the field with data from the satellite images.

In addition to environmental data, related social-economic, demographic, transportation, and health information can be stored in the present semantic database.

4.4.4. Semantic Modeling Approach to the Database Design

Widely used relational databases have provided good service in many conventional database applications. However, in situations where the structure of information is complex, greater flexibility is required (e.g., where objects with unknown identifiers or objects moving from one class to another are involved) or where non-conventional data is involved (long texts, images, etc.), other approaches such as semantic and/or object-oriented databases need to be considered.

The central notion of semantic database models is the concept of the object. An object is any real world entity that we wish to store information about in the database. The objects are categorized into classes according to their common properties. These

classes, called categories, need not be disjoint - that is, one object may belong to several of them. Further, an arbitrary structure of subcategories and supercategories can be defined.

The representation of the objects in the computer is invisible to the user, who perceives the objects as real-world entities. These entities can be tangible, such as persons or cars, or intangible, such as observations, meetings, or desires. The database is perceived by its user as a set of facts about objects. There are three types of facts: (1) facts stating that an object belongs to a category; (2) facts stating that there is a relationship between objects; and (3) facts relating objects to data such as numbers, texts, dates, images, and tabulated or analytical functions. Relationships in the database can be of arbitrary kinds. For example, stating that there is a many-to-many relation 'address' between the category of person and text means that one person may have one address, several addresses or no address at all. We refer to (Rishe, 1992) for a discussion of the semantic modeling approach.

To illustrate some basic principles of the design of a semantic database, one can look at a schema design of a simple semantic database such as the schema found in Figure 51. This schema is designed to store the results of the so-called East Everglades Exotic Plant Treatment Survey.

Eight categories are present in the schema diagram, represented by rectangles. Among them three categories, namely PROJECT, PERSONNEL and PROJECT-SPECIES-CODE, belong to other semantic schemas and are referenced here. The rest of the categories have attributes which are written inside the rectangles. Arrows show

relations between the categories. The direction of the arrow is from the domain to the range. The relations have also their own names.

The schema diagram preserves the logic of the Exotic Plant Treatment. One can read the schema to follow the logic of the survey. The main category of the schema is SITE-TREATMENT. It contains information on the treatment conditions of each particular site. Through the-site relation we are able to know the site-number, which uniquely identifies each site and its coordinates. By following the-herbicide relation we can find out which herbicide was used in the SITE-TREATMENT, the-hydrological-conditions relation link to corresponding information, and the-species relation tells which vegetation species were treated during the session. The treatment was done during the-excursion, which is uniquely defined by date and contains flight and labor information. In turn, the excursion (the category EAST-EVERGLADES-EXOTIC-PLANT-TRATMENT-EXCURSION) was conducted within the-project. Information on the Park employees who took part in the excursion and sites' treatment can be found by following the-crew relation.

A clear presentation of the database's business logic is one of the key advantages of the semantic approach to the design. A professional in any field is able to create a database design for his research without having to be an expert in the theory of databases.

4.4.5. Algorithms and Programs of Implementation

A set of programming tools that can be used to design and create the semantic database, populate the data, and make queries and access the database through the Web is being created at the HPDRC. This technology has been implemented within the present

project. In this section, we will briefly introduce the programs and libraries that are currently available.

A programming realization of the Semantic Database Modeling (SDBM) conception is the Semantic Database Interface (SDBI). The SDBI is a library of classes, functions and procedures, which are used for creating semantic database-oriented applications. Logically all the functions of the SDBI are divided into several groups. The first group of functions is responsible for creating, opening, and closing the semantic database. When the database is opened, the updates to the database are performed within separate transactions. The transaction mechanism checks the database integrity, resolves any conflicts resulting from updates by several users and performs the physical update of the database. Another group of functions supports creating and editing the database schema, allows the implementation of user data types and works with categories and relations. Once a logical schema is designed and the corresponding database is created, the data loading proceeds. As was mentioned in Section 4.4.4, loading refers to populating the database with facts about objects. A set of SDBI functions helps to complete the data loading process. Performing queries is a frequent operation while working with the database and several functions are implemented in the SDBI to support user queries.

Based on the SDBI, a set of applications has been created. These programs are aimed at automating the development process of the semantic database. The user is supposed to be concentrated on the semantic design. All the routine operations on creating the database are performed by the above mentioned applications. Some of these tools are described below.

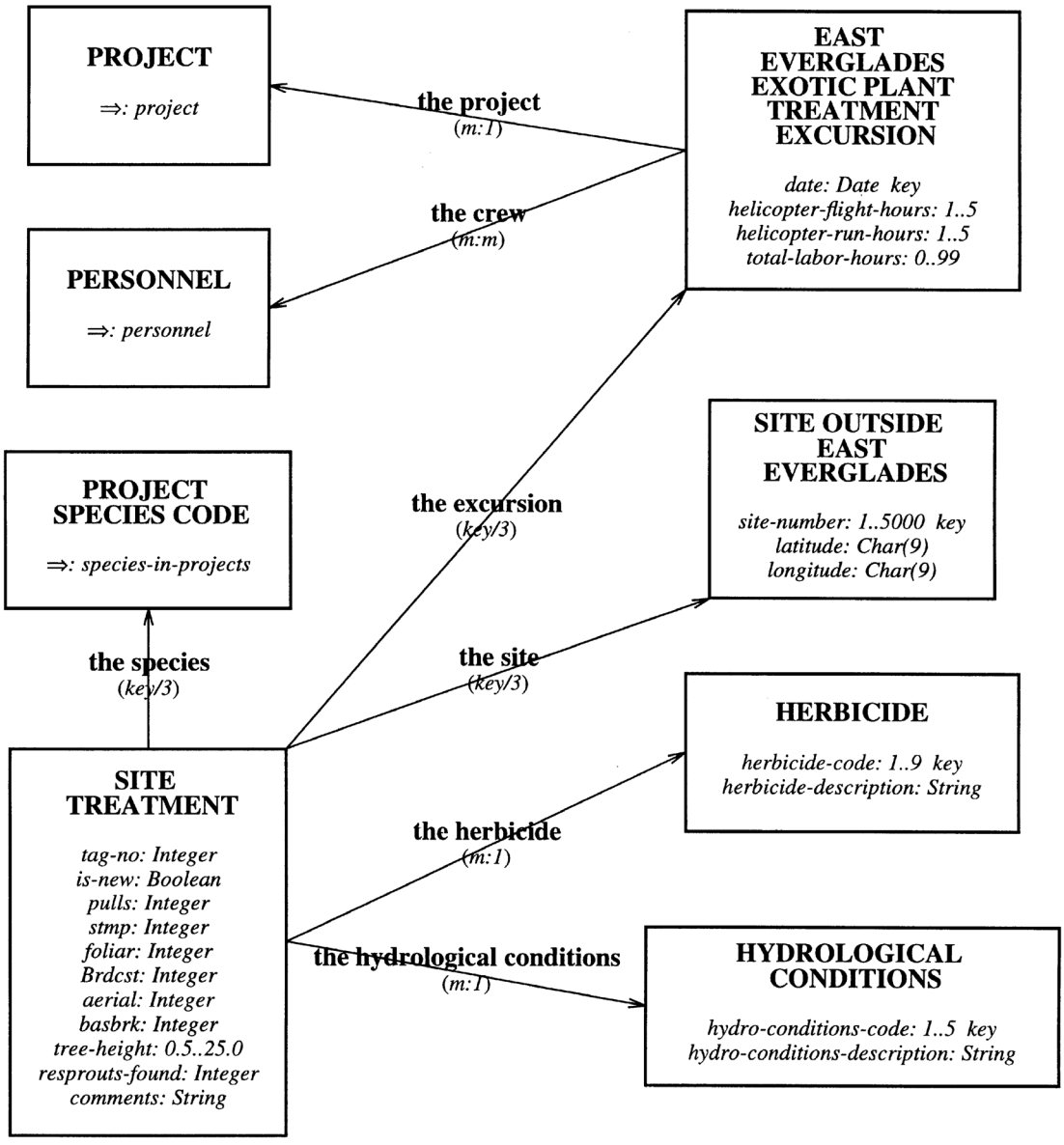


Figure 51. A Sub-schema For East Everglades Exotic Plant Treatment

One tool that is available is the Schema Editor. This is a Java application, which provides a user-friendly environment to create and modify the semantic schema design. By means of windows and dialogue boxes, a user is able to specify categories and relations, fill in comments, and use a mouse to locate categories and relations in the

design. The final schema design is saved to a file in a special Schema Design Language (SDL) format.

The Schema Documentation tool is used to prepare the SDL files for printing. This tool transforms an SDL file to a Rich Text Format (RTF) file. The file can then easily be printed from MS Word or any other package, which supports RTF. Figure 51 shows a sample printout.

To physically create the database, the Semantic Database Generator is used. This tool takes an SDL-file containing the database design and produces a Semantic Database (SDB) file.

The Semantic Forms tool is currently under development. It is a Java program, which analyzes the structure of the database and provides the user with a set of forms that can be used to enter the data. It ensures that all the data to be entered meets the integrity and other constraints of the database.

For making queries, we have adapted Structured Query Language (SQL), the standard relational database language, to semantic databases. The original purpose of this adaptation was to be compatible with, and to be able to communicate with, relational tools. The application of SQL to semantic databases allows utilization of the full semantics of the data, can be applied to scientific and spatial data, properly treats missing values and produces queries which are typically an order of magnitude shorter than if written in SQL for an equivalent normalized relational database.

Several Semantic SQL related tools have also been developed. The Open Database Connectivity (ODBC) SQL server allows interoperability with third party tools. Embedded SQL allows programming of complex applications. A plain SQL server allows

submission of ad-hoc and pre-programmed SQL queries on Windows NT and Unix machines. An internet/intranet World Wide Web (WWW) server allows submission of SQL queries via standard Internet browsers. An Internet SQL query generator helps the user to define a query without writing SQL statements. Finally, a graphical view selector allows the user to select a small view of the database against which to pose queries.

It is well-known that respond time of a user's query and the disk space necessary for storing data are the most critical characteristics for database development. In the present project we are planning to store satellite images in the database. These complex images cause the database to grow dramatically in size. At our site, we are conducting research to determine which image processing and data compression algorithms are most effective. Because the loss any data in compression techniques is undesirable for the satellite images, we are considering loss-less compression techniques. On the other hand, data compression slows down the respond time of the query. Further, queries performed remotely through the Internet prove to be very time-consuming. Thus, we are investigating and implementing various algorithms designed to optimize the image loading and displaying process from remote directories and speed up decompression.

We believe that remotely sensed data should not be analyzed in a vacuum without the benefit of other corresponding information. As a part of our project, integration of the EEDB with the GIS will be done. To achieve this goal we are developing the GIS interface for the EEDB.

Summarizing, the tools and algorithms, which have been developed, or are presently under development, will help create a fully functional prototype of the EEDB.

4.4.6. Application of the EEDB

It is expected that a wide range of users will be interested in the EEDB. One of the main goals of the current project is to make NASA satellite sensor readings available for the needs of South Florida. By being effectively stored and managed, these satellite images are becoming a valuable source of information. Researchers and specialists in biology, urban planning, road construction, water management, medicine, weather forecasting and many other areas can use the images to gain valuable information necessary in their profession. In addition, the textual part of the database will contain a description of the images as well as other alphanumeric information gained from previously conducted environmental studies. The integration of the EEDB with commercial GIS will make the database even more useful and attractive. Availability of the database through the Internet will significantly increase the number of potential users.

Initially, the scientists conducting the biological projects will use the data stored in the EEDB for making reports, calculating statistics and planning new environmental studies. They will also provide new data that will be stored in the database in the future.

The students and faculty at FIU will also use the EEDB. This database will encourage scientific research collaborations among FIU academic departments and will be used for information retrieval and library bibliographic instruction purposes. This will be particularly useful for the existing GIS campus group at FIU and will aid in meeting the increased demands for GIS research/teaching at FIU. The teachers at Dade county public schools will be using the EEDB for educational purposes. Students will be able to receive information for their school environmental projects by visiting the EEDB web site.

Basing on this experience, it will be possible to establish relations with government agencies and private commercial enterprises in South Florida interested in new sources of information. This effort will expand the practical application of NASA activities in exploring the Earth from space.

4.5. Cape Sable Seaside Sparrow Environmental Database

4.5.1. Summary

We discuss an environmental database related to studies of the Cape Sable Seaside Sparrow (*Ammodramus maritimus mirabilis*) performed in the Everglades National Park in 1981 and from 1992 to the present. Besides standard database development steps such as collection of requirements, conceptual and physical design of the database, data cleansing and loading, and establishing maintenance procedures much attention was devoted to the development of an application that allows biologists to query the database and produce reports. In particular, a visualization tool has been developed to allow scientists to study the distribution of sparrow population over several geographical areas within the Park. This database application is fully Internet enabled, giving biologists the freedom to more easily access data and collaborate with their colleagues.

4.5.2. Background

The Cape Sable Seaside Sparrow (*Ammodramus maritimus mirabilis*) is among a group of species first listed as endangered species by the U.S. Fish and Wildlife Service [97] on March 11, 1967. The sparrow has very limited distribution. Threats to its habitat

are posed by large-scale conversion of land in southern Florida to agricultural uses and construction.

Surveys of the sparrow began in 1974. Helicopters have been used to ferry observers to its remote locations. The first range-wide survey was performed in 1981. The survey was repeated in 1992, and range-wide surveys have continued in every subsequent year [98].

The scientists recognized the need to organize the datasets into one database for efficient use and data maintenance. Computer professionals and biologists from the Everglades National Park's South Florida Ecosystem Office [99] and researchers from FIU's High Performance Database Research Center [16] participated in this effort.

Besides data loading, storage, and maintenance, one of the requirements of the database project was to give scientists easy access to the data. Ideally, database access would not require knowledge of database query languages or the installation of additional software on the user's computer. A natural solution was to develop an Internet-based application that only requires biologists to have a standard Internet browser and Internet access. This has been accomplished through the design and implementation of a series of web forms. To specify a database query, the scientist chooses parameters by mouse clicks. The results are displayed in various formats and visualized as overlays on a digital map.

4.5.3. Sparrow Datasets

The surveys that have been conducted revealed that sparrows are found in a set of populations, which were marked in the studies by the letters A through G. The

populations are separated to various degrees by unsuitable vegetation. Ultimately, vegetation plays a crucial role for the sparrows' survival, acting as feeding and breeding grounds for these small birds.

Several natural and anthropogenic factors cause population changes for the Cape Sable Seaside Sparrow. Among natural phenomena that most affect the sparrows' habitat are hurricanes and floods. Fires caused by natural or human intervention also leave a devastating effect on the areas populated by sparrows.

Based on this information, several database concepts have been identified. In particular, the following data elements were determined:

- Vegetation codes and descriptions, vegetation coverage areas;
- Soil depth;
- Water depth.

Other database concepts are related to the conducted surveys, namely:

- Area quads — Orthophoto quadrangles maps of scale 1:24000 as defined by the USGS (United States Geological Survey, [100]);
- Sites — places visited by scientists during field surveys. Sites are geo-referenced in the UTM (universal transverse mercator) coordinate system, assuming UTM zone is 17 for Florida. Sites also fall into the areas of sparrow populations A–G;
- Observations are identified by the observation date. Observations are conducted on particular sites and are part of the survey;
- Sparrows — the only characteristic collected is bird count within each particular observation.

4.5.4. Database Design and Implementation

The semantic modeling approach was chosen for the conceptual database design. We refer to [8] for a thorough discussion on semantic modeling. The semantic description of the database, which is called the semantic schema, serves as a high-level documentation of the database.

For visual presentation and printouts, the semantic schema may be depicted as a diagram of the database concepts, called categories, attributes of the categories, and relationships between the categories. A category is displayed on the schema as a box with the category name in uppercase bold letters. Attributes appear inside the corresponding category boxes in italic letters. Relations between the categories are denoted by arrows. The names of the relations appear in bold letters. Cardinality of the relations as well as additional constraint information is placed on the semantic schema in italics. Figure 52 presents the semantic schema for the Cape Sable Seaside Sparrow database.

Physical design and implementation have been done using the relational database model. The Oracle RDBMS [101] (relational database management system) was chosen to host the Sparrow database. The database was created by a set of SQL (structured query language) data definition instructions. The data was verified and cleaned before being bulk loaded into a set of tables. The database integrity constraints were enforced prior to data loading.

Establishing maintenance procedures for the Sparrow database was influenced by the requirement of regular bulk loads of fresh data from ongoing surveys. Besides identification of the database backup schedule, a set of UNIX shell scripts was written to

automate the process of data loading. Data verification is assumed to be performed by the RDBMS through database integrity constraints checks.

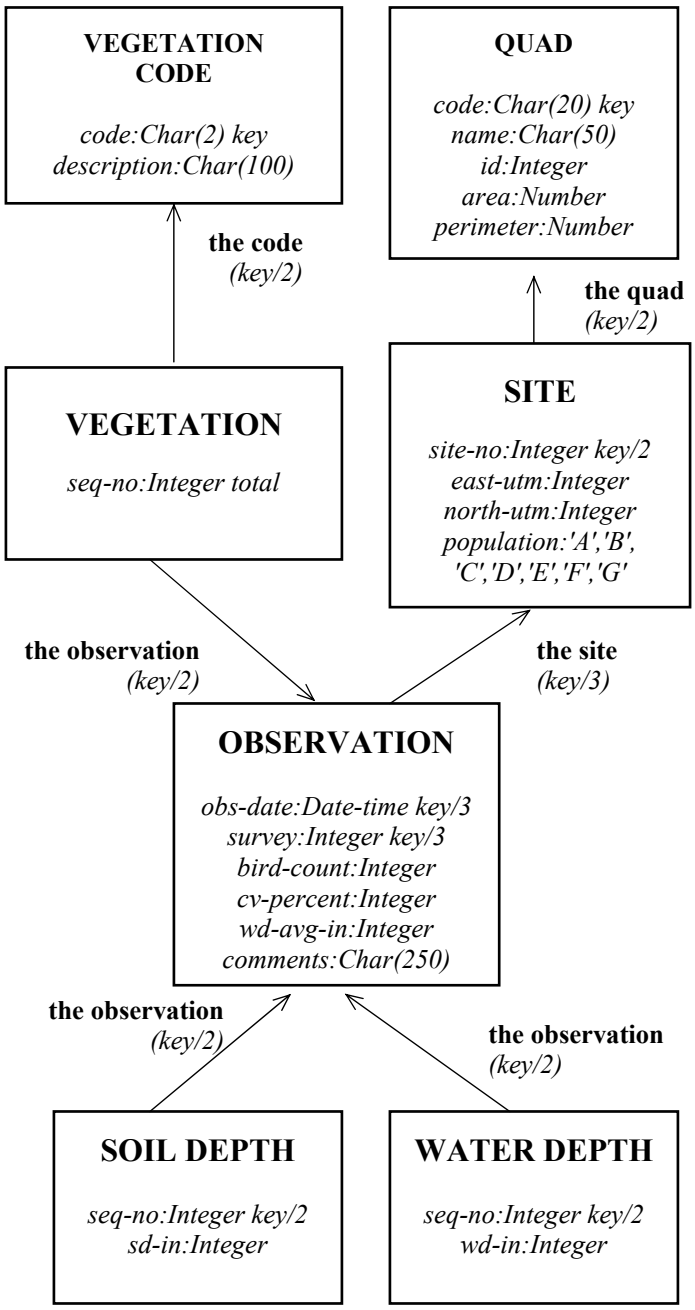


Figure 52. Semantic schema for Cape Sable Seaside Sparrow Database

4.5.5. Data Access and Visualization via the Internet

An Internet-based application [102] was developed for the scientists to submit queries to and receive results from the Sparrow database. The application allows users to work in two modes:

- Download Form;
- Interactive Map.

Download Form is primarily designed to retrieve data for further analysis by third-party tools. The query form supports several parameters:

- Data type, having values 'Bird', 'Water', 'Vegetation', 'Soil';
- Time period of interest — start and end year;
- Type of survey — original or supplemental;
- Sample parameters, like Average Water Depth, Bird Count, Vegetation Coverage.
- The query form allows the user to choose an aggregation level for the data output:
- Sparrow population by POR (period of record);
- Sparrow population by year;
- Observation site by POR.

