POLEMON:
A Federated Database Management System for the Documentation, Management and Promotion of Cultural Heritage

Ch. Bekiari
Institute of Computer Science, Foundation for Research and Technology-Hellas.
P.O. Box 1385. 71100. Heraklion, Crete, Greece.

Ch. Gritzapi
Institute of Computer Science, Foundation for Research and Technology-Hellas,
P.O. Box 1385. 71100. Heraklion, Crete, Greece
E-mail: gritzapi@ics.forth.gr

D.Kalomoirakis
Greek Ministry of Culture, Athens, Greece

Introduction

The Greek Archeological Service, one of the oldest, public services in Greece, was organized from the very beginning to be decentralized. The peripheral units, called Ephorates of Antiquities, are now approaching 55, in number and include a number of large , independent Museums as well as some special services (Underwater Archaeology, Speleology etc.,) are responsible for the field work, excavations, restorations, protection and management of archeological sites and monuments, and their environments.

A special Directive at The General Directorate of Antiquities and Restoration, called the Directorate of Monuments, Records, and Publications, was established in 1977. It collects any useful documents, concerning sites and monuments, compiles a general inventory, and classifies the historical archives, which span the 160 years of works by Greek Archeological Service.

The peripheral units may have their own database management system, to manage their monument data and they give permission to researchers, by request, to access this data.

The dispersed and isolated cultural databases, along the Ephorates, may have the same logical shema, although their vendors and types may be varied, and they may concern different types of monuments.

POLEMON's aim was the creation of a decentralized, management information system for the National Monuments Record, together with an integrated museum information system, for implementation at the national level. This project, entitled "Coordinated Informatics Services for the Documentation, Management, and Promotion of Cultural Heritage" has been finished. This was part of the EPET II program, administered by The (Greek) General Secretariat for Research and Technology. The POLEMON project was carried out by a consortium, headed by the Institute of Computer Science, Foundation for Research and Technology, Hellas (FORTH). Along with the Directorate of Monuments, Records, and Publications, the consortium included the faculty of Rural and Survey Engineering, at the Aristotle University of Thessaloniki, Epsilon Software, S.A., the Institute of Computer and Communication Systems, Intrasoft, S.A., and the Benaki Museum.

POLEMON was designed to cope with monuments of all types, be supported by mapping facilities, and be capable of supporting a wide range of administrative tasks. The latter include making compulsory purchase orders, classifying and declassifying archaeological sites, laying down protection zones, controlling illicit trade in antiquities, planning monument restoration works, etc.

The basic technical requirements and constraints for POLEMON were:

a) The management of a very large volume of data. A museum has, under its responsibility and control, tens or hundreds of thousands' of objects.
b) The nature and frequency of transactions. Data entry was massive and labor intensive. The main use of the system was to retrieve information from remote, eventually heterogeneous, management systems (like relational databases - knowledge bases, relational databases of different vendors) for administrative and research purposes, and to perform administrative functions locally or remotely.
c) Prior systems, already installed in the cultural organizations, defined the initial conditions and constituted a set of knowledge, methods, and systems.

To address the geographical distribution, a wide area network, of interconnected, local area networks, was implemented. Each museum, or unit, that belonged to The Ministry of Culture had its own local area network and application systems, for administrative documentation.

For the implementation of the local, administrative documentation systems, relational database management systems were used, supporting SQL.

These systems were efficient in the storage and retrieval of very large volumes of formatted data, and provided efficient support for administrative operations. Also, the systems cooperated in a federated environment, having significant

autonomy in their execution. Their participation in a federation indicates that they could execute user requests, that accessed the multiple databases of the federation.

One of the main components of The National Monuments Record system is the Global Access System. The feature that makes POLEMON stand out, from a purely technical point of view, is the way in which the Global Access System, in The National Monuments Record system, is built. This provides the foundation, on which a federated database system, can be put together effectively. This system allows the user to define and execute queries, based on a global view, over the federation. The Global Access System is based on SIS, an object-oriented, semantic network database system, developed by FORTH, in the last 5 years, and contains the federated schema, which is an integration of multiple export schemes. It also includes the equivalencies, between federated schema and export schemes, and information on data location, for driving a query generator from federated to export schemes.

In this paper, we present the architecture of the Global Access System, and a methodology for translating an SQL - like query, expressed in a global schema, to SQL queries, expressed in specific databases. The proposed methodology covers schema integration and uses a three-layered, schema architecture. Each layer represents an integrated view of the concepts, that characterize the layer below.

The POLEMON project

In this section, we give a review of The National Monuments Record system, built in POLEMON project.

The main structural and functional features of this system are as follows:

(1) It relates to fixed sites and moveable monuments, their interrelationships and chronological contexts.

(2) It supports administrative documentation of monuments and objects.

(3) It is geographically dispersed, in line with the official administrative plan and distribution of departments, within The Greek Ministry of Culture. Archival information is collected and undergoes specialist processing, at the local level, while The Directorate of Monuments Records, and Publications, within The Ministry of Culture, is responsible for planning and co-ordination, and retrieves data, mainly for administrative purposes.

(4) It is in line with the latest, Greek legislation on Archaeology.

(5) Its architecture is that of a federated database system.

(6) It is connected to a system, which supports mapping documentation and related functions.

(7) It can be linked to domain-specific, cultural documentation databases, which are compatible with those of the integrated museum information system.

(8) It can be linked to the museum information system.

Cartographic support within the system includes the following features:

(9) Infrastructure for drawing up The National Monuments Survey, using Geographical Information Systems (GIS).

(10) Land registry documentation, to cover areas in the vicinity of monuments (mainly archaeological sites), within the framework of the geodesic and administrative reference grid, of the National Land Register.

(11) Necessary requirements for the topographic recording of monuments.

The structure of local stations, in The National Monuments Record, system is illustrated in **Figure 1**.

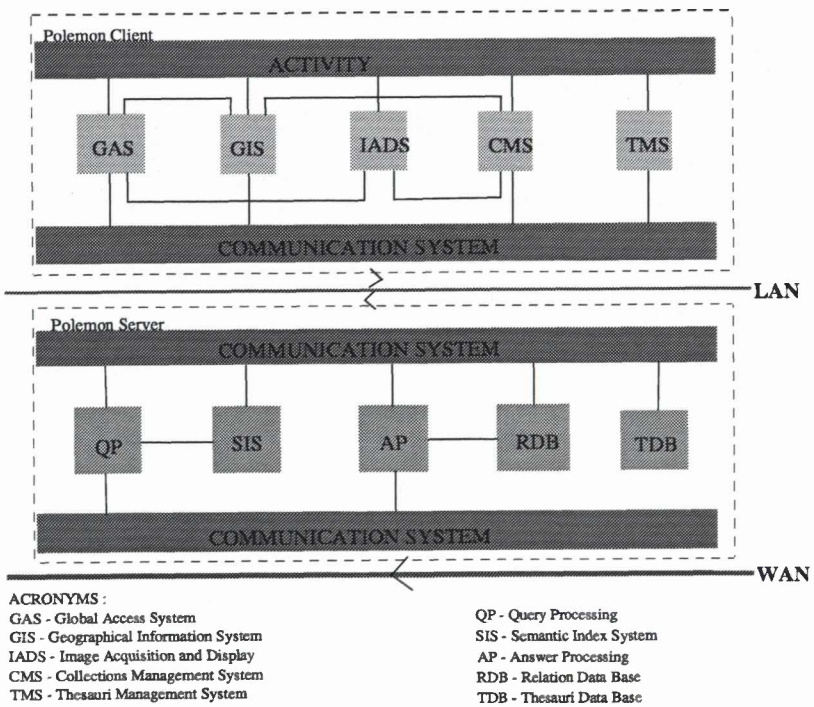

Figure 1. Architecture of the National Monuments Record system It consists of the following major integrated subsystems:

(12) Administrative documentation system
(13) Global access system
(14) Cartographic system
(15) Image acquisition system
(16) Communications interface
(17) Authority service

(1) The administrative system is based on relational data bases. All units have highly similar data schemata and contain administrative data (e.g., museum codes, names, types, administrative and general states etc., designs and cartographic data (connections with specialized data bases)), and may possibly contain images and text (which can be stored separately).
(2) The global access system is based on SIS, and contains the federated schema, which is an integration of mappings of multiple export schemes. It also includes the equivalencies, between federated schema and mappings, and information on data location, for driving a query generator from federated to external schemes.
(3) The cartographic system refers to the use of computer graphic technologies for the organization, processing, storage, display, and presentation of the geometric, spatial, architectural and structural characteristics of monuments, archeological sites, and other relative information (location, usage, bas-reliefs). The image acquisition system deals with image and text acquisition, and processing.
(4) The Communications interface of the network handles functions, such as query formulation, and query processing from one network station to another, answer receipt and transmission between network stations, data transfer control(especially for multimedia data), and periodic data dictionary updates. One of the system units is responsible for the administration of the network, using special programs for network management.
(5) The Authority service system creates, maintains and accessed (browsing and direct querying) formal knowledge structures for reference in, and index of, cultural data.

The National Monuments Record system was installed in a network of stations, with nodes in Komotini, Thessaloniki, Athens and Heraklion (see Figure 2). Following this, experience gained in the pilot stage will be used in the planned extension of the network, so as to serve all divisions of The Ministry of Culture, nationwide.

Figure 2. Polemon's pilot network

The National Monuments Record System's federation architecture

The architecture of The National Monuments Record system's architecture is the architecture of a federated, database management system (FDBMS) (Amit, 1990). It consistes of cooperating, but autonomous, component database management systems, which support monuments of various types. The component DBMS can differ, in many aspects, such as data models, query languages, etc. The database management systems of a FDBMS may be characterized, along three orthogonal dimensions: distribution, heterogeneity, and autonomy.

Distribution

Data may be distributed among multiple databases. These databases may be stored on a single computer system or on multiple computer systems, which are geographically distributed, but interconnected, by a communications system. There are two basic reasons for the data distribution in The National Monuments Record System. The first is the prior database management systems that existed in The Ephorates of Antiquities, before the beginning of the POLEMON project, such as DELTOS[1] and FEIDIAS[2]. The second reason is the geographical distribution and administrative structure of The Ephorates of Antiquities, of the Greek Archeological Service.

Heterogeneity

The types of heterogeneity, in most of the database systems, can be divided into those differences, due to the differences in DBMS, and those due to the differences in the semantics of the data. In more detail, we have :

a) Differences in structure, where we can have differences in data models, constraints or query languages.
b) Differences in the hardware of the database management systems, where we can have differences in synchronization, retrieval, etc.
c) Differences in the semantics of the data.

The (a) and (b) types of heterogeneity arise in The National Monuments Record System, due to the different database management systems, being used by the Ephorates. For example, the 1st Ephorate of Historic and Prehistoric Monuments uses the INGRES RDBMS(in FEIDIAS); the 23rd Ephorate of Historic and Prehistoric Monuments uses the SYBASE RDBMS (in DELTOS). The third type arises, due to the different logical schemes which the geographically dispersed, local databases may have.

For example, the content of the database, of the 13th Ephorate of Byzantine and post-Byzantine antiquities,

[1] DELTOS is a database management system for monuments and preserved buildings and it has been developed by the Institute of Computer Science , Foundation for Research and Technology - Hellas (FORTH) and the Archeological Museum of Crete-Hellas.
[2] FEIDIAS is a database management system for movable monuments and it has been developed by the Archeological Museum of Acropolis in Athens-Hellas.

concerns site monuments and preserved buildings of Crete, while the content of the database, of 23rd Ephorate of Historic and Prehistoric Monuments, concerns ancient sites and immovable monuments, of historic and prehistoric periods of Crete; and, the content of the database, of the 1st Ephorate of Historic and Prehistoric Monuments, concerns movable objects, especially statues, offering objects, etc. from Athens

Autonomy

A component DBS, participating in a FDBS, may exhibit several types of autonomy. Some of these types are :

a) *Design autonomy*: refers to the ability of a component DBS to choose its own design, including the data being managed, the representation and the naming of the data elements, the semantic interpretation of the data, the constraints used to manage the data, and the functionality of the system, or the implementation of the system.
b) *Communications autonomy*: refers to the ability of a component DBS to choose whether to communicate with other component DBMSs.
c) *Execution autonomy*: refers to the ability of a component DBS to execute its local operations, without interfering with the external operations, which are executed from the other component DBMS, of the federation. The component DBS can abort any external operation, that does not meet its local constraints, without affecting its local operations. This means that each, of the POLEMON's DBS, is independent from the other.
d) *Association autonomy*: refers to the ability of its component DBS to freely decide whether it wants to associate, or disassociate itself from the federation. This means that POLEMON's federation has been designed, so that its existence and operation are not dependent on any single component DBS.