The user may define the geographical area of her interest by specifying either a set of quads and sites or by choosing areas of sparrow populations. In both cases, an interactive map with an overlay of observation sites (or population areas) can be used to choose geographical areas. For example, the interactive map for sites shows each particular site as a small colored square. Moving the mouse over the square initiates a hint displaying the quad name and site number. A click on the square displays a photo of

the corresponding site, if available, and an invitation to add the site to the list of geographical areas to be used in the query.

Query results can be received in two ways: displaying data on the screen in HTML (hypertext markup language) tables or saving the resultant dataset as plain text in comma-separated format for download. The latter option is useful when the result set is expected to be relatively large, thus making it inconvenient to browse in HTML, in cases where processing of the query will take a considerable amount of time, and when the results will need to be imported to third party software tool for further processing and analysis. There are options to choose on the query form that allow the result set to be returned and saved with the images and metadata. The result set can also be compressed for faster download. The user enters her e-mail address and gets notification from the system when the result dataset is ready as well as instructions on how to download it from the server. Figure 53 shows the query options form for database access.

Working in Interactive Map mode is different. The Interactive Map is primarily designed to allow visual analysis of data. The user is invited to enter a range of years (start year – end year) and type of survey (original or supplemental). Clicking the "GO" button displays an interactive map with an overlay of bird observation data from the corresponding survey conducted on the start year from a chosen time frame. The scientist may further superimpose layers of vegetation, water depth, or soil depth data. The layers are presented with the colored squares and the legend is available to explain correspondence between colors and ranges of values. When the user clicks her mouse on a square corresponding to a particular observation, a window containing the following information related to the observation appears:

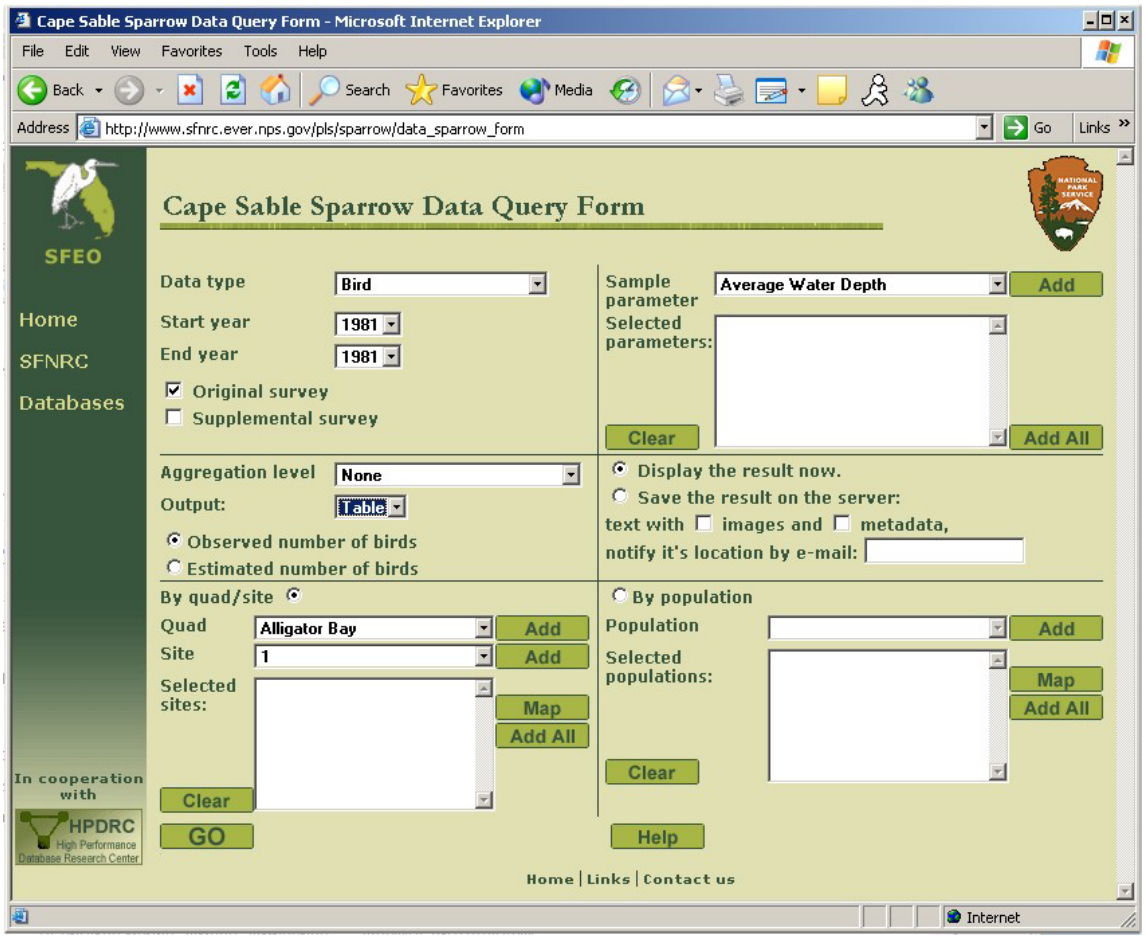


Figure 53. Query options form for Cape Sable Seaside Sparrow database

- Quad name;
- Site number;
- Birds count;
- Average water depth;
- Average soil depth;
- Percentage of the observation site covered with vegetation;
- Dominant vegetation species.

This window also has an additional option to “Show Site's History.” Choosing this option displays a separate HTML window with all the data available for a particular

observation site and all of the years observations were conducted on the site. Additional buttons allow the user to save this data as a plain text file, as a compressed text file (with images and metadata, if chosen), to manipulate data (sorting, matrix report), and to get statistics about the data via a special SPSS [103] analytical module.

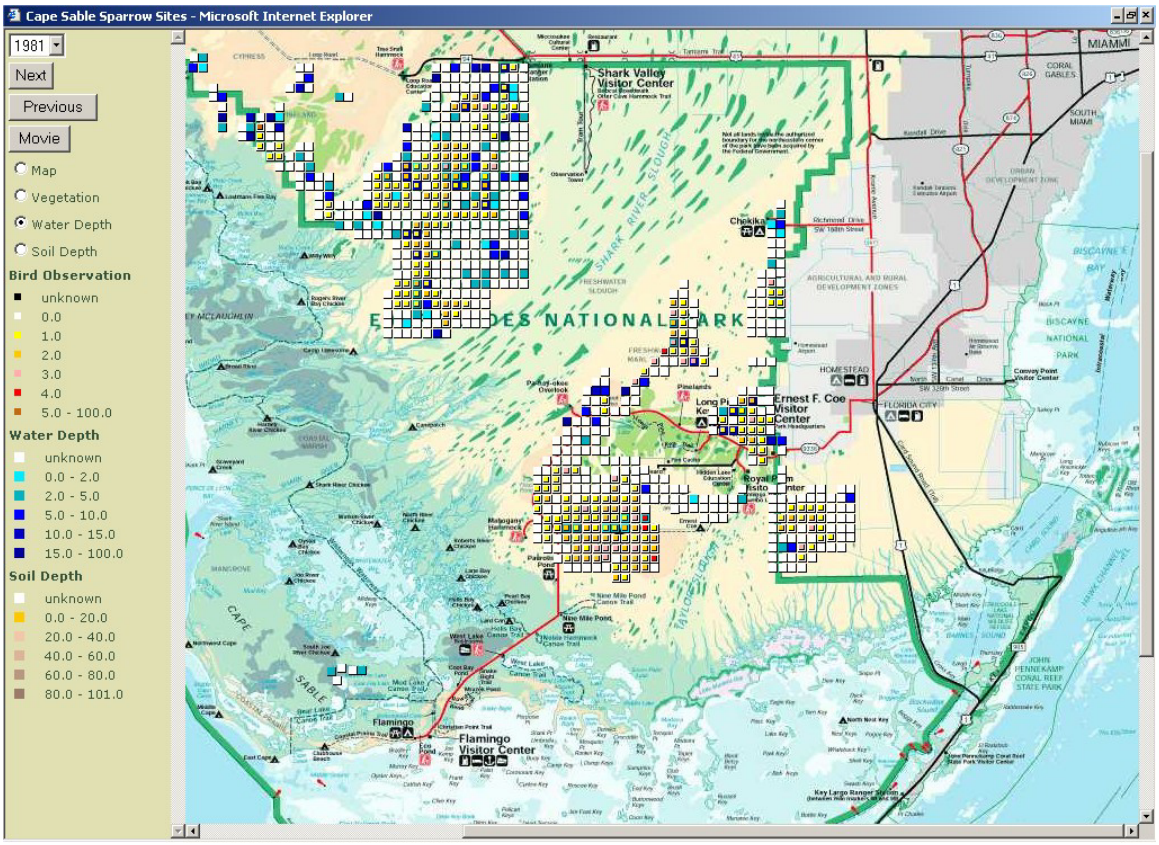


Figure 54. Interactive map with bird count and water depth data superimposed over the map for Cape Sable Seaside Sparrow database

The Interactive Map has one more important visualization tool. If the "Movie" button is clicked, the application displays a sequence of layers superimposed over the map as a movie; each frame of the movie corresponds to the observations done in a particular year. This feature allows the user to visualize the dynamics of changes in

sparrow habitat sites and populations over a series of years. Figure 54 depicts an interactive map with bird count and water depth data superimposed over the map.

4.5.6. Results

The Cape Sable Seaside Sparrow environmental database is now fully functional. It serves the primary goal to collect together all the data related to this endangered species into one repository for efficient storage, maintenance, and use in scientific research. Since the surveys continue on a regular basis, new data is expected to be inserted into the sparrow database.

The Internet-based application to provide user access to sparrow database has been launched. The application has rich visualization capabilities and data query support. It gives scientists greater freedom and flexibility to access the sparrow database.

5. TerraFly

This chapter describes research in support of TerraFly, a web-based system that allows for the dissemination of geospatial information, as well as applications of TerraFly.

The first item presents the method used by TerraFly to integrate heterogeneous of geospatial information from sites distributed across the Web. It was published as:

> N. Rishe, S.C. Chen, A. Mendoza, A. Selivonenko, O. Dyganova, S. Graham. "GIS Inter-Protocol Bridge: GIS Vector and Raster Imagery Inter-Process Wrapper" Seventh International Conference on Remote Sensing for Marine and Coastal Environments. pp. F5.1-F5.7.

The next two items explore the hardware architecture used to manage the TerraFly system, which serves approximately forty terabytes of data to an average of

10,000 unique users per day, and a scheme for replicating portions of the system at remote sites and maintaining TerraFly sites. These items were published as:

> N. Rishe, A. Selivonenko, M. Chekmasov, S. Graham, M. Gutierrez, B. Wongsaroj, O. Dyganova, A. Mendoza. "Overview of TerraFly GIS Hardware Architecture." 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2003). Vol. XVI, pp. 11-16.
>
> N. Rishe, B. Wongsaroj, M. Chekmasov, A. Selivonenko, S. Graham, M. Gutierrez, O. Dyganova, W. Pinckney, E. Padron, C. Armstrong, A. Marshall. "Replication and Maintenance of TerraFly GIS." 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2003). Vol. XVI, pp. 6-10.

TerraFly allows its users to select a point on the displayed imagery; the system then returns information relevant to that point to the user. A paper describing the methods used to provide this functionality was published as:

> N. Rishe, M. Chekmasov, M. Chekmasova, S. Graham, I. De Felipe. "On-demand Geo-referenced TerraFly Data Miner." 2003 International Conference on Intelligent User Interfaces (IUI 2003). pp. 277-279.

Content designers can program scripts that allow TerraFly to be run in a mode that controls the flight path and zoom level for the user; this can be used to show specific features of the dataset. A paper describing the design and use of Autopilot scripting was published as:

> N. Rishe, B. Wongsaroj, M. Chekmasov, A. Selivonenko, Y. Sun, S. Graham, A. Mendoza. "Autopilot Scripting for the TerraFly GIS." Ninth International Conference on Distributed Multimedia Systems (DMS 2003). pp. 98-100.

TerraFly has been used to create custom applications for several domains. The ability to overlay vector and raster data onto TerraFly's aerial photography base map has been combined with the ability to present user-specific information about a given point to facilitate a study of the vulnerability of mobile home communities. The ability to overlay data has been combined with the ability to specify a flight path to design an application

that will allow a TerraFly application to be created for flight planning. Papers describing these applications have been published as:

> M. Gutierrez, N. Rishe, O. Dyganova, A. Selivonenko, G. Rocha, S. Graham, R. Alvarez. "Applying TerraFly for Vulnerability Assessment of Mobile Home Communities" Seventh International Conference on Remote Sensing for Marine and Coastal Environments. pp. F6.1-F6.6.
>
> N. Rishe, B. Wongsaroj, M. Chekasov, S. Graham, E. Johnson, R. Cain, C. Weekes, D. Lawrence. "TerraFly Application for Flight Planning and Training." 7th World Multiconference on Systemics, Cybernetics and Informatics (SCI-2003). Vol. XVI, pp. 1-5.

5.1. GIS Inter-Protocol Bridge: GIS Vector and Raster Imagery Inter-Process Wrapper

5.1.1. Summary

The GIS inter-protocol wrapper is an image processing system that dynamically retrieves and merges remote sensing data obtained from the web's data sources. Processing of spatial imagery such as satellite, aerial, and topographic maps of the earth can be accessed online via Internet/Intranet protocols. The service and communication framework can be loaded into the client side via a web browser. The browser's Java run time environment obtains and hosts the components needed to execute the image processing services. The application gives the end-user the ability to transform the semantics of metadata to the desired imagery format in various resolutions, encoding processes such as JPEG or BMP, and ease of distribution through various protocols and/or computer media. Integrating the services with Internet browsers provides intuitive spatial and text interface to the data. Users need no special hardware, software, or

knowledge to locate and browse imagery. Any PC, MAC, or UNIX workstation with a Java enabled web browser can access the system.

5.1.2. Background

The World Wide Web is an enormous repository and a popular medium for interacting with information. Web based Geographical Information Systems (GIS) are becoming an active area of research in the GIS community. Web-based GIS can make data retrieval and the analysis of GIS data available to users worldwide. Many GIS companies have become web-focused increasing the amount of spatial information that is available on line. The web is inherently a graphical environment and images of neighborhoods are recognizable and interesting to many around the world. Comprehensive information on and easy access to huge amounts of heterogeneous data are important prerequisites for GIS. The value of GIS stems from their ability to integrate great quantities of information about the environment and to provide a powerful repertoire of analytical tools to explore this data.

5.1.3. System Architecture

The GIS bridge is a multi-threaded Java server providing access to a high number of simultaneous users via standard web browsers. The browser's capability to interpret java class byte code allows the server to execute natively on the client's computer. This framework allows embedding of complex java components on demand into the client-side. Internet capabilities allow the system to access numerous data sets without the need for any special GIS software. [104]

The GIS inter-protocol wrapper bridge consists of the three-tier architecture, depicted in Figure 55.

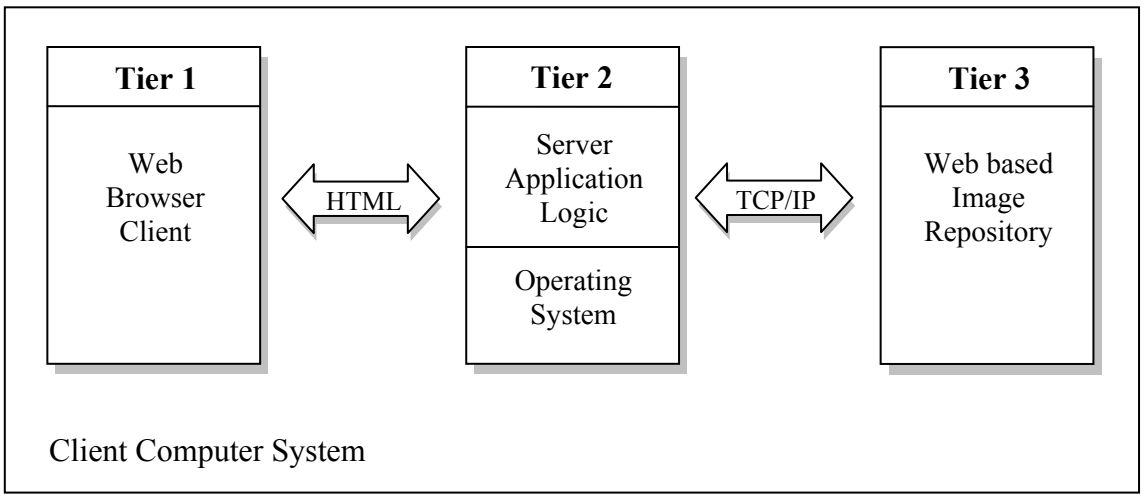


Figure 55. System Architecture

- Tier 1: The *Client* is a graphical web browser or other hardware/software system that supports HTTP protocols. The application was built and tested with Netscape Navigator and Internet Explorer on Windows and UNIX platforms.
- Tier 2: The *Application Logic* is a web server application that responds to HTTP requests submitted by clients by interacting with the Tier 3 *Web Data Source* and applying logic to the results returned.
- Tier 3: The *Web Data Source* data base system contains all the imagery and meta-data required by the Application Logic tier.

5.1.4. General Approach

The application is a wrapper that translates queries and meta-data from one data model to another from different heterogeneous data sources; its general logic is presented in Figure 56. This functionality is provided around each individual meta-data repository.

The wrapper converts queries into one or more queries understandable by the underlying data repository and transforms the result into a format understood by the client. The application provides a simplified interface to encapsulate diverse data sources so that they all present a common interface. [105, 106]

The application provides the client with the ability to dynamically retrieve and merge remote sensing data. Internet connectivity allows the system to access numerous data sets without the need of any special GIS software. Designed for users of all levels and unlike other geographic system, the inner-wrapper runs via standard Web browsers, with no need to download software or data. The system is a Java applet that executes within the runtime environment of the browser once the user reaches the website. The browser's java enabled environment relieves the application from problems relating to platform dependency, thus providing a greater flexibility within multiple environments. The execution process begins by checking that the client's browser contains the latest version of the application. If it does not, the web site provides the browser with a compressed distribution package containing the java byte code needed in order to execute the server services. The site prompts the user with a security certificate informing them of additional security privileges required. The application needs to be given security privileges that are normally restricted by the applet "sandbox" security model such as: network resources to communicate with the web data sources and access to the clients' local disk in order to retrieve and save images. If the user does not accept the security certificate the application cannot establish its services and the process is aborted. The system also checks the client's browser setting. The browser's setting should be set to

allow the execution of Java applets. If the java virtual machine is disabled, the system will prompt the user to enable it.

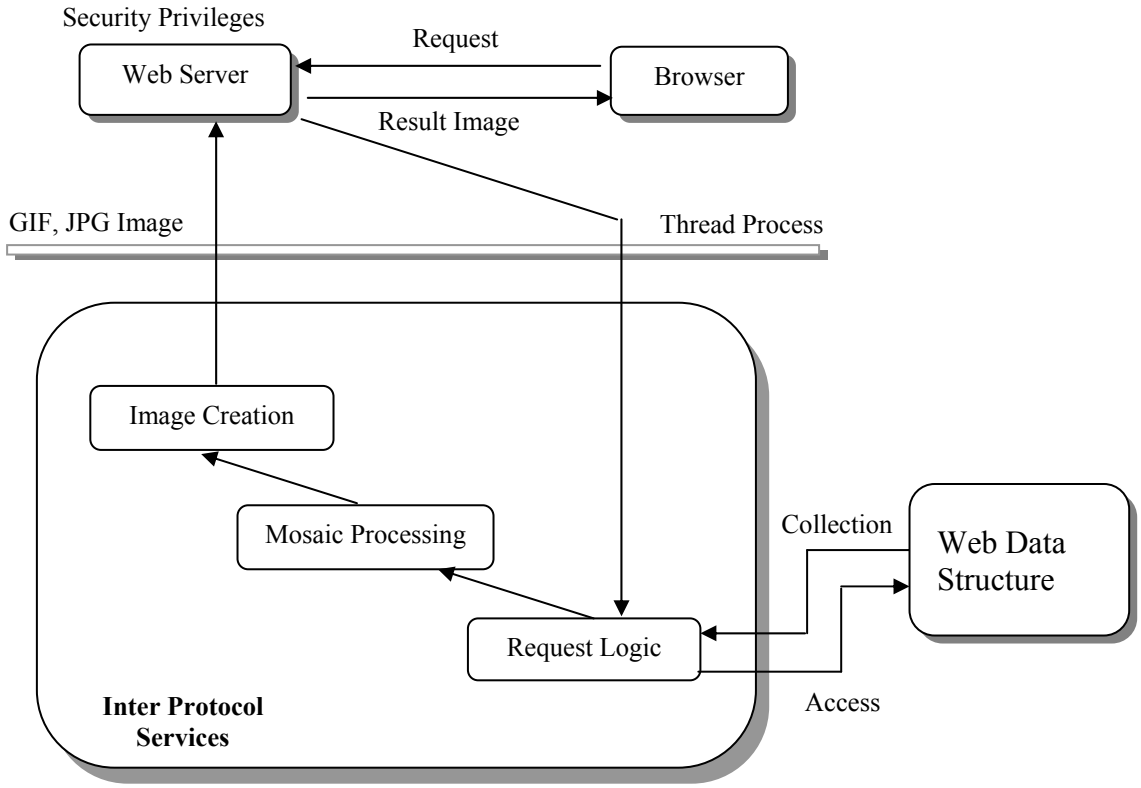


Figure 56. System Logic

When the security privileges are granted, the site starts dispensing the java byte code package to the client's browser. Once the dispensing process has been completed the client is informed that the application needs additional information to optimize the available services. The application prompts the user to provide the port number where the server will be executing and the name of the proxy if one is required. Asking the user for a port provides the user more leeway in setting up the server. If binding of the port entered by the user fails, the application will prompt the user for another by suggesting the next available port from a scan of ports automatically performed by the application. The application is capable of scanning ports on the client's computer until it finds one

that is available and informing the user of the port scan outcome. Once the startup procedure has completed, the application is ready to accept incoming connections.