POLEMON's federation is based on a three-level schema architecture, that addresses the requirements of dealing with the above dimensions, shown in Figure 3.

This architecture includes the following components :

- **Local Schema :** A local schema is the logical schema of a component DBS, expressed in the data model of the federated schema..
- **Export Schema:** An export schema is a subset of a local schema, that is available to the FDBS. This happens, because not all the data of a component DBS may be available to the users of the federation. In our approach, the export schema is declared implicitly, by defining mappings from the global schema to local schemes.
- **Federated Schema:** A federated schema is an integration of multiple, export schemes. This schema also includes the information of data interconnection, that is generated when integrating the export schemes. In POLEMON, this schema is called, *global schema*, and it is declared in a data model, which is capable of expressing a variety of local schemes and mappings, between export schemes and global schema. The global schema consists of three parts:
  - The first part concerns the integration of all of the export schemes. In this paper, we will not give a formal method for this integration. The problem of integrating dissimilar database schemes has been investigated by a number of researchers, and many prototypes have resulted (Batini & Lenzerini, 1984; Navathe & Gadgil, 1982; Navathe, *et al.*, 1984; Motro, 1987; Dayal & Hwang, 1984; Batini, *et al.*, 1986). In the section for *Mapping Definition*, we present a couple of problems that arise during this integration.
  - The second part concerns the user's entry points. They are constituted by a tree of labels, defining a
  - meaningful view, for the final user, helping him to construct queries to the federated schema.
  - The third part include the mappings, between export schemes and federated schema.

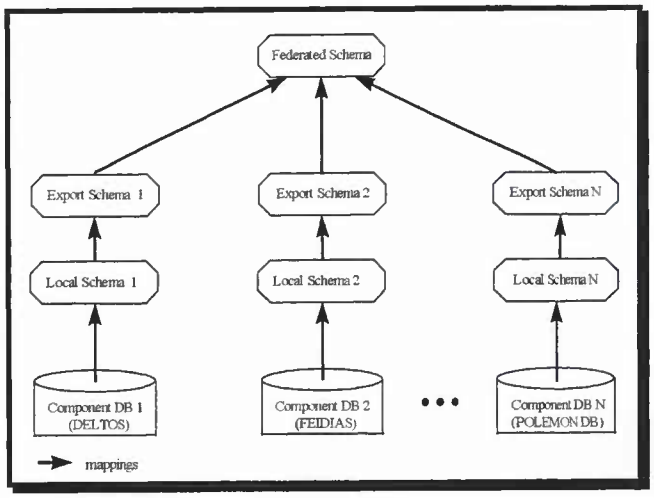

Figure 3.

We use a knowledge representation language and SIS (the Semantic – network, object-oriented database management system) to declare and store the above-mentioned schemes. SIS is a tool for describing and documenting large, evolving varieties of highly interrelated data, concepts, and complex relationships. SIS consists of a persistent, storage mechanism, and generic, interactive user- and program-level interfaces. SIS offers significantly richer, referencing mechanisms than relational, or ordinary, object-oriented systems, offering a very high query speed, for references.

A semantic network usually is depicted as a graph, consisting of nodes and links. Nodes represent entities, that are elements in an enterprise, under certain interest. A node can represent an integer, or a string, or a more complex element such as a car, or even an abstract element, such as a bank account.

Links, in a semantic network, represent any type of connection between entities and are usually called attributes or relationships. A link can represent a connection, between a person and his name, or a person and his father, or even, a set and its elements.

We have defined a semantic model (see Figure 4) for all of the above information.

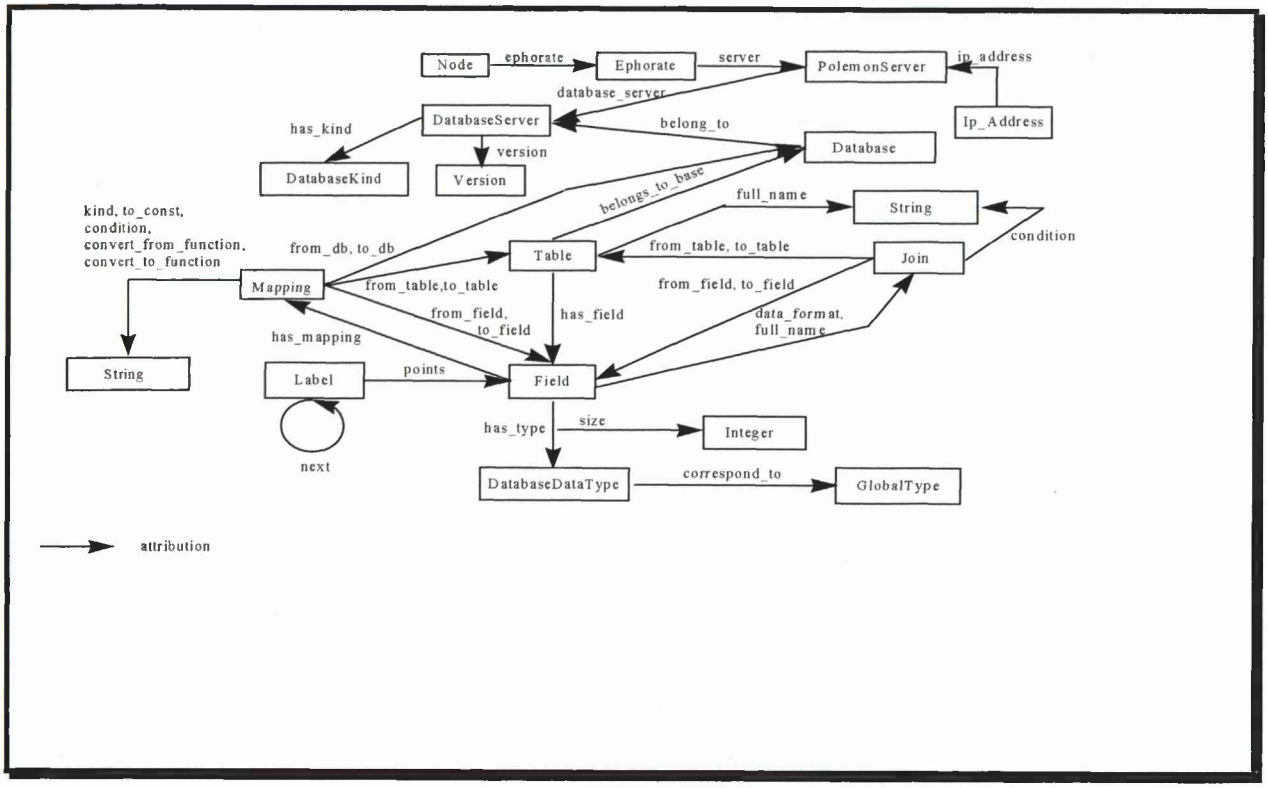

Figure 4. Semantic Model of a global access system database

This model contains classes and attributes for denoting a database scheme, such as the tables and fields of each database, all of the correspondence between database export schema and global schema, and all of the information concerning database servers, database kinds and database distribution, existing in every node of the Polemon network. Basic entities of the above, Figure 4, are the following:

- **Node**
  this entity refers to a geographical location, where the Ephorates of Antiquities exist.
- **Ephorate**
  this entity describes all of the Ephorates of Antiquities. Each of them can have one or more Polemon Server.
- **PolemonServer**
  this entity refers to existing Polemon servers. Each server has an attribute, to declare the unique *ip_address* (entity *Ip_Address*), and may communicate with one or more database servers.
- **DatabaseServer**
  this entity represents the Database servers (DB -server). Each DB-server has an attribute, to declare the *version* e.g., SYBASE v.10) and the *kind* (e.g. SYBASE) and it supports one, or more database.
  of the database *PolemonCrete*, which communicates with a *polemon server*, having *ip_address, 139.91.183.25*, and belongs to a *database server*, with the name *SYBASE10.* We use the following syntax:
- **Database**
  this entity represents the Databases, which participate in the federation of The National Monuments Record System. Each database can participate in this federation, with one or more tables.
- **Table**
  this entity represents the tables of one database.
- **Field**
  this entity represents the fields of one table. Each field has a *data type* (entity *DatabaseDataType*), which is assigned to one of the *Global Types* (*Integer, Float, Char, Image, Date Time*, etc.), used for results integration. Finally, a field can participate in one or more joins.

An example follows, showing the use of the above information.

Suppose that we want to insert, into the global access system database, a part of a local schema for the following tables:

Table name	***movable_objects***	
Field names	*field types*	*field description*
code	varchar(20)	is the monument's code
published	char(1)	a flag declaring if the monument's information has been published or not
name	varchar(50)	monument's name or a brief description
kind_code	int(4)	monument's kind code and it is a foreign key to the table *kinds*
weight	float(8)	monument's dimensions.
dimension	varchar(60)	monument's dimensions.
Table name	***kinds***	
Field names	*field types*	*field description*
kind_code	int(4)	monument's kind code
kind_name	varchar(50)	monument kind's name

Schema Declaration Example
NODE 139.91.183.25
DATABASE polemonCrete@SYBASE10
DB_TYPE SYBASE
TABLE movable_objects ***HAS***
code : varchar(20) ***DATA_FORMAT***(AAKM),
published : char(1) ***DATA_FORMAT***(D),
name : varchar(50) ***DATA_FORMAT***(D),
kind_code : int(4) ***DATA_FORMAT***(D),
weight : float(8) ***DATA_FORMAT***(D),
dimensions : varchar(60) ***DATA_FORMAT***(D)
TABLE kinds ***HAS***
kind_code : int(4) ***DATA_FORMAT***(D),
kind_name : varchar(50) ***DATA_FORMAT***(D)
FROM TABLE movable_objects
FROM FIELD kind_code
TO TABLE kinds
TO FIELD kind_name
WHERE movable_objects.kind_code = kinds. kind_code

In figure 5, we can see a screendump, with the entry points to the Polemon Global Access Database.

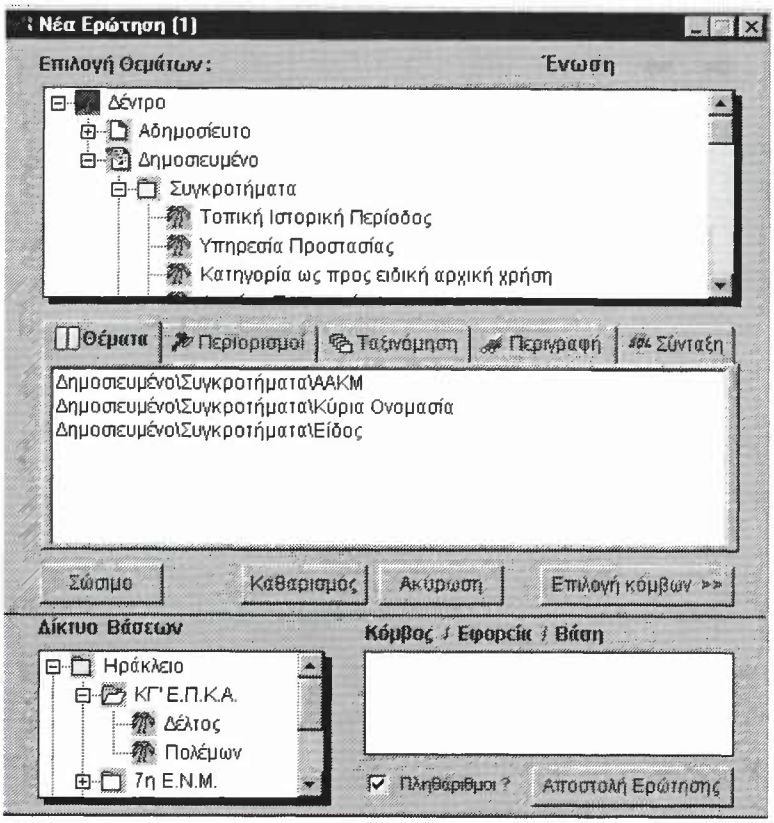

Figure 5.

Finally, we should mention that this tree is very easy to change, or to enrich.

Mapping's Definition

The cooperation between autonomous and already existing databases, in order to share their data, while at the same time maintaining their autonomy, has been the subject of many works, and many approaches exist.

In such a heterogeneous environment, the user cannot be expected to be familiar with every detail of each database of the federation. For this reason, the FDBMS must give the user the ability to have a uniform and integrated access to the data, of all database-components. The problem of uniform and integrated access, to the data of database-components, is known as *data sharing* (Amit, 1990). There are three levels of *data sharing:*

a) Transport of all data from database A to another database, B.
b) Creation of an intermediate schema and transport of all data under this schema.
c) Creation of an intermediate schema as an integration of all the component-database schemas, and development of appropriate mechanisms, which allow the user to pose questions, to the intermediate schema, and get answers from the component-databases.

One of the most important problems in this area is the translation of a schema, S1, expressed in model M1, to a schema, S2, expressed in another model, M2. Closely connected to this problem is the discovery of a mapping, between the two different models. The real problem, in all situations, is to be decided if the two schemas, or parts of them, are equivalent, and many problems can arise in such translations (Batini, et al., 1986; Castellanos, 1993; Chen, 1976; Lien, 1982; Castellanos, et al., 1994; Miller, et al., 1994). Some of these problems can be classified, according to the bibliography, as follows:

- *Naming Conflicts*: Objects in different schemes, representing the same real world concepts, may have dissimilar names, resulting in problems of two types:
  - Homonyms, where the same name is used for two different concepts, and
  - Synonyms, where the same concept is found, with more than one name.
- *Type Conflicts*: one object is represented by different constructs in different schemes; for example, one schema is represented as an attribute in one table, and in another as a constant value.
- *Key Conflicts*: different keys are assigned to the same concept, in different schemes.
- *Behavioral Conflicts*: these conflicts concern the different policies that may be used during the insertion/deletion operations, for the same concepts, in different schemes.
- *Missing Data*: different attributes may be assigned to the same concept in different schemes.
- *Levels of Abstraction*: this conflict arises when the same concept is stored in dissimilar levels of detail in different databases.
- *Identification of Related Concepts*: this conflict arises, when concepts exist in the component schemes, that are

- not the same, but are related; in these cases, the user is required to discover all the inter-schema properties, that relate to them.
- *Scaling Conflicts*: these conflicts arise, when the same concept is stored in dissimilar units of detail, in different databases (for example, the weight of an object in one database is stored in Kg, and in another, it is stored in gr).
- *Multiplicity Conflicts*: these conflicts arise, when the relationship between two concepts can take multiple values in one schema, but only one value in another schema.

In POLEMON's implementation, we followed the third kind of the pre-mentioned case of data sharing, because it was the most appropriate for our goals. During this implementation, we faced Naming, Type, Key, Missing Data and Multiplicity conflicts. In the next sections, we will present examples of these conflicts and the solutions we adopted.

In order to store these mappings in the SIS, we defined a semantic model, shown in Figure 6. This model has been described in the S_Class level, and, in summary, contains the following entities and the relationships, among them.

The **Mapping** entity represents the notion of a mapping and can be described as follows:

* Each mapping has an attribute *kind* which declares its kind. We distinguish two main kinds of mappings: FIELD_TO_FIELD and FIELD_TO_CONST.
* *from_db* and *to_db*, are attributes declaring the from-part and the to-part of a database correspondence.
* *from_table* and *to_table*, are attributes declaring the from-part and the to-part of a table correspondence.
* *from_field* and *to_field*, are attributes declaring the from- part and the to-part of a field correspondence. In some cases, a mapping doesn't correspond to a field, but to one or more constants (*to_const*).
* *condition*, is an attribute denoting the semantics, which should be added to the WHERE part of the translated, SQL query.
* *convert_from* and *convert_to function*, are atributes declaring attached functions to a mapping, used in cases with scaling conflicts (not implemented yet).

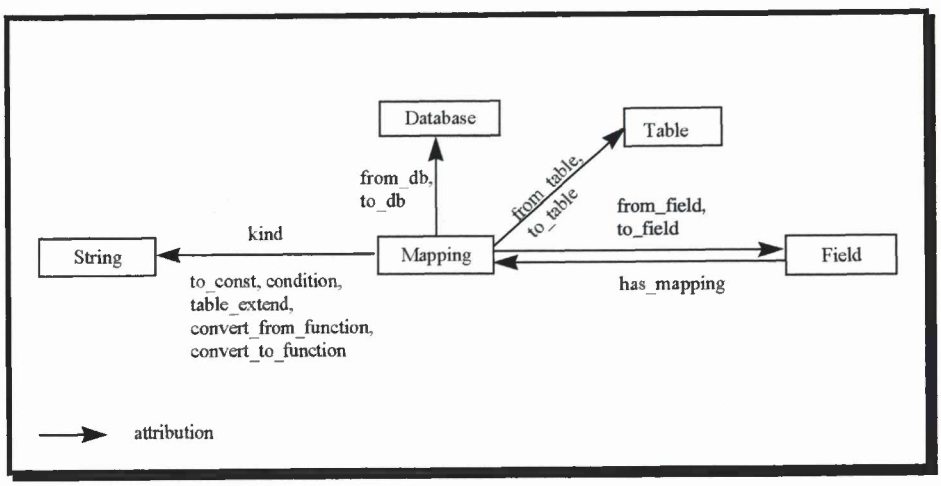

Figure 6.

In the rest of the paper, we will use the following syntax to describe the two kinds of mappings.

<mapping's kind>

from_table	:	<table name>
from_field	:	< table name>.<field name> [,< table name>.<field name>]*
to_table	:	<table name>
to_field/to_const	:	< table name>.<field name> [,< table name>.<field name>]*/< CONSTANT >
condition	:	[< table name>.<field name> = < table name>.<field name> | CONSTANT [*AND* <table name>.<field name> = < table name>.<field name> | CONSTANT]*]

Mapping FIELD_TO_FIELD

This mapping can be applied to solve naming, key, missing data, and multiplicity conflicts. In the following examples we illustrate how this mapping is applied, in each of the above cases.

Naming Conflicts

Naming conflicts arise when objects in different schemes, representing the same real world concepts, have dissimilar names, resulting in problems of two types(homonyms and synonyms). Here, we present an example with synonyms.

Suppose that we have the notion of movable objects, and each movable object has, along with its other attributes, one attribute which stands for its name. Also, suppose that in the global schema, the movable objects are represented by the table, **kinita**, described as follows:

kinita			
Field	Type		Comment
aakm	varchar(20)	not null	monument's code
onomasia	varchar(50)	null	monument's name

In database FEIDIAS, the same notion is represented by the table **directory**, described as follows:

directory			
Field	Type		Comment
dirno	varchar(12)	not null	monument's code
title	varchar(100)	null	monument's name

We want that whenever we send a query, expressed in global schema and asking for the values of the field *kinita.onomasia* to the FEIDIAS database, to be able to retrieve the values of the field *directory.title*. In order to achieve this, we have to declare the mapping, FIELD_TO_FIELD, as follows:

FIELD_TO_FIELD
FROM_TABLE : kinita
FROM_FIELD :onomasia
TO_TABLE : directory
TO_FIELD :title

Missing Data Conflicts

Missing data conflicts arise, when different attributes may be assigned to the same concept, in different schemes. Here we present a problem of missing data, which concerns the place of an object's discovery.