The application is a multi-threaded java server running within the environment of the web browser. The application is a wrapper capable of dispensing images from repositories on the Internet. The application can interact directly with the user through the browser interface or can programmatically interact with other applications requiring image processing. When the server receives a request it creates a thread-process for each incoming request. This thread handles the imaging requirements of the client. The thread gets a handle of the port the client is mapped to and establishes a communication channel with the client. Once the communication channel has been established the client can communicate via HTTP protocol and request the needed services. At the same time, the server continues to intercept any incoming requests and creates an individual thread per request handle. The HTTP request contains commands that the application logic understands in order to initiate the services. There are several commands required in order for the image services to proceed. The server provides several services and each service requires a fixed number of parameters. Though the order of the parameters does not matter, the number of parameters must be correct in order for the services to begin. Once the application has understood the request, the dispensing process begins. The client specifies the coordinates, imagery type, and dimensions. The imagery is based on the Universal Transverse Mercator (UTM) projection using the North American Datum (NAD) ellipsoid. UTM is a projection system that divides the earth into 60 wedges shaped zones number 1 thru 60 beginning with the International Date Line. Each zone is 6 degrees wide and goes from the equator to the poles. UTM grid coordinates are

specified as zone number, then meters from the equator and from the zone meridian (UTM grid units can also be in inches, feet, meters or kilometers). The system supports several resolutions depending on the data being utilized. The system can modify available imagery of power of 2 from 1/1024 meter per pixel through 16384 meters per pixel range. The user specifies the coordinates for the data that needs to be retrieved using UTM coordinates and can select from satellite imagery, maps, and aerial photography.

The application's data fetching logic contacts the data repositories' web site to attain the needed imagery. The application fetches images from repositories that dispense their images in small sizes such as 200X200 pixels. These sizes are ideal since they contain a small amount of data and their transfer rate over the network is fairly fast even when high-speed connections are not available. The latency of this size of image and the bandwidth requirements do not hinder system resources on the client's computer. The application provides the user with the ability to create images of any size. These images are created from smaller images or tiles. Once the user specifies the size of the image, the application logic begins to build the image by mosaic the number of images required to build the one requested by the client. Since the source images are small the application spawns multiples threads whose jobs are to retrieve each individual tile. This process allows the application to request a large number of images at the same time and concurrently wait for them to be received into the client's computer. Since these images are obtained from the web there are problems that can arise from the data source provider. The application must be able to track the images being retrieved and also guarantee that the images obtained have all the correct data. The application has a

monitoring mechanism that allows it to track and check each individual image. If the image requested did not transfer correctly, it resends the request to the data provider requesting the image to be fetched. If the data context of the image is not corrupted, the application temporarily buffers the image in order to mosaic it with the other incoming images. The application also has a fail check mechanism that allows it to deal with any unusual delays encounter through the fetching process. The application waits a fixed amount of time for the request to finalize. Since the fetching of images has variable time requirements the application must be able to handle the wide variety of time discrepancies encounter throughout the process. If the data provider does not deliver the data within this time the application resends the request several times. If no response is received, the application creates a black tile for the particular missing image. This informs the user visually of the fact that an error was encounter during the transmission and that the outcome of the requested image is not correctly formatted.

When the request logic determines that the incoming images have passed all of the preventive measures, they are then transferred to the image assembly module. This modules creates a mosaic from the dynamically retrieved imagery acquired from the web's data repositories. During image acquisition the system aggregates the images and process the images to obtain the final Ad-hoc image request. The construction of the images requires an efficient memory management algorithm to dispense multiple images concurrently. Therefore, memory optimization is of high priority. The web browser-hosting environment natively provides a limited amount of memory to applications. The application must be efficient in the use of memory; otherwise the maximum size of the image it can produce is limited to the physical memory available on the client's

computer. The application keeps track of the images being used and once the images are no longer needed the resources are given back to the system. When the tiling of the images is concluded and the required image has been produced the application then needs to convert the image to one of the encoding format it provides such as JPEG or BMP.

The encoding module takes the newly created native java image object and applies one of the encoding methods provided by the application. The user chooses the encoding format and the application dynamically loads the require image encoder. All the encoders provided by the application have a handle to the outputstream of the client's socket. This stream is use to deliver the final compressed image. The output stream can be directed to the client's local hard drive if the client whishes to do so or it can be redirected to a common TCP/IP port. During the compression phase the application can concurrently deliver the data to the client even though the image has not been totally compressed. The application buffers a small portion of data and applies the compression to it. This approach allows the clients to start receiving the data stream quickly without having to wait for the entire image to be compressed.

The system might need to transfer extremely large amount of data, which may cause significant network delay. To address this issue, progressive transmission of spatial data over the web has been applied. This is particularly useful when the client is using a slow network connection or when the datasets are very large. The system can provide data in raster datasets. Subsampling of the pixels of the image is sent to the client so the client can start working without having to wait for the whole image to be downloaded. The full version of the original images is progressively treated by adding new pixels to the transmitted image on the client side.

Once the entire image has been fully compressed, the application releases all the system resources and closes and terminates the open socket. Terminating the socket to the client guarantees the client will not continue to wait unnecessarily for any more data to be delivered. Once all the dispensing process has been fully completed, the thread handling the request is terminated. The termination of the thread releases all the resources allocated to the system. At the same time the application can continue executing other incoming requests. The application keeps an open socket to the client while the distribution process is in progress. Once all the housekeeping has been executed the application closes the socket. The client's socket is kept open and delivery of data continues until the full image has been delivered.

5.1.5. Security Issues

At first glance, embedding of Java components into a browser seems straightforward and simple. However, if the components to be embedded violate the Java "sandbox" permission model, additional settings in the web browser's environment allows the applet to acquire the necessary system resources. Along with the use of an applet, come issues relating to security and the digital signing of the applet to make a secure I/O operation.

The permission model for trust-based security supports a useful set of parameterized and non-parameterized permissions that can be individually granted or denied for a particular zone. The Internet Explorer security model can be configured to three predefined permission sets, called High, Medium, and Low. These represent the most restrictive to the least restrictive set of permission, respectively.

The system is deployed as an applet to attain purely distributed functionality. As such the system faces many permissions restrictions that need to be granted according to Internet Explorer and Netscape security model. The standard Java sandbox permissions are as follows:

- Thread access in the current execution context.
- Network connections to the applet host.
- Creation of top-level windows with warning banners.
- Reflection to classes from the same loader.
- Access to base system properties.

The system is a multi-threaded application that makes remote network connections with not only the applet host but other computers as well. Therefore, the system violates the first two sets of permissions that correspond to the Java sandbox structure. To circumvent these security constraints the distribution and packaging mechanism contains digital signatures requesting the needed permissions in order for the application to function correctly.

5.2. Overview of TerraFly GIS Hardware Architecture

5.2.1. Background

TerraFly is an Internet-based technology that allows users to virtually “fly” over remotely sensed imagery and maps. Unlike conventional Geographic Information Systems (GIS), which often require installation of additional software on the client's side, TerraFly runs via a standard Internet browser. TerraFly uses a very large dataset of satellite imagery and aerial photography. To complement the imagery, it also feeds

information to users from a set of databases and directly from the Internet. These and other features are reflected in a distributed hardware architecture adopted for the TerraFly system. This paper provides an overview of the hardware architecture of the TerraFly GIS. The architecture takes into account the need to add new databases and imagery sources to TerraFly. It allows the use of different hardware modules and software platforms in the system. The architecture permits an efficient security protection scheme for TerraFly. Integration of new servers and services to TerraFly is straightforward, and maintenance and technical support of the whole TerraFly computer farm is efficient.

5.2.2. Summary

TerraFly [107] is an Internet-based technology that makes it possible for users to "fly" over vast land areas using only an ordinary Web browser. By employing high-resolution satellite imagery and aerial photography collected by the USGS (United States Geological Survey, [100]) and other sources, users can experience an overhead view of almost any location in the United States at a one-meter resolution without the expense of purchasing all of this data or installing GIS application software. TerraFly has an extensive set of features, which is continuously growing. Besides virtual flight and visual analysis of the imagery and overlays, data mining tools enrich the user's capability to understand the data. To accommodate development and inclusion of these tools and features into TerraFly, the following hardware architecture has been adopted for TerraFly's computer farm. The computers are clustered into several server groups (clusters), namely:

- file servers;

- applications servers;
- web servers;
- cache servers;

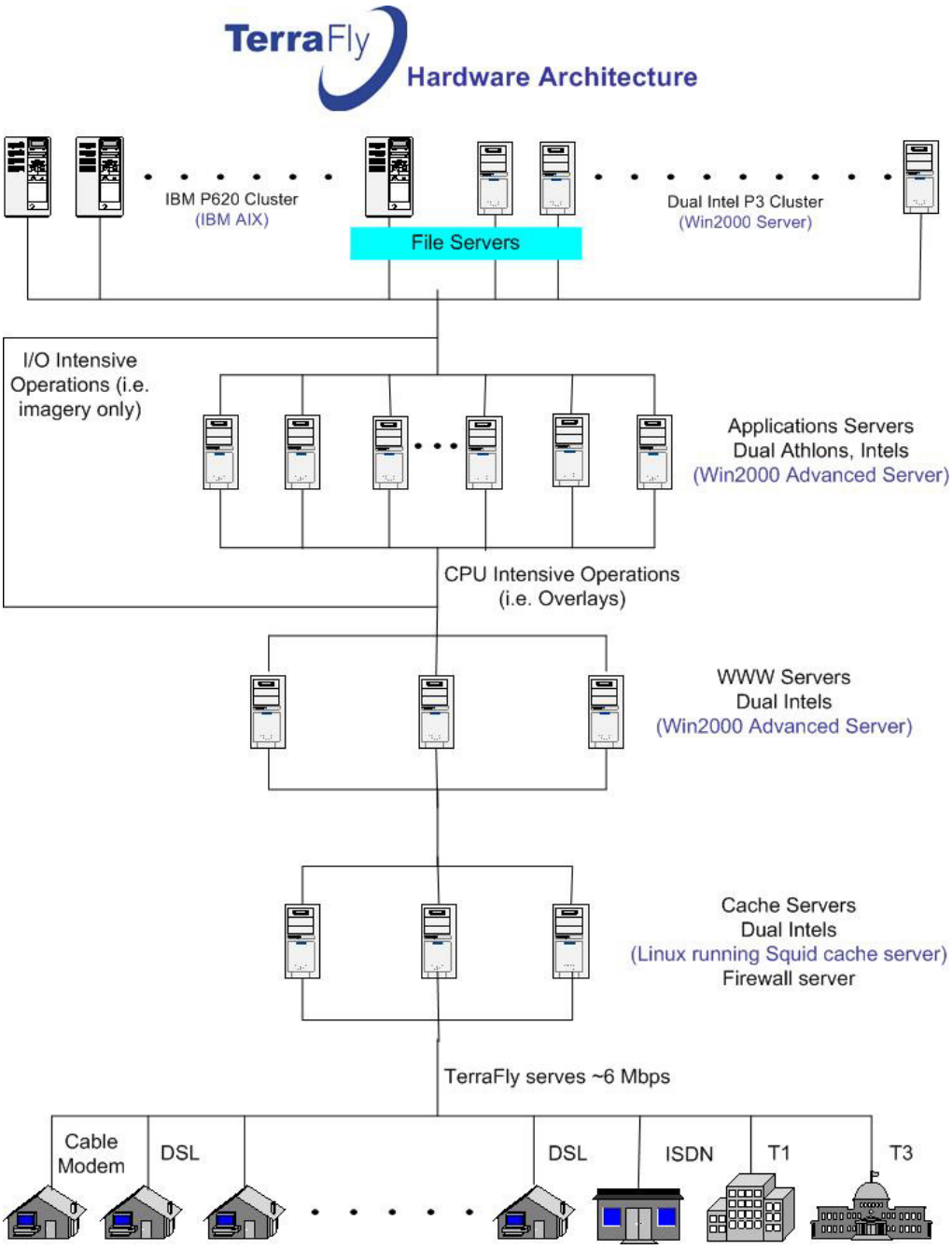


Figure 57. Hardware Architecture for the TerraFly System

- firewall server.

Each group of servers may accommodate computers of different hardware configuration. Various operating systems may be installed on the servers of a single group. This approach gives us the freedom to add new servers to a particular group as they are needed and to install the most suitable operating system on each server. There are, however, internal TerraFly applications that would require several servers to be homogenous in both software and hardware. This is particularly true for the applications servers group. In the following sections we discuss TerraFly's server groups, interfaces between the groups, and maintenance of the computer farm.

5.2.3. TerraFly Server Groups

We will now discuss TerraFly's server groups and the functionality of the system the computers are supporting. Figure 57 shows the general scheme of the TerraFly system's hardware architecture.

File servers are primarily configured to store large volumes of satellite imagery and aerial photography data from different data sources and at various resolutions. The dataset provided to us by the USGS exceeds 20 Terabytes and is constantly growing due to regular updates. This means that scalability is essential for this server group. Another important aspect to consider are intensive I/O (input/output) operations which the file servers sustain due to numerous imagery data requests.

Our current file servers group consists of two computer clusters. One cluster is composed of IBM machines, p620 series, running the AIX operating system, [108]. The second cluster is composed of Intel-based machines [109], dual Pentium III series,

running Microsoft Windows 2000 Server operating system, [110]. The primary characteristic of the file servers group is a large amount of hard drives attached to each server. With hardware configuration for file servers, the goal is to reach maximum storage capacity by employing the least possible number of computers while not compromising the reliability of the servers. The file servers support the core functionality of the TerraFly system:

- Smooth flight over spatial data: incremental tiles streamed to a Java applet are constantly retrieved from the file servers, allowing users to virtually "fly" over imagery or maps.
- Compass Control: allows 360 degree control of flight direction and speed. This feature also controls the data refresh rate. These parameters affect the workload of file servers retrieving the data.
- Go-to Coordinate and Place: allows the user to instantly get to the area of specific interest. This feature usually switches workload from one file server to another.
- Choose Data Type, Zoom In/Zoom Out: allows the user to view spatial data from various data sources at varying resolutions. File servers store imagery of several types (satellite imagery, aerial photography, topographic maps), prepared in various formats (for example, geo-tiff or jpeg), generated in different resolutions (1 foot per pixel, 1 meter per pixel, etc.), coming from different data providers (for example, USGS or Space Imaging [111]). Moreover, the imagery of the same area might have been acquired several times in recent years. These characteristics and the volumes of particular datasets have to be taken into consideration while distributing the workload between file servers.

- Autopilot: pre-programmed virtual flight over a particular area with pre-defined changes of direction, resolution, and flight speed gets its data feed from the file servers.

The second group of TerraFly computers is the applications servers group. These machines are used for CPU intensive operations. The group complements imagery provided by the file servers with the extensive imagery overlay data supported by TerraFly as well as further data mining related to the imagery's particular geographical location. Like the file servers group, this is a large pool of computers which comprises three clusters. The first cluster is responsible for generating information overlays to be superimposed over the imagery, the second cluster provides data mining services, and the third cluster supports mailing address conversion to/from geographical coordinates as well as calculation of geographical objects' proximity relative to a particular geographical point. The first cluster is composed of AMD Athlon-based machines [112], running Microsoft Windows 2000 Advanced Server operating system, the second and third clusters are Intel-based machines running Microsoft Windows 2000 Server.

The applications servers support the following functionality of TerraFly:

- Information Overlay: the applications servers allow users to overlay data from different sources (such as street names, hotels icons, real estate for sale) on top of the imagery and digital maps. Some overlays are generated on the fly, while others are pre-generated due to intensive calculations. Although the overlays are mainly composed of simple objects like points, lines and polygons, the combination of several overlays onto the imagery is a CPU time-consuming

process, since the layers should not compromise the imagery or each other when displayed.

- On-click data mining tool: some machines are database servers and file systems that contain additional data related to geographical areas. Among these data are United States Census 2000, hotels, schools, churches, businesses, places of interests and some other geo-referenced data objects.
- TerraFly data dispenser: the user is allowed to mark and dispense areas in TerraFly and then download data covering the selected area. The data objects to be presented to the user are calculated on the fly by application servers belonging to the third cluster.

The TerraFly web servers group gets all the requests to the system from the Internet, sends the TerraFly Java applet to the user's side, processes user requests for imagery and data, and return the results to the user. The group has Intel-based machines of Pentium III series running Microsoft Windows 2000 Advanced Server.

TerraFly also has a group of cache servers, which help to improve the workload of the whole TerraFly system and to shorten the response time for user requests. This group of machines caches the imagery and overlays that have been prepared and delivered to the user. When the same imagery is requested for the second time, it is taken from the cache server instead of being processed by the file and applications servers. The cache servers usually store imagery over densely populated areas such as U.S. metropolitan areas. TerraFly use statistics shows that many users prefer to take virtual flights over those areas. The group consists of Intel-based machines of Pentium III series running Linux operating system [113] and the Squid cache server [114].

The firewall server stands between the Internet and the TerraFly system. It filters valid user requests to TerraFly, while rejecting invalid and suspicious requests. Its major function is to prevent malicious attacks and non-sanctioned intrusions to the TerraFly system. The firewall server is an Intel-based machine of Pentium III series running Linux.

5.2.4. Server Farm Maintenance

The TerraFly server farm is organized into a separate computer sub-network. A dedicated network switch and a set of network routers for computer clusters operate this sub-network. Historically, several applications servers are located outside the TerraFly sub-network, namely the ones responsible for data mining and data dispensing services. They form a separate cluster, which is accessed through a dedicated Intel-based server, running the FreeBSD [115] operating system. This computer is responsible for security protection of the applications servers and the distribution of data requests between them as well as storing several static datasets and caching frequently requested data. Established interfaces between the TerraFly sub-network and the separate computer cluster as well as security procedures in place allow this architecture scheme to work in a reliable manner.

The TerraFly server farm is monitored by the Spong systems and network monitoring package, [116]. Spong is a simple system that has proven to be reliable for TerraFly's needs. It has these important features, among others:

- monitoring is client based. CPU, disks, processes, and logs are monitored;

- monitoring of network services like HTTP (hypertext transfer protocol) is provided;
- grouping of hosts like routers and servers is supported, making it easier to monitor the TerraFly server clusters as separate groups;
- adequate messaging on the problems is provided;
- history of problems is kept.

Figure 58 shows the Host Groups screen of Spong monitoring the TerraFly server farm.