Suppose that the notion of the place of discovery, for one movable object in the GLOBAL schema, is represented by the table **topos_aneyresis_kinita**, described as follows:

topos_aneyresis_kinita			
Field	Type		Comment
aakm	varchar(20)	not null	monument's code
paratiriseis	varchar(50)	null	comments about place of discovery

While at the same time, in database FEIDIAS, this notion is represented in the table **directory** by the fields:

directory			
Field	Type		Comment
dirno	varchar(12)	not null	monument's code
datofdiscy	int	null	year of discovery
datofdiscm	int	null	month of discovery
datofdiscd	int	null	date of discovery
plaofdisc	varchar(25)	null	place of discovery
wayofdisc	varchar(25)	null	way of discovery

Here we note the following:

a. The GLOBAL scheme has less information than the FEIDIAS scheme, about the discovery of a monument.
b. we want that whenever we send a question, expressed in GLOBAL schema asking for the values of the field *topos_aneyresis_kinita paratiriseis*, to the FEIDIAS database to be able to get the values of the field, and to take as a result, the values of the fields *directory.title, directory.datofdiscy, directory.datofdiscm, directory.datofdiscd, directory.plaofdisc,* and *directory.wayofdisc.*

In order to achieve this, we must use the mapping, FIELD_TO_FIELD, as follows:

FIELD_TO_FIELD
FROM_TABLE : topos_aneyr_kinita
FROM_FIELD :paratiriseis
TO_TABLE : directory
TO_FIELD : plaofdisc,
wayofdisc,datofdiscd,datofdiscm,datofdiscy

Key Conflicts

Key conflicts arise when different keys are assigned to the same concept in different schemes. In POLEMON's federation, we detected the following situation:

In GLOBAL scheme, the monument's key is an alphanumeric field (20-characters long), and is called, *aakm*, and it is produced by the concatenation of several parts, such as:

* the global kind of monument (movable, immovable, coin)
* the kind of Ephorate of Antiquities, to which the monument belongs (Historic, Prehistoric, etc.)
* the prefecture's code, where the monument is kept, and, finally
* the monument's insertion number in the local database.

In DELTOS database, the monument's key is a numeric field, called *AAD*, and it has the monument's registration number, in the local database.

And finally, in FEIDIAS database, the monument's key is called, *dirno*, and it is an alphanumeric field (12-characters long), which the user gives, during the insertion transaction.

To overcome this conflict, we declared one mapping for each field, associated with the monument's key from each table (which had this field) of the GLOBAL scheme, to each table (which also had this field) of the DELTOS or FEIDIAS

database. For example, suppose that in the GLOBAL scheme, the movable objects were represented by the table, **kinita**, described as follows:

kinita			
Field	Type		Comment
aakm	varchar(20)	not null	monument's code
...	...	...	...

And that, in database FEIDIAS, the same notion was represented by the table **directory**, described as follows:

Directory			
Field	Type		Comment
dirno	varchar(12)	not null	monument's code
...	...	...	...

We want that whenever we send a question, asking for *kinita.aakm*, expressed in global scheme to the FEIDIAS database, to take as a result, the values of the field *directory.dirno*. In order to achieve this, we have to use the mapping, FIELD_TO_FIELD, as follows:

FIELD_TO_FIELD
FROM_TABLE : kinita
FROM_FIELD :aakm
TO_TABLE : directory
TO_FIELD :dirno

Multiplicity Conflicts

Multiplicity conflicts arise when the relationship between two concepts can take multiple values, in one schema, but only one, in another schema.

Suppose that we have the notion of immovable objects, and each one of them has, along with its other attributes, one attribute which stands for the kind of monument. This attribute is unique. Also, suppose that in the GLOBALscheme, the immovable objects are represented by the table, **akinita**, described as follows:

akinita			
Field	Type		Comment
aakm	varchar(20)	not null	monument's code
kwd_eidoys	varchar(30)	null	monument's kind
...	...	...	...

Each immovable object, in DELTOS database can have multiple kinds, which are stored in the table **EIDOS_OBJ**, which is described as follows:

EIDOS_OBJ			
Field	Type		Comment
AAD	int	not null	monument's code
akinito_eidos	varchar(100)	null	monument's kind

We want that whenever we send a query, asking for the values of the field, *akinita.kwd_eidoys*, expressed in GLOBAL scheme of the DELTOS database, to take as a result, the values of the field *EIDOS_OBJ.akinito_eidos*. In order to achieve this, we use the mapping, FIELD_TO_FIELD, as follows:

FIELD_TO_FIELD
FROM_TABLE : akinita
FROM_FIELD : kwd_eidoys
TO_TABLE : EIDOS_OBJ
TO_FIELD : akinito_eidos

Mapping FIELD_TO_CONST

This mapping can be applied to solve missing data and type conflicts.

In order to see how this mapping works in each case we should study the following examples.

Type conflicts

Type conflicts arise when one object is represented by different constructs, in different schemes; for example, one schema is represented as an attribute in one table, and in another, as a constant value.

Suppose that we have the notion of movable objects, and each movable object has, along with its other attributes, one attribute which stands for the name of the Ephorate of Antiquities, to which it belongs. Also, suppose that in the GLOBAL scheme, the movable objects are represented by the table, **kinita**, described as follows:

kinita			
Field	Type		Comment
aakm	varchar(20)	not null	monument's code
ypiresia_prostasias	varchar(50)	null	monument's Ephorate of Antiquities

In database FEIDIAS, there is no field for Ephorate's, name, because all the monuments have the same Ephorate, which is named, "A' Ephorate of Historic and pre Historic Monuments" :

We want that whenever we send a query, asking for the values of the field, *kinita. ypiresia_prostasias*, expressed in GLOBAL scheme to the FEIDIAS database, to take as a result, the value "A' Ephorate of Historic and pre Historic Monuments". In order to achieve this, we use the mapping, FIELD_TO_CONST:

FIELD_TO_CONST
FROM_TABLE : kinita
FROM_FIELD : ypiresia_prostasias
TO_CONST : "A'EPKA"

A Query Translation Example

In this section, we give an example of the whole process. We describe how the user formulates his query, how this query is translated from a Global scheme to different local schemes, and how the user retrieves the results of his query.