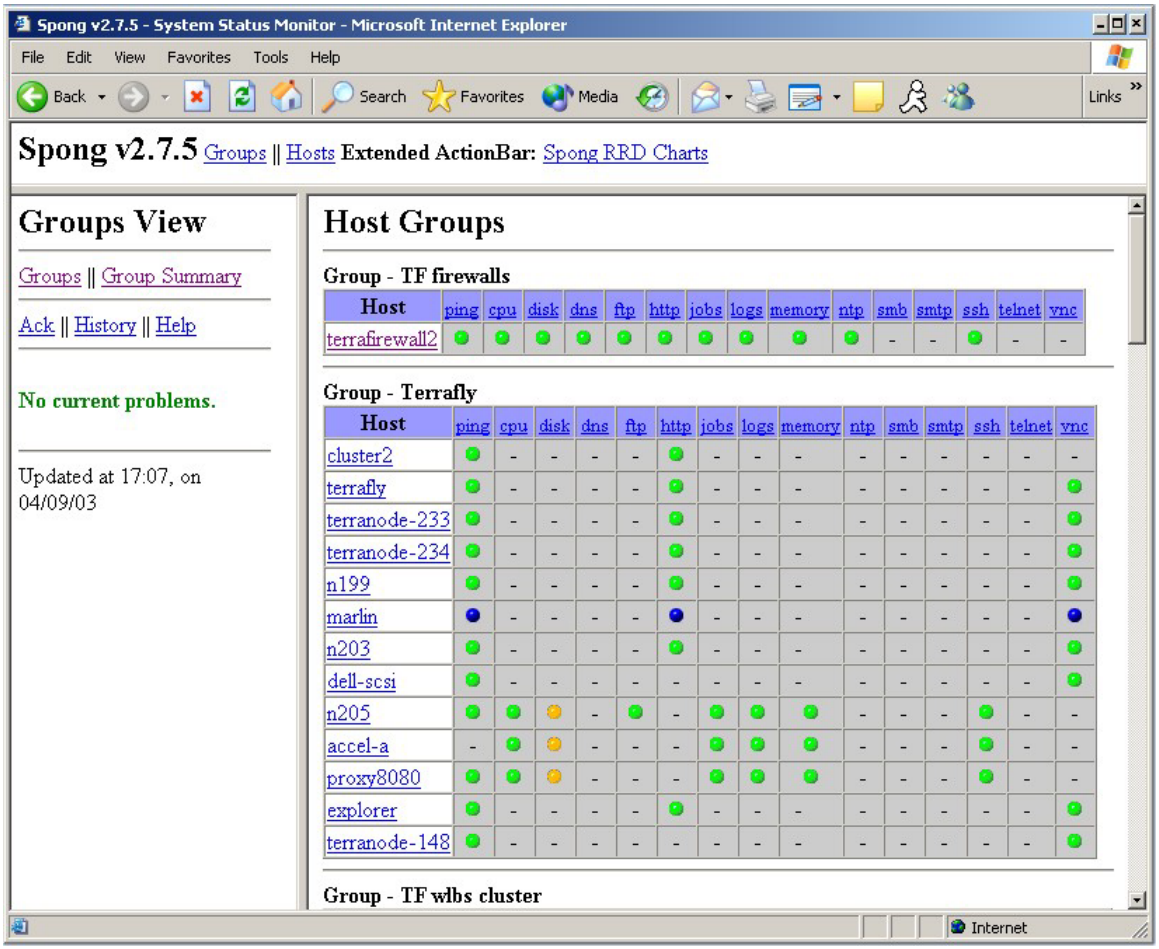


Figure 58. Host Groups Screen of the Spong Monitoring for TerraFly

Among other maintenance tasks, the following problems require daily attention of TerraFly staff:

- Hard drive failure. With hundreds of hard drives employed to store terabytes of data, the chances for hard drive failure are high. TerraFly employs RAID 5 (Redundant Array of Independent (or Inexpensive) Disks, level 5) to provide data striping at the byte level and to stripe error correction information. This results in good performance and fault tolerance.
- Power surge protection. A set of battery backups [117] is used to protect the servers from power outages.
- Protecting servers from overheating. Depending on computer configuration the necessary number of cooling fans is supplied to each machine. The servers are distributed in several physical locations to ensure adequate air conditioning.

The protection of stored data against corruption in case of infrastructure support failure is an important maintenance task for large data warehouses such as TerraFly. In cases of transitory unavailability of some servers carrying out some services, the software redundancy mechanism employed via Windows 2000 Advanced Server failover could sustain overall operation at the cost of somewhat reduced performance.

The TerraFly application consists of multiple software services, which are allocated on the overall field of service in such a way that any particular service employs servers located in more than one physical location. This ensures that each TerraFly service has high chance of survival when some server facilities become totally unavailable because of network, electrical power, or air conditioning outages, floods, or

other infrastructure failures. Taking into consideration the geographically dispersed allocation of TerraFly servers, a total outage is highly unlikely.

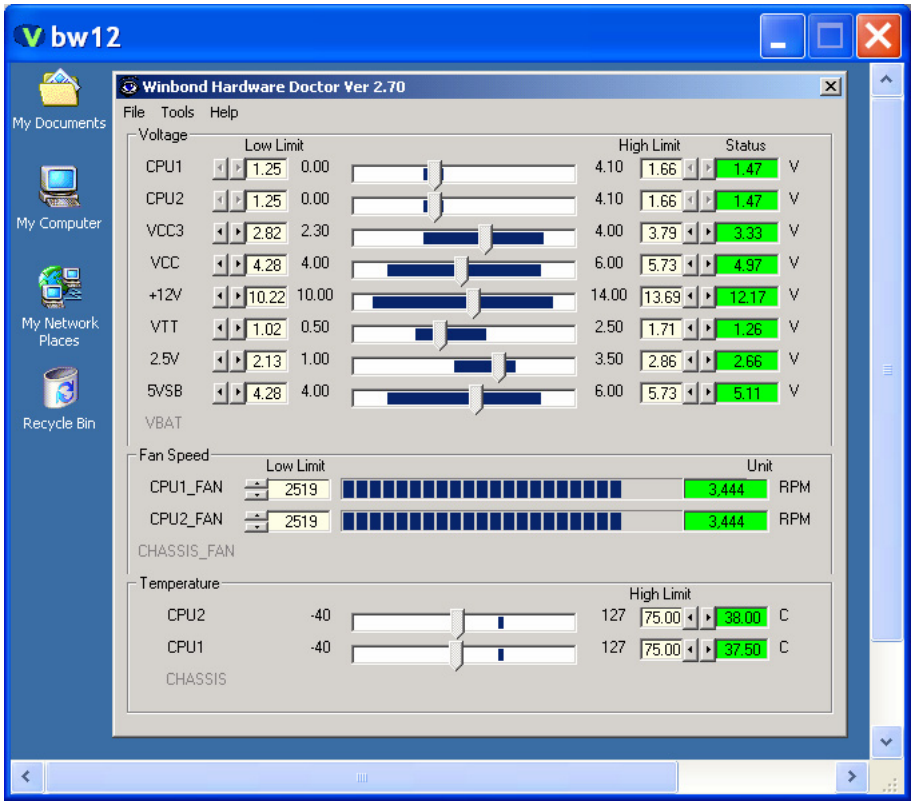


Figure 59. Winbond monitoring a TerraFly Server

Air conditioning outages represent one of the most dangerous conditions to the integrity of TerraFly's server farm. Such outages are frequent in our University environment. They do not affect hardware immediately, but if the outage continues for a significant time damage to hardware and a loss of data integrity occur. The resulting damage typically requires hardware replacement and database repopulation. To avoid such costly consequences, we employ a system to monitor each server's micro-climate– the Intel-recommended Winbond Hardware Doctor [118]. As shown in Figure 59, Winbond allows us to monitor various physical parameters of CPU and motherboard

operations: voltage of various DC points, fans speed, and CPU temperature. The latter is especially interesting to us. We have established that the normal temperature of CPU operations within our server facilities is between 30 and 45 degrees Celsius, with an average operating temperature of 38 degrees. When server facility ambient temperature rises, CPU temperature rises accordingly. By monitoring CPU temperature we can predict that a particular server is operating in undesirable temperature conditions. Winbond allows us to set margins of allowable parameters; upon exceeding these parameters, previously programmed actions are launched. Programming servers to shut down upon reaching a critical CPU temperature can prevent damage to the server's CPU and storage components.

Once a server is shut down, its circuits do not produce additional heat and could survive quite high ambient air temperatures without damage. Since there is no heat production within the server facility, once all the servers are shut down ambient air temperature does not rise beyond the temperature of outdoor air and thus stays within safe levels.

Unfortunately, software like Winbond does not allow auto-start of servers once temperature conditions return to tolerable levels. Since the server is shut down, there is no opportunity to monitor the server by its own means anymore so re-starts require human intervention. The TerraFly server center is considering the employment of specialized server management hardware that could diagnose dangerous environmental changes, make hardware surviving actions and resume normal operation when environmental conditions permit them.

5.3. Replication and Maintenance of TerraFly GIS

5.3.1. Summary

TerraFly is a fully web-enabled GIS (geographic information system) that makes a multi-terabyte collection of satellite imagery and aerial photography available and allows its users to virtually "fly" over the continental United States. While TerraFly is currently a large complex centralized system, a new set of applications dictates the necessity to replicate the TerraFly GIS to be part of third-party information systems. These replicas are to be housed at a separate physical location, will typically be characterized by a smaller set of customized imagery and information datasets, and could be maintained at the replication site or remotely. In this section, we describe the replication process and outline the maintenance procedures for TerraFly replicas. We discuss packaging TerraFly on DVD (digital video disk) and the installation of TerraFly at Florida Memorial College as examples of the replication process.

5.3.2. Background

TerraFly [107] is an interactive software tool for visualization of remotely sensed and spatial data. is TerraFly allows users to experience virtual "flight" over imagery or maps by streaming incremental imagery tiles to a Java applet. This virtual flight can be performed at various speeds, in all the compass directions, and at various altitudes (resolutions of the imagery) within the continental United States. Several data overlays are supported over the imagery include GNIS (Geographic Names Information System [119]) objects, hotels, real estate listings for sale, fires, property parcels, zip code contours, and other data that have associated geographic coordinates. Extensive textual

data related to geographical locations is delivered by the data mining tool, [120]. All the data services are provided via the Internet, giving TerraFly a much broader audience than conventional geographic information systems.

The above-mentioned functionality suits the information needs of public users related to local geography, demography, quality of life, economics, and the environment. However, our experience shows that professionals, researchers, and educators require TerraFly to be customized for use in their everyday activities. Among typical customization requests are

- TerraFly support for a relatively small geographic area with recently acquired high-resolution proprietary imagery (for example, to assist water management in the area);
- temporal imagery support: the same geographic area over a period of time (urban construction and infrastructure development);
- custom data overlays (environmental studies, education);
- extra security and restricted access to sensitive imagery and datasets (homeland security, disaster management and mitigation).

To meet these requirements, a full or partial replication of the TerraFly system is often needed. While each replication process will have specific requirements related to a particular TerraFly application, establishing general replication and maintenance procedures is essential. In the following sections, we discuss these issues and provide an example of a TerraFly replica.

5.3.3. Replication Process

The first step in the replication process is identifying the need for a full or partial replication, depending on the users' fields of study. By full TerraFly replica, we refer to a copy of TerraFly system customized for particular user applications and physically installed and operated at the user's site. In this case, no interaction between the main TerraFly installation and the replica is maintained. Partial replication is the case where interaction is constantly maintained between the TerraFly system and the replica. The typical example is when the TerraFly system hosts imagery for the replica. In this case, the replica is a semi-dependent application using the TerraFly system as a major data source.

The following are the major steps in the replication process:

- Collecting user requirements for the TerraFly replica. This includes identifying whether the replica should be a standalone or an Intranet or Internet application, the potential number of concurrent users, and the availability of imagery and data layers;
- Identifying hardware and network needs for the replica (processor speed for TerraFly replica server(s), RAM, network cards, a farm of servers versus one server);
- Resolving software issues (operating system, web server, firewall, antivirus software, necessary drivers);
- Establishing the TerraFly replica as part of organization's network (administrative privileges to the replica server(s), subnet organization within the network if necessary, routers/switches dedicated to TerraFly replica);

- design of backup/recovery (in particular, the decision of whether RAID (Redundant Array of Inexpensive Disks) system should be in place).

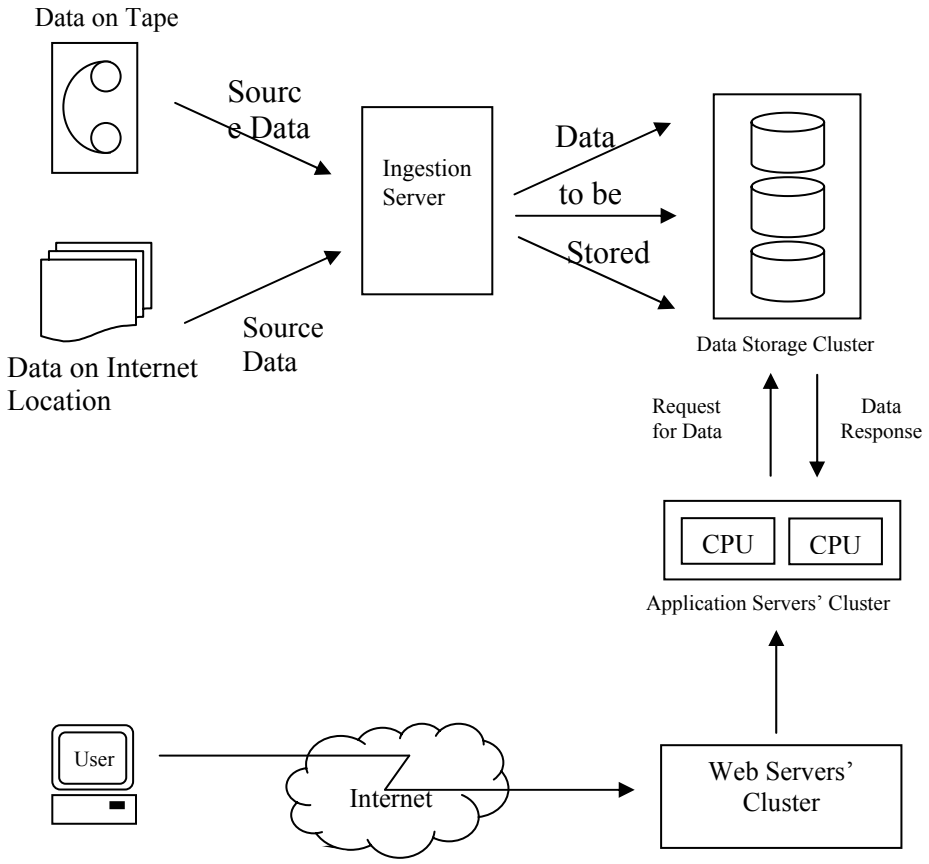


Figure 60. General components of TerraFly replica

Figure 60 shows the major components of a typical full TerraFly replica designed to work over the Internet and its data flow diagram. In this case, the main functional components are:

- data ingestion server;
- data storage cluster;
- applications' servers cluster;
- web servers' cluster.

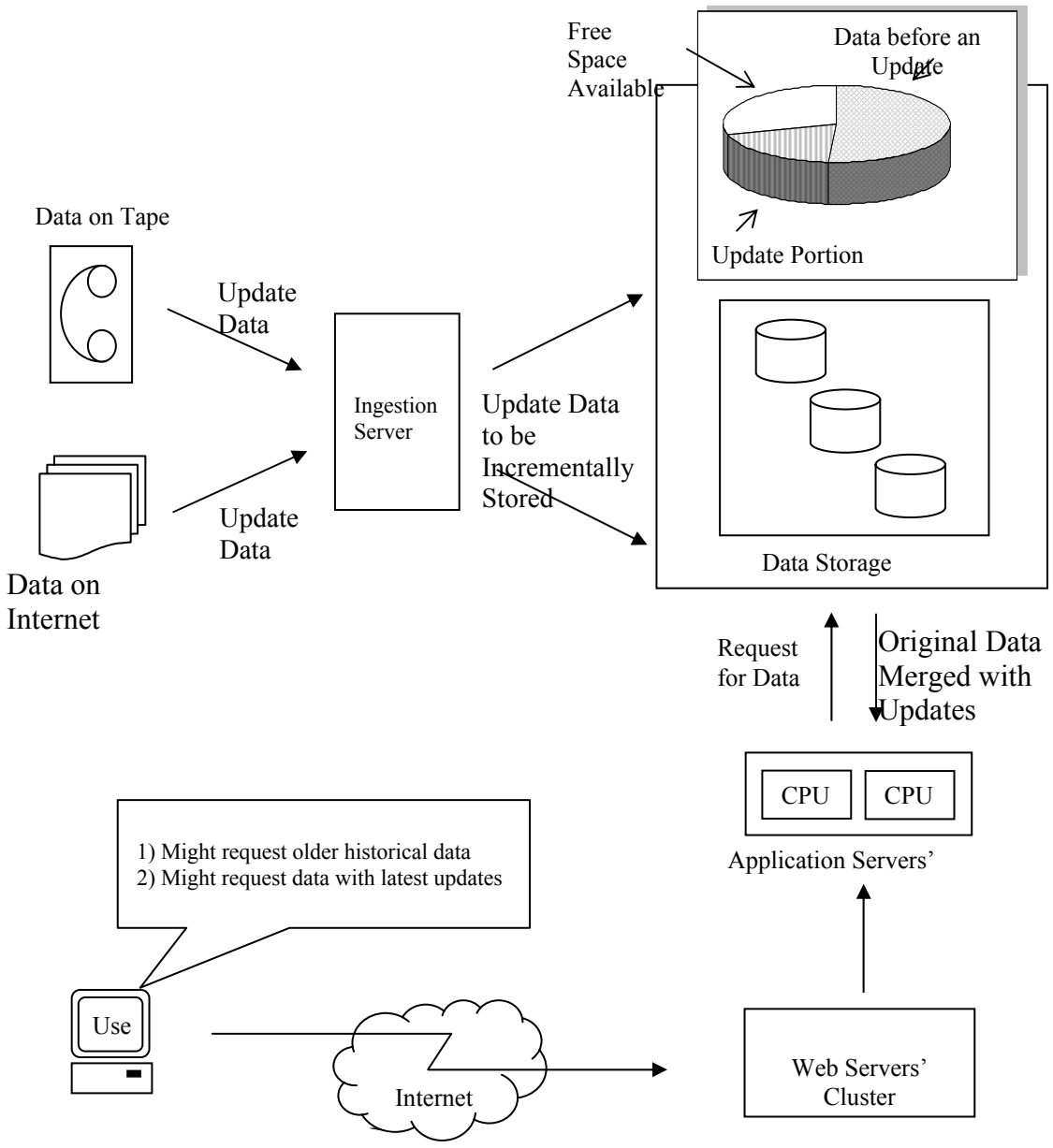


Figure 61. Incremental Data Updates in TerraFly Replica

Depending on the data volume and other parameters of the user applications, all the functional components may reside on one computer. However, it is highly recommended to dedicate separate machines for these services. The data to be stored in the TerraFly replica usually comes from storage devices such as tapes or through the network (the

Internet in particular). This data comes to the ingestion server for initial processing. The ingestion server moves processed data to the data storage cluster according to some pre-programmed storage strategy and naming conventions. The user accesses the TerraFly replica via the Internet. The Web servers' cluster provides the user interface for interaction with the system. All the data requests generated by the users' activity are passed to the application servers' cluster. The application server generates the data requests, sends them to the data storage cluster, receives the result, and further processes the output (say, by adding additional overlays on top of the imagery) before finally sending the result to the user.

Figure 61 illustrates the process of incremental data updates in a TerraFly replica. This functionality is important if the user application needs to add temporal dimension to data analysis. Unlike Figure 60, where data updates typically replace the previous dataset, here a separate method is invoked by the ingestion server to support incremental data storage and labeling (versioning) of the historical datasets. Application servers have enriched functionality to support historical data retrieval and amelioration. Web servers provide the user with the interface to request older historical data along with the latest updates.

5.3.4. Maintenance of TerraFly Replica

Maintenance of TerraFly replica requires the following general tasks to be performed:

- Hardware maintenance. This depends on the workload on the TerraFly replica servers and the volume of data stored. Adequate cooling of servers (air

conditioning of the server room as a minimum measure), as well as power surge protection are essential. Our experience shows that hard drive failure is a common problem when large volumes of imagery are stored on multiple drives.

- Operating system maintenance for the servers. A clear procedure should be in place to regularly install security patches, upgrades of antivirus databases, and other software.
- Update of the imagery. Depending on the nature of user applications, adding new imagery data or replacing old images with new images has to be scheduled.
- Similarly, updating information overlays for the TerraFly replica has to be scheduled. Unlike imagery, the information overlays are more frequently updated. Ideally, the update process should be fully automated.
- Backup and recovery procedures have to be in place.
- Maintenance of user access to TerraFly replica is essential. As part of the procedure, analysis of access logs has to be performed to trace unauthorized attempts to send requests to the system.

If TerraFly replica is integrated as part of the user's information system, additional measures related to its synchronization with other software modules has to be taken care of.

5.3.5. TerraFly on Digital Video Disk

TerraFly might be replicated on DVD. This could be particularly useful when the imagery dataset is relatively small (up to 4GB of data) and a network connection for the user is unavailable or slow. TerraFly replicated onto DVD has limited functionality when

compared to a normal replica. With DVD replicas, data updates are usually achieved by burning the next version of the DVD. Figure 62 demonstrates the typical use of a TerraFly DVD replica. In this case, the TerraFly GUI (graphical user interface) is pre-installed on the user's computer. The DVD contains datasets (imagery plus textual data) and TerraFly functionality such as generation of the imagery overlays. In other words, the TerraFly GUI plays the role of the Web servers used in a conventional TerraFly replica, data stored on DVD emulates a data storage cluster, and TerraFly functions on DVD emulate the applications server cluster. Data ingestion server functionality is omitted for DVD version since data updates are not considered.

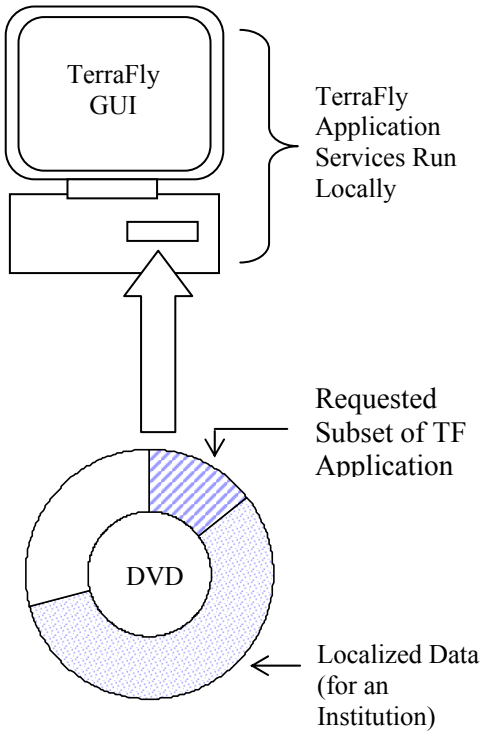


Figure 62. Use of TerraFly DVD replica

Figure 63 illustrates the process of creating a DVD replica. The requested subset of data is identified on the TerraFly storage cluster. Since the data is most likely stored on

several storage servers, it is first copied onto one drive prior to DVD creation. A subset of application services is identified on the TerraFly applications server cluster. Due to the modular architecture of the TerraFly software and adopted standard interfaces between the programs, the identified software modules can be recompiled without modifications of the program code. Both data and programs are then burned onto a blank DVD.

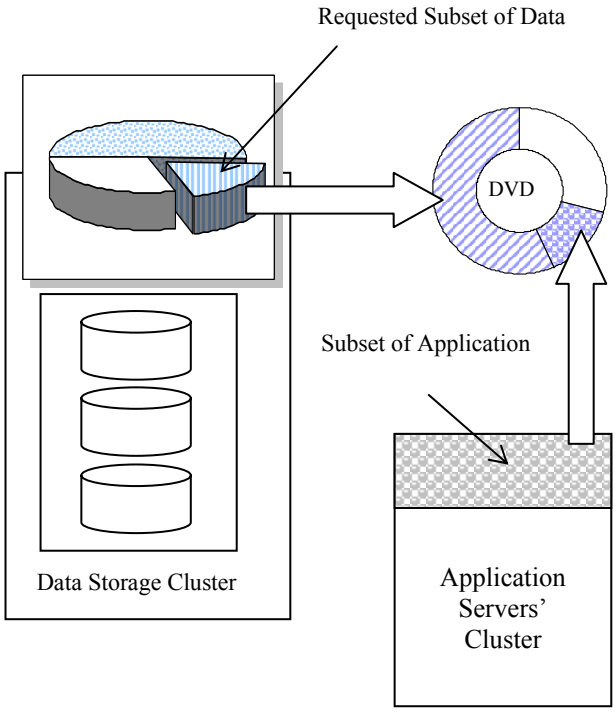


Figure 63. Creation of TerraFly DVD Replica

5.3.6. TerraFly Replica Example

We use the TerraFly replica installed at FMC (Florida Memorial College, [121]) to illustrate the design and implementation of the replication process. This TerraFly installation will be used by the students to design aviation software application and to study concepts of geographic information systems, environmental, and computer sciences, as well as data mining and analysis techniques.

One computer server has been chosen to handle all the functionalities of TerraFly replica. The computer is a Dell [122] 2600 with dual 2.0GHz Xeon [123] processors, 3GB of RAM, and six SCSI 73 GB hard drives configured with RAID level 5. It has two 10/100 and one 10/10/1000 network cards to support network load balancing. The server is connected to a 10/100 switch on a fiber backbone with a full T1 Internet connection. It is placed on a dedicated subnet for better performance and throughput. An additional proxy server filters and caches web requests to the TerraFly replica. A separate computer with a hardware firewall further protects the TerraFly server from unauthorized access.

On the software side, the following installation steps have been performed:

- A Microsoft [124] Windows boot NTFS partition of 4GB size (plus the size of available RAM) and largest available size "data" partition were created.
- Microsoft Windows Server 2000 Advanced Server operating system was installed on a startup partition with all default settings and no IIS (Internet Information Server).
- All software patches available on Microsoft Windows update Web site were applied via the Windows Update mechanism.
- "Data" partition was formatted as an NTFS partition.
- IIS was installed for home directories on the "data" partition.
- Microsoft XML v.40 package was installed.
- Microsoft .NET SDK v.1.0 package was installed.
- Due to installation of XML, .NET packages and additional relevant patches available on Microsoft Windows update Web site were applied via the Windows Update mechanism.

- The TerraFly software package was remotely downloaded and installed.

This TerraFly replica has been configured to have its Web Server, Application Server, Ingestion Server and Database Server all reside on a single machine. This option is specifically reserved for limited size installations and was chosen for the initial TerraFly replica at the FMC site. Despite a single server installation instance, application components communicate with each other via virtual pathways in the same way they would work in an Enterprise multi-cluster environment. Disadvantages of the current deployment include not being able to employ the advanced redundancy, failover, load-balancing, or self-recovery features that are available in an enterprise-scale replica.

Since the dataset to be used at FMC replica was too large to employ FTP to copy it over the network, it was brought to the FMC site on several hard drives. The drives were connected to the FMC computers and then copied to the TerraFly replica server.

5.4. On-demand Geo-referenced TerraFly Data Miner

5.4.1. Summary

We present a comprehensive Internet data extraction tool adopted for the TerraFly Geographic Information System (GIS). TerraFly is a web-enabled system that allows users to virtually fly over remotely sensed data, including satellite imagery and aerial photography, using a standard Internet browser. The data extraction tool presented here is designed to augment the user's virtual flight experience with extensive data relevant to any given geographical point along the virtual flight path. The data presented to the user is retrieved from several server-side databases and is collected from the Internet data providers using our patented data extraction technology. Some data elements are

presented to the user as overlays, some in popup windows, and some via hyper-linking to third-party web sites.

5.4.2. Background

The availability and use of remotely sensed data has steadily increased since the first commercial imaging satellite was launched several years ago. To meet an increasing demand for this data, the TerraFly system has been developed. TerraFly is an interactive fly-over vehicle designed to aid in the visualization of remotely sensed and spatial data via the Internet. The system has a broad range of spatial data manipulation capabilities. TerraFly's features include:

- Smooth flight over spatial data: streaming incremental tiles to a Java applet allows users to virtually fly over available data.
- Synchronized flight: multiple frames allow users to fly in sync over different data sets.
- Compass Control: supports 360 degree of flight direction and speed. It is also used to control data refresh rate.
- Zoom In/Zoom out: allows the user to view spatial data at varying resolutions.
- Image Processing Filters: allow users to customize the display of data. Filters enhance the appearance of images.

We refer to [107] for further information about the TerraFly system.

While TerraFly gives users a unique visualization experience by flight simulation, a large number of its applications require additional data to be made available to the user alongside the imagery. This brings to life the concept of data mining based on

geographical location. More precisely, each geographical location is considered as a point identified by coordinates (say, by Easting, Northing, and Zone in a Universal Transverse Mercator (UTM) grid). By clicking a geographical point on the imagery, the user initiates a data mining process against numerous data sources to get information on the point and the area(s) the point belongs to. The data to be collected and presented to the user is diverse: demographic, social, economic, and environmental among others. The granularity of data is also variable. For example, the point clicked within the continental United States lies in some block group, census tract, zip code, county, and state. Each of these census designated areas has its own demographic dataset with specific format of data presentation.

5.4.3. Datasets Available to the User

The largest and most comprehensive dataset currently used in the data mining module is collected from the US Census Bureau, see [125]. The US Census dataset contains both textual and cartographic information and is based on the census conducted in the United States in the year 2000. The user gets demographic statistics for the following areas which contain the geographical point clicked: Block Group, Census Tract, Zip code, City, Census County Division (CCD), Congressional District, County, Metropolitan Area, and State. A quick look report gives the user area population, square mileage, number of housing units, and percentage of territory covered by water.

A mouse click on any of the displayed areas delivers a comprehensive demographic report to the user. This report provides detailed figures on population, including race, age, sex; on households including size, type, presence of children, non-

relatives; on families including type, size, race; imputations of sex, race, age by housing units including urban/rural, occupancy/vacancy status; and other information. The data mining tool supplements the demographic report with an Expanded Map Album, containing the following maps encompassing the point of interest:

- Far ZoomOut map;
- Regional map;
- Street Detail map;
- Median Family Income ZoomOut map;
- Population Density ZoomOut Map.

The Geographic Names Information System (GNIS) database contains information about almost 2 million physical and cultural geographic features in the United States, see [119]. It is used as another data source for TerraFly's on-click data mining tool. When the user clicks on the imagery to identify the geographical point, she gets a list of the five nearest objects retrieved from the GNIS database. The distance in feet or miles and a direction arrow indicating the proximity of the object to the point clicked is calculated by the data mining module and is displayed along with the name of the object. Parks, rivers, bridges, distinguished buildings and monuments are some examples of GNIS objects. The names of the objects are also clickable for further data mining: a mouse click initiates a search for additional information about the object using the Google search engine, see [126]. However, to reduce a number of returned web pages related to the object and to force the search output to be more geographically specific, the data mining module calculates the zip code of the object area and passes it to Google as

an additional search parameter. A "More" button is presented to allow the user to get 100 nearest GNIS objects from the point clicked.

Proven to be important for travel applications, a Hotel dataset is also available for data retrieval. Presentation of hotel data to the user follows the same rules as GNIS objects: distance and direction arrow from the geographical point clicked and a "More" button for a larger set of records. However, unlike the GNIS set, the hotel dataset provides detailed information. Besides the hotel's name, a description of the hotel facilities, policies, price range, address, and indoor/outdoor photos are displayed for the user to easily compare nearby hotels and to make a decision on hotel preference. Hotel reservation is one button click away with the user invited to specify dates to check availability and further process of booking.

A similar approach is used to display real estate properties for sale in the vicinity of the point of interest. General and real estate specific data is mined for the user: property overview including price, square footage, number of baths, year built; property description including photo, address, community information, property features; contact information, including real estate agency name, address and phone numbers. A uniform resource locator (URL) is computed to lead the user to the particular property on the data provider's web site, where the original data is not ameliorated and has a different look-and-feel.

School and crime data may be viewed by the user in addition to real estate properties. This data characterizes neighborhoods to a large extent. General information on each school includes name, address, contact phones, type (public or private), grade levels, total number of students and number of students per teacher. Detailed school

information contains enrollment by race/ethnicity and by grade, various statistical figures, school identifications, and programs. Crime data is related to the zip code of the clicked geographical point. The major characteristic is crime rating, which is a weighted average for the entire zip code.

TerraFly's on-click data mining module provides access to several datasets related to natural and environmental resources. A simple feature displays current time and weather conditions in the area of the geographical point clicked. Additional historical weather and basic astronomical data for the region is available via the URL. The Federal Emergency Management Agency supplies the data mining module with hazard information, most notably hazard maps, see [127]. Floods and earthquakes data are of particular importance for any region. The fire detection program of the National Oceanic and Atmospheric Administration is a data source for urban and forest fires, [128]. After processing of the fire dataset, TerraFly's module displays date/time, coordinates, direction, temperature, land cover and distance to the point clicked by the user assembled in one table.

To easily correlate the above mentioned datasets with the imagery, the data mining tool displays a static aerial photography image of the area encompassing the point clicked. The user is able to instantly change the resolution of the aerial photography by zooming in and out in the range of 1-64 meters per pixel. The image contains several layers of data:

- icon of the geographical point clicked,
- street names,
- highways,

- hotel icons and names.

The user may "travel" in any of eight compass directions to adjust or further explore imagery beyond the initial static image.

5.4.4. Design of the Data Mining Module

TerraFly's data mining module has been designed to easily add and remove data sources for the application. Hypertext transfer protocol (HTTP) is adopted as the internal interface between the server side and the data providers. All data requests are sent via HTTP using URLs. Most data is returned to the client side in hypertext markup language (HTML) or plain text formats.

Three types of data providers may be identified for the data mining module, namely:

- Third party service. The data mining module calculates the parameters of the URL to the third party web site. Thus, the URL with some particular data request is sent to the external data provider.
- TerraFly's internal data provider. Internal providers are usually implemented on a separate physical server with web server software installed and running. The Common Gateway Interface (CGI) programs reside on the server to receive data requests, query the database and send query results back to the client side. In this case the database is stored in-house; it usually resides on a separate database server to provide easier system maintenance.

- TerraFly's data service. The data service receives the requests, compiles and sends web agents to collect data from the Internet in real time. We refer to [129] for the description of this technology.

The server side of the data mining module is yet another CGI program. When the user clicks a geographical point in TerraFly, TerraFly calls the CGI program with the coordinates of the point and other parameters. The server side program calls providers for the initial data and generates the HTML page to be presented to the user. Based on the user's preferences and data mining activity, the program calls the corresponding data providers for additional datasets.

One internal service is not directly seen by the user, but is heavily used by the data mining server side program. This is the spatial service, which is responsible for conversion of geographical coordinates into place names and vice versa. This service performs the following functions:

- Depending on the request, given geographical coordinates are transformed to street address, zip code, name of the nearest city, and so forth.
- Given street address it returns geographical coordinates. Given name of geographical area (like zip code, county, state) returns official coordinates of the center of the area.
- Given rectangular geographical area it returns zip code(s), district/city/county/state name(s), associated with this area.

The importance of the spatial service is that it allows the integration of textual data with remotely sensed imagery.

TerraFly displays remotely sensed imagery and is designed to work in terms of geographical coordinates. Most of the sources return data in terms of addresses and place name with no geographical coordinates. The spatial service supports association between names and coordinates thus allowing TerraFly together with its data mining module to provide the user with a unique application.

The spatial service is implemented using algorithms described in [130]. Being designed for specific conversion needs, the service is exceptionally fast, handling up to several thousand requests per second.

5.5. Autopilot Scripting for the TerraFly GIS

5.5.1. Summary

In this section we discuss an autopilot user interface for the TerraFly Geographic Information System (GIS). TerraFly is a web-enabled interactive GIS that allows dynamic browsing, composing, and navigating over satellite, aerial photography and other remotely sensed imagery received from a variety of data sources as well as information about the data being viewed. Autopilot technology supports the automation of TerraFly's features otherwise run by a user in manual mode. These features can be automatically shown in a pre-defined sequence that allows the user to experience TerraFly's multimedia capabilities without direct interaction. The applications for autopilot are numerous in the field of e-education including environmental education, training for emergency preparedness in urban areas, and the study of crops in rural areas via the Internet.

5.5.2. Background

TerraFly is a software tool that allows Earth Science information to be displayed and analyzed through the Internet's global information infrastructure by any connected users. TerraFly's Internet capability allows the system to access numerous heterogeneous multimedia data sets without the installation of any specialized GIS programs. The applied technology of streaming incremental image tiles to a Java applet allows the user to virtually fly even when high-speed connectivity is not available while employing a modem connection. While flying over imagery, the user also sees various overlays, such as road names, application-relevant points that are hyperlinked to more information, shaded zones, etc. The user can also view multi-spectral data, visualizing the application of various algorithms and filters of multiple spectral bands or datasets. The user can compare imagery of the same area shot at different times. We refer to [107] for more information related to TerraFly.

While many users prefer to use a keyboard and mouse to navigate TerraFly (we call this *general mode*), there are many applications that require an automatic interface to the system (we call this *autopilot mode*). Flying over official evacuation routes for the purposes of better hurricane preparedness in the Miami area could serve as an example of such an application. In this case the flight should be pre-programmed and delivered to the public's attention via an Internet browser.

To accommodate the above-mentioned TerraFly applications, we propose an autopilot interface. This proves to be beneficial for application developers and their respective users. The developers program the flight by writing a set of simple instructions to be sent to the TerraFly system. This programming process does not require much time

and knowledge therefore minimizing the learning curve. The users receive a new visualization tool for their geographic orientation needs.

5.5.3. Requirements of Autopilot

Autopilot is a mechanism (or a set of functions) that controls the behavior of the TerraFly application, as well as flow-control and decision-making led by a user script. Autopilot has to meet certain requirements imposed by the TerraFly system, potential applications, developer's needs, and convenience to the end-users. We will now briefly discuss these requirements.

1. Autopilot must simulate all user input/actions. The TerraFly system responds to user input/action in the following way. It catches user input (keyboard strokes or mouse operations) and traps them into the event processing method. The event processing code gathers necessary parameters to perform the according action. The system then performs the action. To enable autopilot, the TerraFly call interface has been modified to receive input programmatically in addition to manual input.
2. Autopilot asks/receives the status of all visible controls of TerraFly. The status of all visible controls of TerraFly’s windows is stored in a memory-buffer in the user's computer. This status comprises data retrieved from each control's attributes and methods. The TerraFly call interface has been improved with added functionality to read and modify this information; the autopilot module retrieves this data via the call interface.

3. While running an *autopilot script* (a set of flying instructions for TerraFly, usually stored in a plain text file), the autopilot module is able to call another autopilot script from the TerraFly server. This gives greater flexibility to the developer of TerraFly applications that use the autopilot feature.
4. The autopilot module has the ability to synchronize its time from the host computer clock. This has proved to be important for visual presentations where flight timing is an important aspect.
5. Autopilot fully supports the following categories of TerraFly features.

Functions that control the behavior of all flight windows, such as:

- Display geo transform information.
- Adjust flight speed and direction.
- Keep reference point at all times.
- Goto street and goto place.
- Return actual time all flight windows have needed to fly one step.

Functions that control each individual flight window, such as:

- Resolution of the imagery.
- Color bands of the imagery.
- Imagery data source or image describing template.

Functions that coordinate flight windows within the application, such as:

- Open a new flight window.
- Return how many flight windows currently exist.
- Bring a specific flight window on the front.

- Bring a specific flight window to the back.

Miscellaneous functions:

- Get system time.
- Play sound.

5.5.4. Autopilot Scripting Language

Common LISP (see [131, 132]) is used as the scripting language for TerraFly's autopilot. LISP has been in use for 50 years. It was designed to run in both interpreted and compiled modes. The Java source code for a LISP interpreter is freely available. With minor improvements and tuning to meet TerraFly's needs, the interpreter supports all of the necessary functionality required of autopilot. The LISP interpreter we are using supports two types of flow controls: *IF-THEN-ELSE* and *WHILE*. The combination of these two controls fulfills all of autopilot's flow-control requirements. The interpreter also provides support for user-defined functions, which can be defined inside autopilot scripts. Another advantage of choosing LISP is that the language is easy to understand, especially for those TerraFly applications developers who do not have knowledge of more traditional programming languages.

In choosing LISP, we have also kept the following considerations in mind:

- The LISP interpreter has a smaller memory footprint than most other programming language interpreters.
- LISP dialects are widely used for server-side programming – SCHEMA, which has a LISP-like syntax with only slight differences from Common LISP, is a very popular scripting language as are Perl, TCL, and Python.

Taking into consideration the popularity of LISP dialects in scripting and the need for a minimal memory footprint since the script language interpreter will be running within the browser environment, we consider LISP to be a practical choice.

The TerraFly system provides interfaces (methods) that support the full functionality of flight controls; autopilot invokes these methods. In the case of LISP, the autopilot scripting language implementation did not require substantial changes to the internal implementation of TerraFly. This significantly minimizes the effort to switch to another scripting language if we choose to do so in the future. This software reusability strategy has been widely adopted in software design and proves to be very practical in this case.