Suppose that a user constructs a query asking for :

* monument's code ,
* monument's name,
* monument's general kind,
* monument's material ,
* monument's main chronology, and
* name and surname of the archaeologist, who registered the monument data for all the immovable objects, which are *"Temple"* or *"TEMPLE"*.

What the user should do, is select the subjects and describe the constraints for his query. He can do this by using the screens in Figure 7.

In the screen shown in Figure 7, we can see the tree of subjects. The user can select, from this area, the subjects and the constraints for his query. What he selects, appears in the area, called *"Selected Subjects"*; he can then add, delete, and sort (area *"Sort Subjects"*) his selections; also, he can save (button *"Save Query"*), or cancel (button *"Cancel Query"*), or see the description of (button *"Query's description")* his query. When the user finishes the construction of the query, he selects the databases in which he prefers his query to be executed.

In our example, the user selects the DELTOS database, of the Archeological Museum of Heraklion, and the Polemon Crete database, of the 13th Ephorate of Byzantine and Post-Byzantine. When he presses the *"Send Query"* button, the query's translation phase starts.

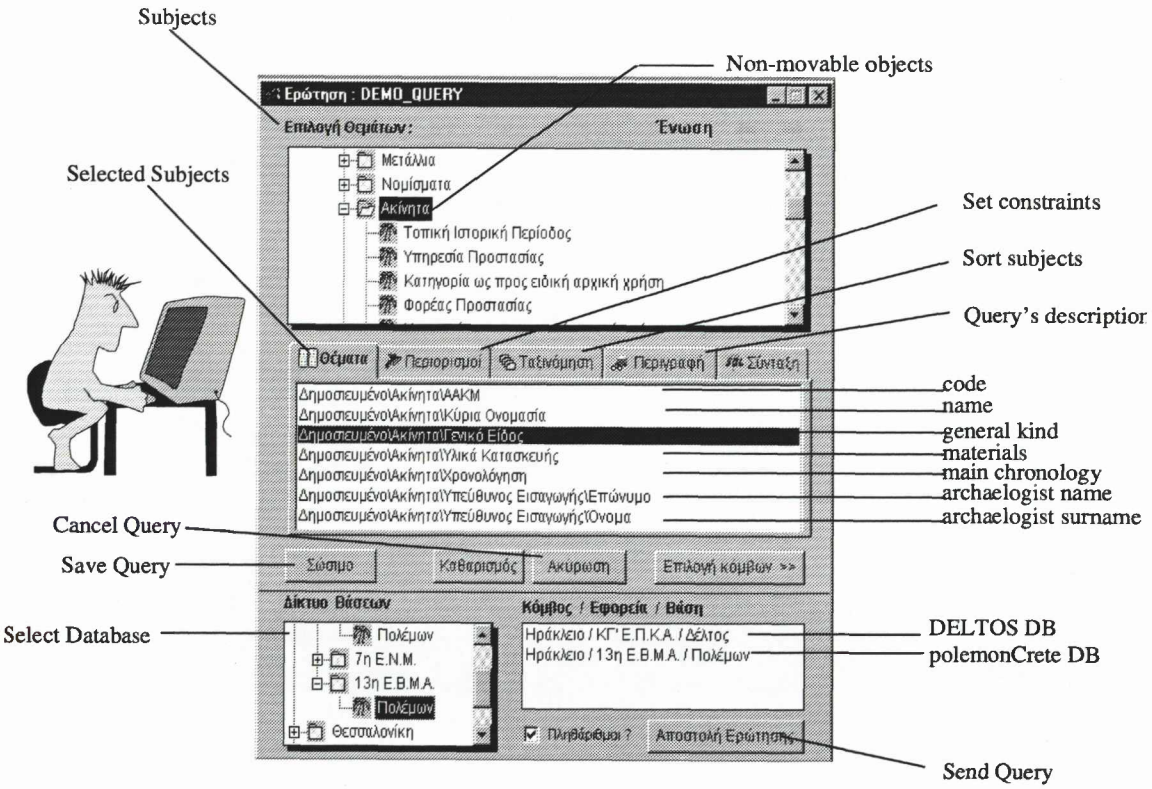

Figure 7

The following table presents the form of the prescribed query:

Demo Query		
Subjects	:	Object[3].Code, Object.Name, Object.GeneralKind, Object.Material, Object..MainChronology, Object.ArcheologistName, Object.ArcheologistSurname
Constraints	:	Object.Kind = "Temple" OR Object.Kind = "TEMPLE"

After applying the mappings, declared in the federated scheme, the above query is expressed in the schemes, of each one of the target databases. In our example, the resulting queries are shown in the ***DELTOS's table*** and ***PolemonCrete's table***:

[3] Object stands for immovable object

DELTO's table

Keyword	*Field*	*Mapping*	*Remarks*
SELECT	KEY.AAD,	F_F[4]	code
	FASEIS_OBJ.ONOMASIA,	F_F	name
	"Á",	F_C[5]	general kind
	----		materials
	FASEIS_OBJ.XR_PER_APO,	F_F	main chronology
	FASEIS_OBJ.XR_PER_EWS,	F_F	-//-
	EPWNYMA.EPWNYMO,	F_F	arch. Name
	ONOMATA.ONOMA,	F_F	arch. Surname
FROM	KEY , FASEIS_OBJ , EPWNYMA,		
	ELEGTES , ONOMATA , EIDOS		
WHERE	(EIDOS.EIDOS LIKE "Íáüò%"	F_F	kind="Temple"
	OR		or
	EIDOS.EIDOS LIKE "ÍÁÏÓ%") AND		kind = "TEMLE"
	FASEIS_OBJ.KYRIA= 'Í' AND	F_F	
	FASEIS_OBJ.EIDOS_KWD =EIDOS.EIDOS_KWD AND		
	FASEIS_OBJ.KYRIA= 'Í' AND		the rest of
	KEY.AAD=FASEIS_OBJ.AAD AND		the joins among
	KEY.AAD=ELEGTES.AAD and		the query's tables
	ELEGTES.EPWN_KWD=EPWNYMA.EPWN_KWD and		
	ELEGTES.ONOMA_KWD=ONOMATA.ONOMA_KWD		