5.5.5. Implementation Strategy

Figure 64 shows how the autopilot module interacts with the TerraFly system.

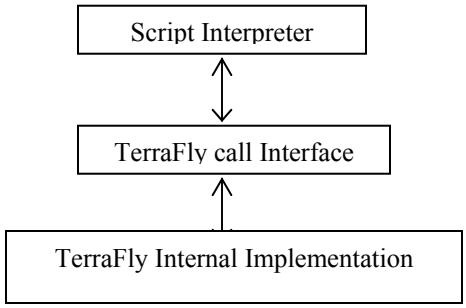


Figure 64. Autopilot Module Interaction

The autopilot Script Interpreter performs the following tasks:

- Reads autopilot script
- Prepares an execution plan for the user script based on its semantics and built-in functionality.

- Interacts with TerraFly via the Call Interface to execute the script instructions.

The TerraFly Call Interface is a set of open methods provided by the TerraFly client-side program. The interface supports all necessary approaches to control the behaviors of the TerraFly application, hides much of the complicated implementation details and provides a higher level of abstraction. Most of the methods the interface provides are already used when the TerraFly application is run in general mode, so the TerraFly Call Interface is nearly optimal since it is used in both general mode and autopilot mode. The utilization of this interface in creating autopilot also resulted in a significant reduction of the development cycle. Therefore, autopilot benefits from the general mode application as much as possible. A higher level of abstraction allowed us to divide the autopilot development process into several relatively independent parts. Its modular design provides easier system wide development and maintenance.

5.5.6. Autopilot Example

As an illustration of autopilot's functionality in a real-life scenario, we have chosen the American Frontiers trail application for TerraFly. The American Frontiers trail follows a route from Mexico to Canada on public lands in the United States, see [133] for details. A TerraFly autopilot script was implemented to allow Internet users to virtually fly over the trail.

Figure 65 shows the autopilot script for one part of the American Frontiers trail. The script instructs TerraFly to start virtual flight over Columbus, New Mexico at a resolution of 32 meters per pixel. The flight then follows the trail zooming out to imagery at a resolution of 64 meters per pixel. The virtual flight depicted in this example ends at

the Pine Grove Campground, near Happy Jack, Arizona, when TerraFly is instructed to gradually zoom-in to imagery until it reaches a resolution of 2 meters per pixel.

```
(setimagesource 0  "best_available")
(SetRefPointUtm 250185.656 3524351.500 13 t)
(showflyer 0)
(setres 0 32.0)

(setflyxy -499.30 1414.65)
(while (lte (GetUTMNorthing) 3543291.250) (flyonestep))
(SetRefPointUtm 243500.859 3543291.250 13 t)

(setflyxy 1.49 1500.00)
(while (lte (GetUTMNorthing) 3622015.500) (flyonestep))
(SetRefPointUtm 243578.859 3622015.500 13 t)

(setflyxy 1497.65 85.27)
(while (lte (GetUTMEasting) 259386.688) (flyonestep))
(SetRefPointUtm 259386.688 3622915.500 13 t)

(setflyxy -958.16 1154.16)
(while (lte (GetUTMNorthing) 3644704.250) (flyonestep))
(SetRefPointUtm 241298.031 3644704.250 13 t)

(setflyxy 1474.07 277.72)
(while (lte (GetUTMEasting) 707830.188) (flyonestep))
(SetRefPointUtm 707830.188 3732599.750 12 t)
(setres 0 64.0)

(setflyxy -1390.90 561.65)
(while (gte (GetUTMEasting) 677520.750) (flyonestep))
(SetRefPointUtm 677520.750 3744838.750 12 t)

(setflyxy -1327.43 698.53)
(while (gte (GetUTMEasting) 585400.250) (flyonestep))
(SetRefPointUtm 585400.250 3793314.750 12 t)

(setflyxy -1495.23 119.68)
(while (gte (GetUTMEasting) 504445.344) (flyonestep))
(SetRefPointUtm 504445.344 3799794.500 12 t)

(setflyxy 1389.59 -566.39)
(while (lte (GetUTMEasting) 505876.906) (flyonestep))
(SetRefPointUtm 505876.906 3799211.000 12 t)
```

```
(setflyxy -783.56 1279.10)
(while (lte (GetUTMNorthing) 3876309.750) (flyonestep))
(SetRefPointUtm 458647.156 3876309.750 12 t)

(setres 0 16.0)
(setres 0 8.0)
(setres 0 2.0)
```

Figure 65. Example of TerraFly autopilot script

5.6. Applying TerraFly for Vulnerability Assessment of Mobile Home Communities

5.6.1. Summary

Many coastal environments have a large number of mobile home communities. These communities can experience large losses during hurricanes. By making these communities easier to identify, disaster management teams can more effectively mitigate these losses. We have applied our graphical internet-based spatial data browser, TerraFly, to this application. TerraFly allows multiple forms of spatial data to be integrated into one easily viewable form.

As described in this section, our project involved the creation of a database for the International Hurricane Center's "Hurricane Loss Reduction for Residences and Mobile Homes in Florida" project. This database includes USGS aerial photography imagery, maps of Florida, and mobile home community vector data overlain on the imagery. This database includes Geographical Names Information System data produced by the USGS for the counties relevant to this project.

The result of this project, the web-enabled TerraFly application, allows disaster mitigation researchers to easily identify weak spots in their plans as well

as in the structure of the mobile homes. Users of the application can "fly" over the data and zoom in on further details regarding each community to better assess its vulnerability.

5.6.2. Background

Efficient methods for well-designed loss prevention and mitigation programs are needed in order to avoid or minimize human and financial losses triggered by possible disaster strikes. Inspectors from the Federal Emergency Management Agency (FEMA) and disaster mitigation programs are required to visit focus areas in order to assess vulnerability. Once the damage is done and depending on the degree of a disaster, losses rise and recovery becomes a cumbersome process. Assessment of damages may also be delayed until floods recede and areas are safe. The mitigation process and individuals affected by the disaster find themselves once again at the mercy of nature, and recovery efforts are further delayed.

Mobile home communities are highly vulnerable sites to natural disasters. In hurricane prone states such as Florida, evacuation of such areas is often required prior to potential disaster strikes, so the availability of efficient loss prevention, reduction and mitigation procedures is a high priority.

Aided by the rapid evolution of remote sensing technologies, the High Performance Database Research Center (HPDRC) has developed TerraFly [60], a Web-enabled data visualization tool that presents a major contribution to the design and implementation of disaster mitigation procedures. High resolution imagery provides accurate and detailed information that facilitates the observation and analysis of specific

geographic areas. TerraFly enables the application of remote sensing technologies to vulnerability assessment programs by facilitating access to focus areas through the use of remote sensed imagery.

5.6.3. TerraFly Features Important to the Project

TerraFly is an interactive vehicle for 'flying' over remotely sensed data. The TerraFly system allows the visualization, analysis and manipulation of numerous types of remotely sensed data via any standard Web browser. It does not require the installation of additional Geographic Information Systems (GIS) software on the user's computer. This simplification greatly enhances the usability of our system and accessibility of spatial data due to user familiarity with browser technology.

5.6.3.1. System Features

Options and features currently available for the data sets are found below.

- Multiple Frame Support: TerraFly supports multiple frames containing various views over a specific location. As the user navigates over the data, frames are updated simultaneously to create a seamless flight-like experience for the user. When the TerraFly application is first launched three main frames are displayed. The Flight Control frame provides flight control tools and information relevant to the imagery and location loaded on the two additional windows. These windows or FlyFrames display aerial photography imagery, street maps and mobile home vector data overlain on remote sensed imagery.

- Multiple Data Type and Resolution Support: TerraFly allows the manipulation of numerous data types and resolutions without additional requirements for GIS-specialized software.
- Multiple Navigation Processes: The TerraFly system supports multiple types of navigation. Navigation may be achieved in the FlyFrames via the use of mouse events for flight direction control. Data may also be retrieved via place names, street address or geographic coordinate values [134].
- Latitude and Longitude: Geographic coordinates of the center point of the current image loaded are calculated and displayed. These values change continuously while the user is engaged in flying.
- Go-To Features: These features add additional functionality to the system.
    - Go-To Place: loads image data corresponding to a desired location.
    - Go-To Street: loads image data corresponding to a specific street address.
- Sensor Band Controls: These controls allow the user to manipulate the sensor band combinations of multispectral data types, such as Landsat TM data, to view false color images providing greater flexibility and availability of information. Landsat data users are able to select from a list of seven possible sensors for each color band. When a band combination is selected, the set of bands is retrieved from the database, the associated false color image is computed and the resulting image is displayed in the corresponding FlyFrame. TerraFly provides two ways of doing this:
    - Pre-defined Three-Band Combinations: This tool provides predefined sensor combinations. These are commonly used combinations that

provide interesting data for basic users as well as short-cuts for the advanced user.

  - Advanced Three-Band Combinations: This allows the specification of Red, Green and Blue values enabling the advanced user to create any desired three-band combination they wish to analyze.

- RGB Intensity Control: Three sliding selectors allow the user to increase or decrease the intensity of the color bands.
- New Fly Frame: Additional fly frames may be loaded for the display of imagery.
- New View Frame: Additional view frames that allow further processing of image data.
- Frame List: A list of currently loaded FlyFrames and view frames is provided.
- Lookup Conf: Advanced configuration allows the user to concentrate on specific object types when performing a search or retrieving information (buildings, populated places, parks, etc).
- Help: System manual and help documentation are provided for user support.
- Data Dispensing Capability: User specification of an Area of Interest (AOI) is enabled via a graphical user interface.
- Image Processing Filters: enable users to customize the display of data and to enhance the appearance of images.

5.6.3.2. Interactive Interface

TerraFly offers a GUI-based interface that further enhances the usability of the system. TerraFly's user-friendly interface efficiently implements user commands and

system responses via mouse events and graphical tools. Interaction between the system and the user becomes an intuitive process that does not require specialized knowledge of additional GIS software or computer languages.

5.6.4. Vulnerability Assessment Project

The project implements a customized version of our TerraFly application aimed toward the assessment of absolute and relative vulnerability of the mobile home communities in Florida, and the understanding of possible issues that may influence hurricane loss mitigation. TerraFly technology is used to identify mobile home communities within the designated counties by way of colored highlighting of each park. Highlighted mobile home parks are made "clickable" to allow viewers to access information about each park such as name, address, number of units in the park, general demographic characteristics, integration of the park within surrounding community, photographs of park, and other information.

5.6.4.1. System Specifications

This project includes the creation of a database containing aerial photography imagery, maps of Florida, and mobile home park vector data overlain on the imagery. The database includes the Geographical Names Information System (GNIS) data produced by the United States Geological Survey (USGS) as well as information for Mobile Home Parks in Miami Dade, Broward, Hillsborough and Pinellas counties provided by the International Hurricane Center (IHC).

The system is designed to provide continuous access to georeferenced objects associated to mobile home locations overlain over remotely sensed imagery and maps.

The overlain data displays mobile home park names and highlights such communities to the user. Additional information over the location of interest is available through the creation of static html pages linked to the overlain data.

Additional street maps are created and provided to the user. Maps are displayed simultaneously to allow efficient geolocation of areas of interest. Maps display postal code information and boundaries as well as major streets and highways.

5.6.4.2. Implementation

A mobile home park filter layer was created and merged with 1 meter USGS Digital Ortho Photo Quads (DOQQ) derived from aerial photography. The filter layer serves to highlight the entire area of the mobile home park. Highlighted areas over mobile home communities were linked to related information and pictures.

Geographic coordinate parameters were attained via Terrafly technology by visual identification of focus areas. Once identified, these were "marked", using the mark feature, and metadata was generated by TerraFly that would later be used to create a database of mobile home parks and their respective geographical coordinates, among other attributes. These geographic parameters would later be used to create a unique mobile home park vector layer.

Street maps were created to aid the user in identify the location they are "flying" over. A user can select to "fly" over both aerial photography and street maps simultaneously to attain a better reference point. Street maps were created using several vector layers including streets, major roads, highways, and zip codes. We developed software to read these layers and create two rasterized map layers, a zip code layer, and a

street, major roads, and highway layer. The mobile home parks filter layer was created using the same proprietary software.

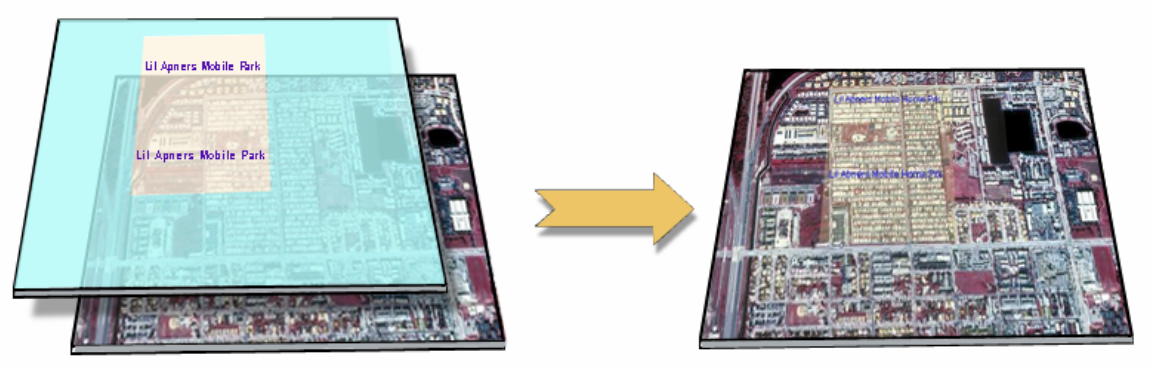


Figure 66. Filter layer highlighting the mobile home park area and name merged with USGS DOQQs

Each rasterized layer is made up of smaller components called tiles. Each tile is a fixed size square (512 x 512 pixels) PPM (portable bitmap) and each is loaded into the database along with geographical coordinates and vector layer metadata. The USGS DOQQs are tiled into square (512 x 512 pixels) colored PPM images and loaded into the database with geographical indexes and metadata [135].

5.6.5. Results

Our TerraFly system presents a major contribution to the design and implementation of disaster mitigation procedures. TerraFly makes remote sensing data available for the identification and analysis of vulnerable communities. This system uses an efficient database schema [8], data structures, search procedures and calculation functions as well as appropriate client/server work distribution to allow advanced analysis and processing of data at real-time speed. The complexity, however, comprised in the application of GIS technologies is maintained transparent to the user.

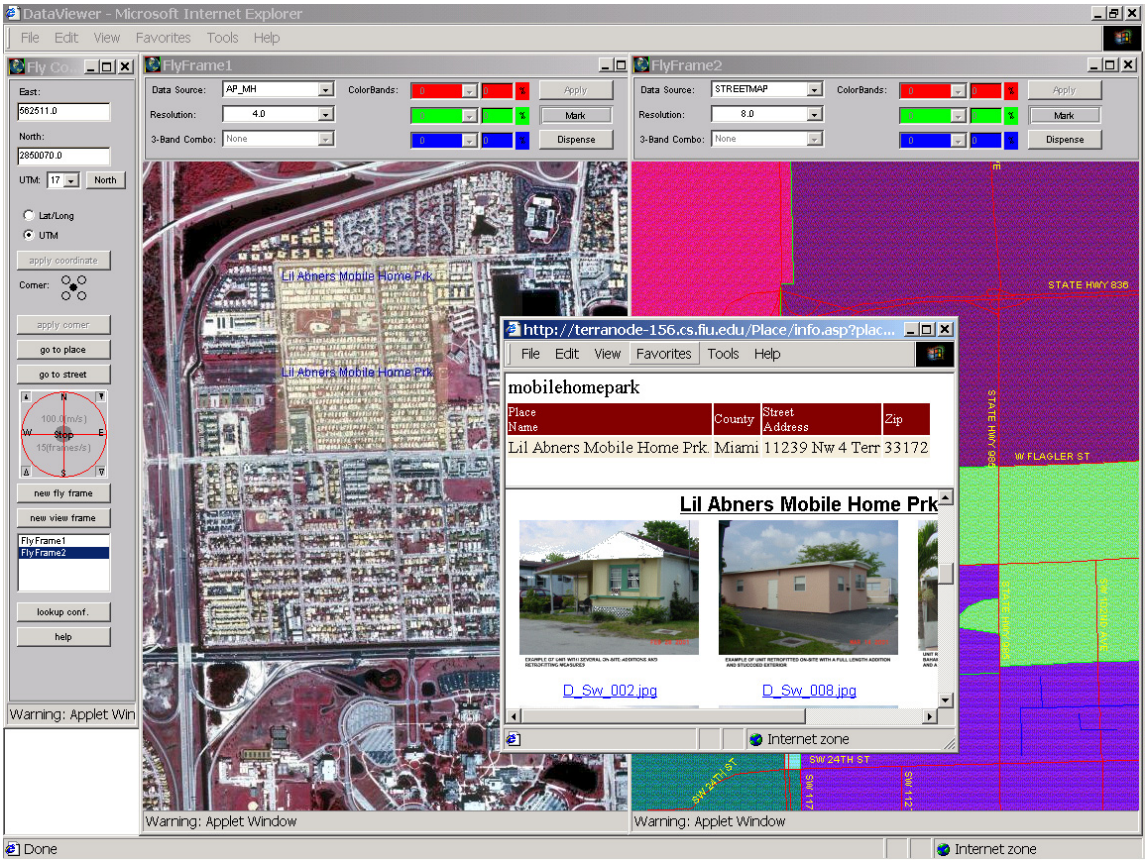


Figure 67. TerraFly application for Mobile Home Project

We feel that the approach presented here helps reduce the time, expenses and difficulty involved in designing efficient loss reduction techniques. High-resolution space data available through our system automates the vulnerability analysis of focus areas, allows efficient and accurate identification of such areas and quick access to related attributes.

5.7. TerraFly Application for Flight Planning and Training

5.7.1. Summary

Flight planning is an essential part of flight preparation activities. Several software products are now available to assist pilots in completing flight plans in a fast

and efficient way. However, the flight planning programs offered are standalone computer applications. The pilots who travel to numerous destinations are often unable to use the same computer for all their flight plans. To avoid this inconvenience we propose to use TerraFly GIS as a platform for an Internet service for flight planning. This will allow pilots to prepare future flight plans from any computer that has Internet access. TerraFly also has a set of unique features that will enrich the functionality and visualization capabilities of the flight planning process as well as computer training for future pilots.

5.7.2. Background

TerraFly [107] is an interactive fly-over vehicle that permits the visualization of remotely sensed and spatial data via the Internet. The database used by TerraFly contains textual, remotely sensed and graphical data (graphical maps), which can be viewed and manipulated using any standard browser. Textual data is available for the description and location of specific areas of interest. Graphical maps aid in the visualization of remotely sensed data. Internet capability allows the system to access numerous data sets without the installation of any specialized GIS (geographic information system) programs. A friendly graphical user interface is provided for ease of use.

Among a broad spectrum of TerraFly features, we highlight a few that will be specifically useful for the flight planning and training application:

- smooth flight over spatial data: streaming incremental tiles to a Java applet allows users to virtually "fly" over imagery or maps.

- Compass Control: allows 360 degree control of flight direction and speed. The feature is also used to control data refresh rate.
- Geolocation ID: Geo-coordinates of the center point of the data are constantly calculated and displayed in a Flight Control window.
- Go-to Coordinate and Place: allows user to instantly get to an area of specific interest.
- Zoom In/Zoom Out: allows the user to view spatial data at varying resolutions.
- Information Overlay: allows users to overlay data from different sources on top of imagery and digital maps.
- Autopilot: pre-programmed virtual flight over a particular area with desired change of direction, resolution, and flight speed.

We will now briefly describe the data objects that will appear in the flight application we are planning to design based on the above TerraFly features. A flight plan is an outline of all the major criteria that are important to a flight in a specific region. It consists of the origin airport, the destination airport, and fixes between that are defined in relation to radio aids to navigation. The filing or entry of a flight plan is imperative before every flight and involves careful preparation. The pilot or co-pilot usually performs this duty within the hour prior to a flight. At most airports all the calculations necessary for filing a flight plan are done using pen and paper and thus involve little computerization. Flight plans are usually filed by telephone, in person, or by radio with the FAA (The Federal Aviation Administration, [136]) Air Traffic Control Facility.

Of all the various factors that can affect flight, weather is the most crucial. Weather can be divided into a wide range of categories such as type of cloud cover, wind velocity, temperature, and pressure. All of this information can be downloaded from a

meteorological briefing system available via the Internet. Other factors that are important in filing a flight plan include:

- The Aircraft Call Sign, i.e. the designated symbol that is demarcated for each aircraft, usually a combination of letters and numbers, e.g. CF456 or N5432;
- The pilot in command and the number of passengers;
- The altitude of flight;
- The true airspeed – average speed that the aircraft will fly for duration of flight;
- The amount of fuel on-board;
- The flight routing – a graphical representation of the flight route;
- Elapsed time – estimated time from take off to destination;
- Alternative airports – in event of an aircraft emergency or problems at the intended destination.

There are two types of flight plans, namely VFR (Visual Flight Rules) and IFR (Instrument Flight Rules). Filing of a VFR plan is not necessary, but is recommended.

5.7.3. Flight Planning and Training Software

In this section we will shortly review the functionality of the flight planning and training software currently available for the pilots. We have evaluated the products of Navigation and Planning Services [137], Jeppesen Sanderson, Inc. [138], and Maptech, Inc. [139].

Navigation and Planning Services have developed flight plan software called Destination Direct [140]. Destination Direct is a Windows-based computer software package. This software has the ability to provide distance, time and fuel calculations, a

comprehensive navigation log and a flight plan. The software also has the capability to recommend rational stopover locations for long journeys, calculate weight and balance, and cost estimates. Destination Direct can be used to assist in planning both VFR and IFR flights. Some of its notable features include the Route Table, which gives a detailed view of the planned route, and the Navigation Log, which provides all the information a pilot needs during the flight. This software has been tested by actual pilots who have had years of experience. It has a user-friendly interface that is straightforward for the basic computer user.

Destination Direct is also able to provide the following features for flight planning:

- Ability to locate cities when one is unaware of the name or identifier for the airport;
- Ability to enter computer automated or personalized waypoints and stops;
- Simplification of flight planning by storing information on the pilot's aircraft and flight preferences in a database. Future flight plans can be built automatically to match the flying style of the pilot.
- Accurate time management features for scheduling departures and estimated arrivals. The software has a feature that recommends what time you should set your alarm clock to get to the airport; it takes time zone changes into account;
- Preset data sets for weather. This automatically downloads the latest weather data from DTC DUATS (Digital Technical Control Direct User Access Terminal System, for example see [141]) or AccuWeather [142]. Latest Fuel prices can also be downloaded from Fillup Flyer Fuel Finder.

- Printouts can be made of Airport Information, FAA flight plan, Route Table, Fuel requirements, Weight & Balance Information, or FBO (Fixed Based Operator) and hotel details.
- Graphic layout of weight and balance information makes it simple to plot the center of gravity envelope and weight and balance for any flight.

Jeppesen, an industry leader in aviation products and resources, offers flight planning software called FliteMap and FliteStar. This software enables users equipped with a notebook computer and a GPS (Global Positioning System) receiver to get a high-resolution electronic moving map, flight planner, and easy DUATS access interface. It also offers a tool to produce accurate flight plans by providing a thorough look at weather, airport information, FBO data, airspace, terrain, and weight, balance, and cost information for each of the flights. Additional planning features include:

- Point-and-click and automatic routing;
- Routing between VORs (VHF (very high frequency) omni-directional ranges), NDBs (non-directional radio beacons), airports and user-defined waypoints;
- Direct and great circle paths;
- Routing around special use airspace;
- Color-coded elevation information in the plan views and profile views;
- Terrain avoidance flight planning option;
- Connection to GTE (general telephone and electric) or DTC DUATS via the Internet;
- Graphic weather overlays;
- Printed navigation logs, flight plan forms, etc.;

- Weight and balance, plus CG (center of gravity) location calculations;
- Over 50 aircraft models;
- Sunrise and sunset calendar;

Online aeronautical chart resources can be obtained from Maptech. Maptech's digital mapping products include navigation software; digital, nautical and aeronautical charts; and topographical maps. Maptech obtains charting data from NOAA (National Oceanic & Atmospheric Administration, [143]). Their VFR and IFR Aeropacks are direct replications of the FAA paper charts.

By analyzing the features available in third-party software packages related to flight planning and training, we intend to design an Internet-based flight application enriched with TerraFly functionality.

5.7.4. Design of the TerraFly Flight Application

The general TerraFly system appears to be naturally suited for the flight application. The concept of virtual "flight" implemented in TerraFly resembles, to a great extent, the conditions of a real flight over some region/terrain. The concept of changing virtual flight resolution is similar to the one of changing altitude in real flight conditions. The same is true of the compass flight direction, flight speed, and so forth. It is natural to utilize the current TerraFly interface for the flight planning and training application, see Figure 68. As the virtual pilot approaches the airport, the view can shift from a topographical angle to an approach angle; this would allow the user to visualize the approach as it would be seen from the cockpit.

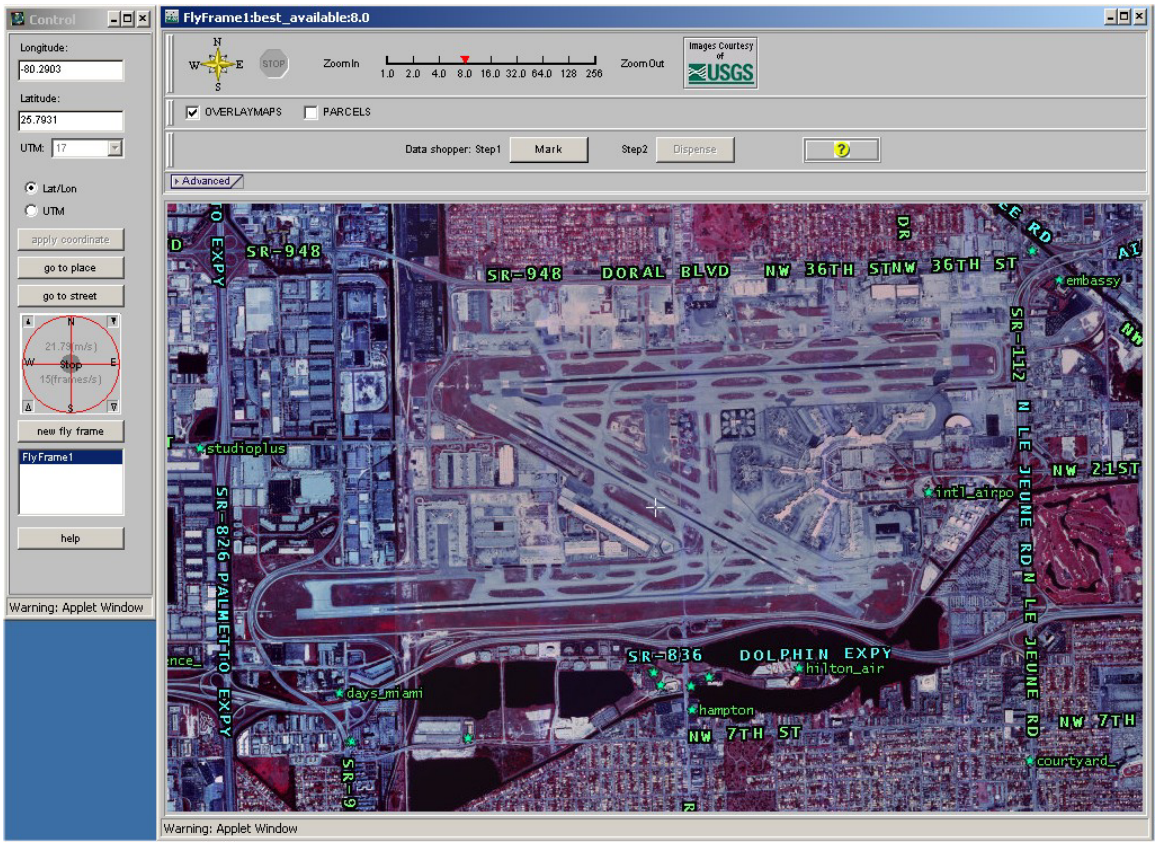


Figure 68. Miami International Airport and surrounding area, Miami, Florida featured in TerraFly

We have also decided to use remotely sensed data already available via TerraFly. In particular, satellite and aerial photography imagery provided by USGS (United States Geological Survey, [100]) will be extensively used in the flight application. This dataset comprises 15 terabytes of imagery and maps covering almost the whole territory of the continental United States. The USGS dataset is constantly updated with recent imagery. It allows the application to simulate flights over a large territory that has busy air traffic in the skies, thus making simulation closer to reality. Virtual flights over imagery are more suitable for pilot training. For flight planning, virtual flights over topographical maps with appropriate symbols for the radio aids to navigation (VOR's/NDB's) seem to be

more appropriate. The resolution of the imagery and maps is as high as 1 foot per pixel, at which the imagery appears on the computer screen as close to that viewed out the window of a low altitude helicopter or small aircraft during flight. Lower resolution imagery (in the range of 4-16 meters per pixel) allows one to simulate the flight of larger aircraft covering long distances at a higher altitude. Thus, we have to concentrate our efforts on the design of information overlays onto the imagery/maps and information support of the flight application via TerraFly's on-click data mining tool, [120].

We will first discuss the overlays onto the imagery. We plan to utilize some of the overlays currently available in TerraFly. In particular, the overlay of administrative boundaries (states, counties, metropolitan areas, zip codes), town/city names, airport/heliport codes and names, and major highways might be displayed. At high resolutions it would be beneficial to show boundaries of airports, and to label runways and control towers. These overlays are not available on TerraFly and have to be separately generated. Additionally, the application needs a layer of flight routes, radio aids to navigation, and FAA defined airways. Note that each overlay has to be generated separately for each resolution of the imagery. Depending on the resolution, the layer shows less or more details. This allows us to maintain the readability of a map or image when a combination of overlays is applied over each particular resolution. The color scheme for lines and point objects of each overlay needs to be carefully selected to allow a pilot to clearly see each overlay in a given combination. Finally, the pictograms for airports, VOR's, NDB's, and airways should match what is found on FAA charts, be of the right size, and have readable captions.

TerraFly has a set of software tools to generate imagery overlays. However, the TerraFly system needs a set of geo-referenced data objects to correctly produce the overlay. These datasets have to be collected and appropriately formatted prior to overlay generation. To collect and format this data we are planning to employ Internet data mining technology [129], developed at HPDRC (High Performance Database Research Center, [16]).

The same technology would be applied to collect data for the databases supporting the on-click data mining tool for the TerraFly flight application. The Internet data sources, provided and maintained by FAA and IATA (International Air Transport Association, [144]) would play a major role in data acquisition. Other related data sources are yet to be identified and analyzed. The following datasets need to be acquired, verified and used in the flight application:

- Airport dataset, including codes, full and short names, geographical coordinates as well as general information related to the airport, like elevation, dimensions of its runaway(s), etc.
- Aircraft dataset, including technical characteristics of the aircraft (maximum flight distance and speed, fuel load and consumption, etc.), number of passengers and crew.
- Weather dataset, including wind speed and direction, air temperature and pressure, visibility and cloud cover.
- Approach angle photographs of airports, similar to those provided by corporate airport qualification and familiarization services.

Besides the datasets, several software modules have to be written for the TerraFly flight application. Some of the programs might be described as follows:

- Distance calculator – given departure and destination airports calculate the distance and flying time;
- Calculation of weight and balance, as well as fuel consumption based on the aircraft model;
- Support for flight planning activity, including input forms, storage and maintenance of the flight route information, pilot/co-pilot and passenger data, aircraft and home base information, flight details.
- Support for flight training activities will include routine creation of scripts for the TerraFly autopilot tool, [145].
- Support to allow realtime weather data to affect virtual flight.
- Support for approach angle photographs to be displayed on virtual approach to an airport.

We are planning to fully utilize the current TerraFly computer cluster architecture, imagery storage/retrieval, overlay generation and security capabilities for the flight application. A separate server will be assigned to host the database to store and maintain datasets specific to the flight application.

5.7.5. Results

The TerraFly flight planning and training application is targeted towards the needs of both professional and enthusiast pilots and trainees. Professional and enthusiast pilots will use this tool for flight planning and virtual flights over the chosen routes. Pilot trainees will use the tool for better understanding of the existing flight preparation rules,

flight and navigation-related concepts as well as familiarization with the airports and their surroundings in the United States.

Unlike existing software tools, the TerraFly flight application will be fully Internet enabled. This will allow pilots to use it from any computer connected to the Internet that has a standard Internet browser. No software installation procedures would be necessary on the pilot's side.

List of References

1 D. Sitaram, A. Dan, and P. Yu. “Issues in the Design of Multi-Server File Systems to Cope with Load Skew.” Proceedings of the Second International Conference on Parallel and Distributed Information Systems (San Diego, January 20-22, 1993), IEEE Computer Society Press, 1993.

2 K. Hua. and J. Su. “Dynamic Load Balancing in Very Large Shared-Nothing Hypercube Database Computers.” IEEE Transactions on Computers, Vol 42 No 12 (December 1993) pp 1425-1439.

3 C. Lee and K. Hua. “A Self-Adjusting Data Distribution Mechanism for Multidimensional Load Balancing in Multiprocessor-Based Database Systems.” Information Systems, Vol 18 No 7 (1994) pp 549-567.

4 S. Stolfo, H. Dewan, D. Ohsie, and M. Hernandez. “A Parallel and Distributed Environment for Database Rule Processing: Open Problems and Future Directions.” *Emerging Trends in Database and Knowledge-Base Machines: The Application of Parallel Architectures to Smart Information Systems*. M. Abdelguerfi and S. Lavington, eds. IEEE Computer Society Press, 1995, pp 225-253.

5 J. Xu and K. Hwang. “Heuristic Methods for Dynamic Load Balancing in a Message-Passing Multicomputer.” Journal of Parallel and Distributed Computing, Vol 18 (1993) pp 1-13.

6 I. Ahmad and A. Ghafoor. “Semi-Distributed Load Balancing For Massively Parallel Multicomputer Systems.” IEEE Transactions on Software Engineering, Vol 17 No 10 (October 1991) pp 987-1004.

7 N. Rishe, A. Shaposhnikov, and W. Sun. “Load Balancing Policy in a Massively Parallel Semantic Database.” Proceedings of the First International Conference on Massively Parallel Computing Systems, IEEE Computer Society Press, 1994, pp 328-333.

8 N. Rishe. *Database Design: The Semantic Modeling Approach*. McGraw-Hill, 1992, 528 pp.

9 N. Rishe. “Efficient Organization of Semantic Databases.” *Foundations of Data Organization and Algorithms*. (FODO-89) W. Litwin and H.-J. Schek, eds. *Springer-Verlag Lecture Notes in Computer Science*, Vol 367, pp 114-127, 1989.

10 N. Rishe. “A File Structure for Semantic Databases.” Information Systems, Vol 16 No 4 (1991) pp 375-385.

11 Y. Zheng, J. Pieprzyk and J. Sebery. "*HAVAL - a one way hashing algorithm with variable length of output*." Advances in cryptology - AUSCRYPT 92, Lecture Notes in Computer Science, vol. 718, pp. 83-104, 1993.

12 B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, N. Ferguson. **"**Twofish: A 128-Bit Block Cipher." http://www.counterpane.com/twofish-paper.html.

13 R. Rodriguez. *Semantic Wrapping for Heterogeneous Database Management*. PhD thesis, Florida International University, 2002.

14 N. Rishe. Semantic SQL, internal document, High Performance Database Research Center, Florida International University, Miami, Florida, 1998.

15 X. Lu. *A Semantic Wrapper Used in Heterogeneous Database Systems*, Master's thesis, School of Computer Science, Florida International University, 2000.

16 N. Rishe, W. Sun, D. Bardon, Y. Deng, C. Orji, M. Alexopoulos, L. Loureiro, C. Ordonez, M. Sanchez, A. Shaposhnikov. "Florida International University High Performance Database Research Center." SIGMOD Record, 24 (1995), 3, pp.71-76.

17 R. Athauda. *Integration and Querying of heterogeneous, autonomous, distributed database systems*. Ph.D's dissertation, School of Computer Science, Florida International University, 2000.

18 G. Aslan, D. McLeod. "Semantic Heterogeneity Resolution in Federated Databases by Metadata Implantation and Stepwise Evolution." in VLDB Journal, Vol.8, No.2, 1999, pp.120-132.

19 E.F. Codd. *The Relational Model for Database Management*. Version 2. Reading, MA: Addison-Wesley, 1990.

20 N. Rishe, J. Yuan, R. Athauda, X. Lu, X. Ma. "SemWrap: A semantic wrapper over relational databases, with substantial size reduction of user's SQL queries." EDBT 2000 VII. Conference on Extending Database Technology March 27-31, 2000 -- Konstanz, Germany.

21 N. Rishe, J. Yuan, R. Athauda, S.-C. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, D. Vasilevsky. "SemanticAccess: Semantic Interface for Querying Databases." Proceeding of the VLDB conf., Cairo, Egypt, pp. 591-594, September 10-14, 2000.

22 HPDRC. *Semantic Binary Database C++ Interface Version 3*. High Performance Database Research Center. School of Computer Science, Florida International University, 1995.

23 HPDRC. *Semantic Binary Database Java Interface*. High Performance Database Research Center. School of Computer Science, Florida International University, 1995.

24 Sem-ODB, *http://hpdrc.cs.fiu.edu/_SemODB/text/*.

25 A. Silberschatz, H. Korth, S. Sudarshan. *Database System Concepts,* 4th edition, *McGraw-Hill*, 2002, 786 pp.

26 S. White, M. Fisher, R. Cattell, G. Hamilton, and M. Hapner. *JDBCTM API Tutorial and Reference.* Second Edition, Addison-Wesley, 1999.

27 "*Extensible Markup Language (XML) 1.0 (Second Edition)*", W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006, work in progress.

28 N. Rishe, L. Yang, M. Chekmasov, M. Chekmasova, S. Graham, A. Roque. "Mapping from XML DTD to Semantic Schema." 6th World Multiconference on Systemics, Cybernetics, and Informatics (Sci-2002). v. VII, pp. 450-455.

29 Java Applet. *http://java.sun.com/applets/*

30 Applet. *http://java.sun.com/docs/books/tutorial/*applet/practical/workaround.html

31 Java Servlet, http://java.sun.com/products/servlet/

32 J. Hammer, H. Garcia-Molina, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. "Information Translation, Mediation, and Mosaic-Based Browsing in the TSIMMIS System." In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995.

33 G. Mecca, P. Atzeni, A. Masci, G. Sindoni, and P. Merialdo. "The Araneus Web-based management system." Proceedings of ACM SIGMOD International Conference on Management of Data, 1998, Pages 544 – 546.

34 M. Bauer and D. Dengler. "InfoBeams--configuration of personalized information assistants." Proceedings of the 1999 International Conference on Intelligent User Interfaces, 1999, Pages 153 – 156.

35 M. Bauer, D. Dengler, and G. Paul. "Instructible information agents for Web mining." Proceedings of the 2000 International Conference on Intelligent User Interfaces, 2000, Pages 21 – 28.

36 M. Muslea, S. Minton, and C. Knoblock. “A hierarchical approach to wrapper induction.” Proceedings of the 3rd Annual Conference on Autonomous Agents, 1999, Pages 190 – 197.

37 World Wide Web Consortium. Document Object Model (DOM). http://www.w3.org/DOM.

38 A. Heydon and M. Najork. “Performance Limitations of the Java Core Libraries.” In Proceedings of the 1999 ACM Java Grande Conference, pages 35-41, June, 1999.

39 B. Adelberg and M. Denny. “Nodose version 2.0.” Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 559 – 561.

40 S. Grumbach and G. Mecca. “In Search of the Lost Schema.” In Proceedings of International Conference on Database Theory (ICDT'99), 1999.

41 S.-J. Lim, and Y.-K. Ng. “An automated approach for retrieving hierarchical data from HTML tables.” Proceedings of the 8th International Conference on Information Knowledge Management, 1999, Pages 466 – 474.

42 L. Liu, W. Han, D. Buttler, C. Pu, and W. Tang. “An XJML-based wrapper generator for Web information extraction.” Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 540 – 543.

43 A. Sugiura and Y. Koseki. “Internet scrapbook: automating Web browsing tasks by demonstration.” Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology, 1998, Pages 9 – 18.

44 W. Cohen. “A Web-based information system that reasons with structured collections of text.” Proceedings of the 2nd International Conference on Autonomous Agents, 1998, Pages 400 – 407.

45 L. Lakshmanan, F. Sadri, and I. Subramanian. “A Declarative Language for Querying and Restructuring the World-Wide-Web.” Post-ICDE IEEE Workshop on Research Issues in Data Engineering (RIDE-NDS'96), New Orleans, February 1996.

46 G. Arocena and A. Mendelzon. “WebOQL: Restructuring Documents, Databases, and Webs.” Proceedings of ICDE'98, Orlando, February 1998.

47 A. Mendelzon, G. Mihaila, T. Milo. “Querying the World Wide Web.” PDIS 1996, December 1996, pages 80-91.

48 F. Neven and T. Schwentick. “Expressive and efficient pattern languages for tree-structured data” (extended abstract). Proceedings of the 19th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2000, Pages 145 – 156.

49 D. W. Embley, Y. Jiang, and Y.-K. Ng. “Record-boundary discovery in Web documents.” Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data, 1999, Pages 467 – 478.

50 R. Doorenbos, O. Etzioni, D. Weld. “A Scalable Comparison-Shopping Agent for the World-Wide Web.” Proceedings of Autonomous Agents '97. pp. 39-48.

51 World Wide Web Consortium. XSL Transformations (XSLT). W3C Recommendation. http://www.w3.org/TR/xslt.

52 The World Wide Web Consortium, http://www.w3.org

53 G. Kappel, E. Kapsammer, S. Rausch-Schott, and W. Retschitzegger, “X-Ray - Towards Integrating XML and Relational Database Systems.” 19th International Conference on Conceptual Modeling, Salt Lake City, Utah, USA, October, 2000. pp. 339-353.

54 J. Shanmugasundaram, K. Tufte, and C. Zhang, G. He, D. DeWitt, and J. Naughton. “Relational Databases for Querying XML Documents: Limitations and Opportunities.” The VLDB Journal, 1999, pp.302-314.

55 D. Florescu and D. Kossmann. “Storing and Querying XML Data Using an RDBMS.” IEEE Data Engineering Bulletin, Special Issue on XML, Vol. 22, No. 3, September, 1999.

56 V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl “From Structured Documents to Novel Query Facilities.” Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May, 1994.

57 Excelon Home Page, http://www.exceloncorp.com/.

58 R. Goldman, J. McHugh, and J. Widom. “From Semistructured Data to XML: Migrating the Lore Data Model and Query Language.” Proceedings of the 2nd International Workshop on the Web and Databases (WebDB '99), Philadelphia, Pennsylvania, June 1999.

59 Tamino XML Database Home Page, http://www.softwareag.com/tamino/.

60 N. Rishe, “ TerraFly: A High-Performance Web-based Digital Library System for Spatial Data Access.” ICDE Demo Session, 2001.

61 J. Bosak, T. Bray, et. al. “W3C XML Specification DTD.” http://www.w3.org/XML/1998/06/xmlspec-report.html.

62 D. Fallside (Ed). “XML Schema Part 0: Primer.” W3C Recommendation, May 2, 2001, http://www.w3.org/TR/xmlschema-0/.

63 A. Deutsch, M. Fernandez, and D. Suciu, “Storing Semistructured Data with STORED.” Proceedings of the ACM SIGMOD International Conference on Management of Data, Philadelphia, Pennsylvania, USA, June 1999.

64 D. Lee and W. Chu. “Constraints-preseving Transformation from XML Document Type Definition to Relational Schema.” Proc. 19th Int'l Conf. on Conceptual Modeling (ER), Salt Lake City, Utah, October, 2000.

65 I. Manolescu, D. Florescu, and D. Kossmann. “Pushing XML queries inside relational databases.” Tech. Report No. 4112, INRIA.

66 J. Cheng, J. Xu. “XML and DB2.” ICDE 2000. pp. 569-573.

67 S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. “Oracle8i – The XML Enabled Data Management System.” ICDE 2000. pp. 561-568.

68 T. Bray, J. Paoli, C. M. Sperberg-McQueen and E. Maler (Eds), “Extensible Markup Language (XML) 1.0 (Second Edition).” W3C Recommendation, October 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

69 R. Xiao, T. Dillon, E. Chang, and L. Feng. “Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema.” In 12th International Conference on Database and Expert Systems Applications (DEXA 2001), Munich, Germany, September 2001. pp. 795-804.

70 Sem-ODM Semantic Schema Editor User’s Guide. High-Performance Database Research Center (HPDRC) documents, School of Computer Science, Florida International University, 2000.

71 B. E. Reddy, P. G. Reddy, and A. Gupta. “A methodology for integration of heterogeneous database.” IEEE Transactions on Knowledge and Data Engineering, 6 (6): 920-933, December 1994.

72 A. Sheth and V. Kashyap. “So far (schematically) yet so near (semantically).” Interoperable Database Systems, pages 283--312, Elsevier Science Publishers B. V. (North-Holland) 1993.

73 A. Sheth and J. Larson. “Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Database.” ACM Computer Surveys, 22 (3):183-235, Sept. 1990.

74 A. Levy, A. Rajaraman, and J. J. “Ordille: Querying Heterogeneous Information Sources Using Source Descriptions” In Proceedings of the 22nd VLDB Conference, 251-262, 1996, Bombay, India.

75 W. Litwin and A. Abdellatif. “Multidatabase interoperability.” IEEE Computer, pp. 10-18, December 1986.

76 W. Litwin and A. Abdellatif. “An overview of the multi-database manipulation language MDSL.” Proceedings of the IEEE, 75(5):621-632, May 1987.

77 R. Ahmed, et al. “The Pegasus heterogeneous multidatabase system.” IEEE Computer, 24 (12):19-27, December 1991.

78 C. Collet, M. N. Huhns, and W. Shen. “Resource integration using a large knowledge base in Carnot.” IEEE Computer, pp. 55--62, December 1991.

79 Y. Arens, C. Chee, C. Hsu, and C. Knoblock. “Retrieving and integrating data from multiple information sources.” International Journal on Intelligent and Cooperative Information Systems, 2(2), June 1993.

80 H. Garcia-Molina, et al. “The TSIMMIS approach to mediation: data models and languages.” In Next Generation Information Technologies and Systems, June 1995. Nahria, Israel.

81 Y. Papakonstantinou, S. Abiteboul, and H. Garcia-Molina. “Object fusion in mediator systems.” In Proceedings of the 22nd VLDB Conference, 1996, Bombay, India. pp. 413-424.

82 A. Segev. “Strategies for distributed query optimization.” Information Sciences, 54 (1-2), Mar. 1992.

83 N. Rishe, J. Yuan, J. Meng, S. Chen, and O. Wolfson. “On Database Integration over Intelligent Networks.” Proceedings of the ISCA 2nd International Conference on Information Reuse and Integration (IRI-2000), November 1-3, 2000, Honolulu, Hawaii, USA. pp. 70-75.

84 N. Rishe. “Managing network resources for efficient, reliable information systems.” Proceedings of the Third International Conference on Parallel and Distributed Information Systems, p.223-6, Austin, TX, USA, Sept. 1994.

85 W. Kim, et al. “On Resolving Schematic Heterogeneity in Multidatabase System.” Distributed and Parallel Databases, Vol.1 No. 1, p.251-279, 1993.

86 N. Rishe, R. Athauda, J. Yuan, S.-C. Chen. “Semantic Relations: The key to integrating and query processing in heterogeneous databases.” The 4th World Multiconference on Systemics, Cybernetics and Informatics, SCI 2000, Vol. VII, pp. 717-722.

87 M. Zeiler. *Inside ARC/INFO.* 2nd Edition, On Word Press, Santa Fe, NM, 1997.

88 ESRI, *ARC/INFO data management*, ESRI, Inc, New York, NY, 1994.

89 R. Elmasri and S. Navathe. *Fundamentals of Database Systems.* 2nd Edition, Addison-Wesley Publishing Company, Menlo Park, CA, 1994.

90 J. Nievergelt and M. Freeston. “Special Issue Editorial Spatial Data: applications, concepts, techniques.” The Computer Journal, Vol. 37, No. 1, 1994.

91 J. R. Jensen. *Introductory Digital Image Processing - A Remote Sensing Perspective*. 2nd ed. NJ: Prentice Hall, pp139-165, 1996.

92 W. K. Pratt. *Digital Image Processing.* 2nd ed. New York: Wiley, 698 pp., 1991.

93 A. K. Jain. *Fundamentals of Digital Image Processing.* Englewood Cliffs, NJ, Prentice Hall, pp.342-357, 1989.

94 J.A. Richards. *Remote Sensing Digital Image Analysis.* New York: Springer -Verlag, p281, 1986.

95 B. Jahne. *Digital Image Processing*. New York: Springer-Verlag, p383, 1991.

96 Space Systems/Loral Operations Ground Equipment (OGE) Interface Specification, DRL 504-02-1 Part 1, Palo Alto, CA, 1995.

97 U.S. Fish and Wildlife Service, available at http://www.fws.gov.

98 S.L. Pimm, J.L. Lockwood, C.N. Jenkins, J.L. Curnutt, M.P. Nott, R.D. Powell, and O.L. Bass, Jr. “Sparrow in the Grass. A report on the first ten years of research on the Cape Sable Seaside Sparrow (Ammodramus maritimus mirabilis).” Internal Report, Everglades National Park, 2002, 182 pp.

99 South Florida Ecosystem Office, available at http://www.sfnrc.ever.nps.gov.

100 United States Geological Survey, available at http://www.usgs.gov.

101 Oracle Corporation, available at http://www.oracle.com.

102 Sparrow database web application, available at http://www.sfnrc.ever.nps.gov/pls/sparrow/data_sparrow_forms.

103 SPSS Inc., available at http://www.spss.com.

104 N. Rishe, O. Dyganova, A. Selivonenko, M. Chekmasov, and A. Mendoza. "MedFerret: Client-Based Semantic Query Integrator." In Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001), Orlando, FL., Vol I, pp. 66-70. July 22-25, 2001.

105 N. Rishe, T. Berk, O. Dyganova, A. Selivonenko, S. Graham, and D. Mendez. "Crawling Distributed Operating System within Multiple Heterogeneous Operating Systems of Internet-connected Hosts and Devices." In Proceedings of the 5th World Multiconference on Systemics, Cybernetics and Informatics (SCI 2001), Orlando, FL., Vol I, pp. 71-76. July 22-25, 2001.

106 S.-C. Chen, N. Rishe, X. Wang, and A. Weiss. "A High-Performance Web-Based System Design for Spatial Data Access." In Eighth Symposium of ACM GIS, Washington D.C., pp. 33-38, November 10-11, 2000.

107 TerraFly: A Web-Enabled Application for Visualization and Manipulation of Remotely Sensed Data, available at http://terrafly.fiu.edu/tf-whitepaper.pdf.

108 IBM AIX: UNIX operating system - an open UNIX solution, available at http://www.ibm.com/servers/aix.

109 Intel Corporation, available at http://www.intel.com.

110 The Windows 2000 Operating Systems, available at http://www.microsoft.com/windows2000.

111 Space Imaging -- Visual Information. Visible Results, available at http://www.spaceimaging.com.

112 Advanced Micro Devices, AMD, available at http://www.amd.com.

113 Linux Online Inc., available at http://www.linux.org.

114 Squid Web Proxy Cache, available at http://www.squid-cache.org.

115 The FreeBSD Project, available at http://www.freebsd.org.

116 Spong - Systems and Network Monitoring, available at http://spong.sourceforge.net.

117 American Power Conversion Corp., available at http://www.apcc.com.

118 Winbond Hardware Doctor, available at http://www.intel.com/support/motherboards/server/sai2/hardware_dr.htm.

119 The Geographic Names Information System, http://gnis.usgs.gov.

120 N. Rishe, M. Chekmasov, M. Chekmasova, S. Graham, and I. De Felipe. "On-Demand Geo-Referenced TerraFly Data Miner." Proceedings of the 2003 International Conference on Intelligent User Interfaces. pp. 277-279.

121 Florida Memorial College, http://www.fmc.edu.

122 Dell Computer Corporation, http://www.dell.com.

123 Intel Xeon Processor, http://www.intel.com/products/server/processors/server/xeon.

124 Microsoft Corporation, http://microsoft.com.

125 US Census Bureau, United States Department of Commerce. Available at http://www.census.gov/.

126 Google search engine. Available at http://www.google.com.

127 Federal Emergency Management Agency. Available at http://www.fema.gov/.

128 Fire detection program, National Oceanic and Atmospheric Administration. Available at http://nhis7.wwb.noaa.gov/website/SSDFire/viewer.htm.

129 N. Rishe. Data extractor, U.S. Patent 6339773. Issued January 15, 2002.

130 A. Shaposhnikov, and N. Rishe. "S-tree - a High Performance Multidimensional Spatial Index." HPDRC internal report, May 2002.

131 G.L. Steele. Common LISP the Language, Thinking Machines, Inc. Digital Press, 1990, 1029 pp.

132 Y. Sun. "Manual of TerraFly LISP." Technical Report, Florida International University, 2000.

133 American Frontiers: A Public Lands Journey. Available at http://americanfrontiers.net/.

134 D. Davis-Chu, N. Prabakar, N. Rishe, and A. Selivonenko. “A System for Continuous, Real-Time Search and Retrieval of Georeferenced Objects.” Proceedings of the 2nd International Conference on Information Reuse and Integration, Honolulu, Hawaii, pp. 82-85, 1-3 November 2000.

135 A. Selivonenko, N. Prabakar, N. Rishe, and D. Davis-Chu. “Dynamic Mosaicking of Heterogeneous Digital Images.” Proceedings of the ISCA 2ndInternational Conference on Information Reuse and Integration, Honolulu, Hawaii. pp. 86-90, 1-3 November 2000.

136 The Federal Aviation Administration, available at http://www.faa.gov.

137 Navigation and Planning Services, available at http://www.navplanning.com.

138 Jeppesen - Making Every Mission Possible, available at http://www.jeppesen.com.

139 Maptech: Topo Maps Charts Navigation Software GPS and Online Mapserver, available at http://maptech.com.

140 Destination Direct - Flight Planning, available at http://www.destdirect.com.

141 Data Transformation Corp., available at http://www.duat.com.

142 AccuWeather - The World's Weather Authority - International & Local Weather Forecasts, available at http://www.accuweather.com.

143 National Oceanic & Atmospheric Administration, available at http://www.noaa.gov.

144 International Air Transport Association, available at http://www.iata.org.

145 N.Rishe, M.Chekmasov, A.Selivonenko, Y.Sun, S.Graham, and A.Mendoza. “Autopilot Scripting for the TerraFly GIS.” Proceedings of the 9th International Conference on Distributed Multimedia Systems, Miami, USA, 24-26 September, 2003.

VITA

SCOTT C. GRAHAM

1989	B.S.E. (with honors), Computer and Information Engineering Sciences, (minor in Mathematics), University of Florida, Gainesville, Florida
1989-1995	Research / Teaching Assistant, School of Computer Science, Florida International University, Miami, Florida
1992	M.S., Computer Science, Florida International University, Miami, Florida
1995-	1995-1996, Research Associate; 1996-, Senior Research Associate / Assistant Director, High Performance Database Research Center, Florida International University, Miami, Florida

SELECTED PUBLICATIONS

D. Beryoza, N. Rishe, S. Graham, I. De Felipe. "Data Extractor Wrapper System." 6th World Multiconference on Systemics, Cybernetics and Informatics SCI-2002, Orlando, USA, July 14-18, 2002. pp. VII-425 - VII-431.

T. Ho, E. Alvarez, N. Prabhakaran, D. Barton, N. Rishe, S. Graham. "Integration of a GIS and a Semantic Database System." First International Conference Geospatial Information in Agriculture and Forestry, Lake Buena Vista, FL June 1-3, 1998, pp. I-629 - I-636.

N. Rishe, D. Barton, N. Prabakaran, M. Gutierrez, M. Martinez, R. Athauda, A. Gonzalez, S. Graham. "Landsat Viewer: A Tool to Create Color Composite Images of Landsat The Matic Mapper Data." International Conference Geospatial Information in Agriculture and Forestry, Lake Buena Vista, FL June 1-3, 1998. pp.I-529 - I-536.

N. Rishe, M. Chekmasov, M. Chekmasova, S. Graham, I. De Felipe. "On-demand Geo-referenced TerraFly Data Miner." 2003 International Conference on Intelligent User Interfaces IUI 2003, Miami, FL, January 12-15, 2003. pp. 277-279.

N. Rishe, M. Chekmasov, R. Rodriguez, S. Graham, D. Mendez. "On the Algorithm for Semantic Wrapping of Relational Databases." 6th World Multiconference on Systemics, Cybernetics and Informatics SCI-2002, Orlando, USA, July 14-18, 2002. pp. VII-436 - VII-440.

N. Rishe, S.C. Chen, A. Mendoza, A. Selivonenko, O. Dyganova, S. Graham. "GIS inter-protocol bridge: GIS vector and raster imagery inter-process wrapper" Seventh International Conference on Remote Sensing for Marine and Coastal Environments, 20-22 May 2002, Miami. pp. F5.1-F5.7.

N. Rishe, S. Graham. "Database Query Distribution over Intelligent Networks." Air Force Research Laboratory Information Directorate Final Technical Report, AFRL-IF-RS-TR-2001-65, 2001. 132 pp.
N. Rishe, T. Huang, M. Chekmasov, S. Graham, L. Yang, S. Himmelsbach. "Semantic Wrapping Tool For Internet." 6th World Multiconference on Systemics, Cybernetics and Informatics SCI-2002, Orlando, USA, July 14-18, 2002. pp. VII-441 - VII-445.
N. Rishe, A. Selivonenko, M. Chekmasov, S. Graham, M. Gutierrez, B. Wongsaroj, O. Dyganova, A. Mendoza. "Overview of TerraFly GIS Hardware Architecture." Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics SCI-2003, Orlando, USA, July 27-30, 2003. Vol. XVI, pp. 11-16.
N. Rishe, A. Shaposhnikov, S. Graham. "Load Balancing in a Massively Parallel Semantic Database." Computer Systems Science and Engineering (Special issue on massively parallel processing). (v. 11, n. 4, July 1996). pp. 195-199.
N. Rishe, A. Shaposhnikov, S. Graham, G. Palacios. "Implementation of Security in Semantic Binary Object Database." 6th World Multiconference on Systemics, Cybernetics and Informatics SCI-2002, Orlando, USA, July 14-18, 2002. pp. VII-446 - VII-449.
N. Rishe, E. Ugboma, M. Chekmasov, M. Chekmasova, S. Graham, I. De Felipe. "Cape Sable Seaside Sparrow Environmental Database." Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics SCI-2003, Orlando, USA, July 27-30, 2003. Vol. XVI, pp. 17-21.
N. Rishe, O. Wolfson, S. Graham, W. Sun. "Database Query Distribution over Intelligent Networks." Proceedings of the Workshop on Database Technologies at the Beginning of the New Millennium, SoftCOM 2001, October 09-12, 2001, Croatia. Volume 1, pp. 109-116.
N. Rishe, B. Wongsaroj, M. Chekmasov, A. Selivonenko, S. Graham, M. Gutierrez, O. Dyganova, W. Pinckney, E. Padron, C. Armstrong, A. Marshall. "Replication and Maintenance of TerraFly GIS." Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics SCI-2003, Orlando, USA, July 27-30, 2003. Vol. XVI, pp. 6-10.
N. Rishe, B. Wongsaroj, M. Chekmasov, A. Selivonenko, Y. Sun, S. Graham, A. Mendoza. "Autopilot Scripting for the TerraFly GIS." Proceedings of the Ninth International Conference on Distributed Multimedia Systems (DMS 2003), Miami, FL, September 24-26, 2003. pp. 98-100.
N. Rishe, L. Yang, M. Chekmasov, M. Chekmasova, S. Graham, A. Roque. "Mapping from XML DTD to Semantic Schema." 6th World Multiconference on Systemics, Cybernetics and Informatics SCI-2002, Orlando, USA, July 14-18, 2002. pp. VII-450 - VII-455.
O. Wolfson, N. Rishe, S. Graham, W. Sun. "Database Integration over Hybrid Networks." Proceedings of the 2001 International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2001. October 9-12, 2001, Croatia. Volume 2, pp. 935-942.
L. Yang, N. Rishe, M. Chekmasov, S. Graham, I. De Felipe. "XML Schema Modeling Using Sem-ODM." Proceedings of the Ninth International Conference on Distributed Multimedia Systems (DMS 2003), Miami, FL, September 24-26, 2003. pp. 187-192.