Polemon Crete's table

Keyword	*Field*	*Mapping*	*Remarks*
SELECT	akinita.aakm,	F_F	code
	akinita.onomasia,	F_F	name
	akinita.gen_eidos,	F_F	general kind
	eidi_ylikoy.yliko,	F_F	materials
	akinita.xronologisi,	F_F	main chronology
	atomo.onoma,	F_F	arch. Name
	atomo.epwnymo	F_F	arch. Surname
FROM	akinita , eidi_ylikoy , atomo,		
	eidi_mnimeioy , mnimeia_ylika		
WHERE	(eidi_mnimeioy.eidos LIKE "Íáüò%"	F_F	kind="Temple"
	OR		or
	eidi_mnimeioy.eidos LIKE "ÍÁÏÓ%") AND		kind = "TEMLE"
	akinita.kwd_eidoys =eidi_mnimeioy.kwd AND	F_F	
	akinita.gen_eidos ="Á" AND		the rest of
	akinita.dimosieymeno ="Í" AND		the joins among
	mnimeia_ylika.kwd_ylikoy =eidi_ylikoy.kwd AND		the query's tables
	akinita.aakm =mnimeia_ylika.aakm AND		
	atomo.kwd_atomoy =akinita.kwd_ypeyth_eisagwgis		

The translation process is invisible to the user.

Just after the translation of the query, a *Waiting Results Window* appears on the user's screen (see Figure 8). In this window, the user sees how many tuples satisfy the query, for each of the target databases.

The user can see the results, by selecting the *Confirmation Button* on the previous window. The results appear on his screen, which is shown in Figure 9. The results window is divided into five main areas:

* *Database* , the name of the database, where the data come from
* *Monument's Code*, monument's code in this database
* *Image*, denotes if this monument has an image, or not

[4] F_F stands for FIELD_TO_FIELD mapping
[5] F_C stands for FIELD_TO_CONST mapping

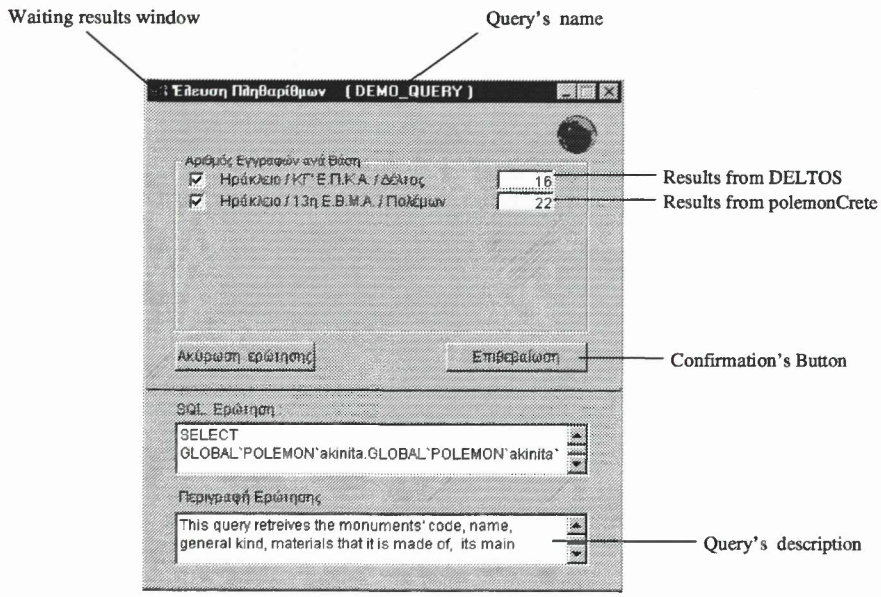

Figure 8

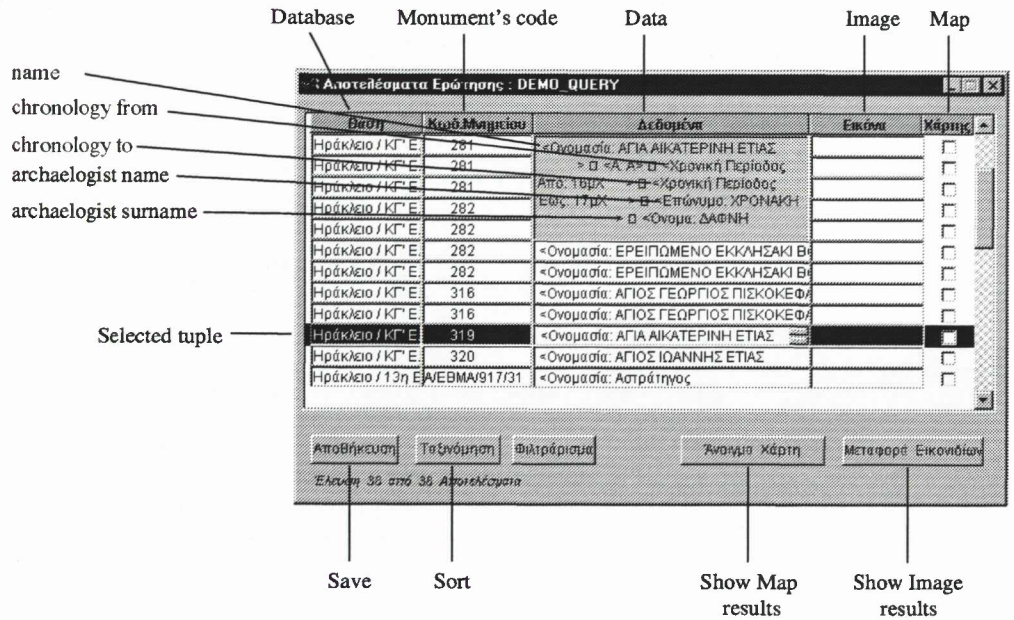

Figure 9

* *Map*, denotes if this monument has map information, or not, and
    * *Data*, all the rest of the data, that the user has asked for, are presented in this area.

The user can save (button "*Save*") or sort (button "*Sort*") the results of his query.

Comparing the results (Figure 9 and Figure 10), coming from the DELTOS and PolemonCrete databases, we see the different structures of the tuples.

Conclusions

In this paper, we have presented two types of mappings, between heterogeneous databases. These mappings have been applied in the POLEMON project, and have been successful in most of the cases encountered, although these are not enough, for solving any kind of conflict arising, during the integration of schemes, from two or more heterogeneous databases.
Several problems need further research and development. Some of them need immediate answers, within Polemon's project scope, such as:

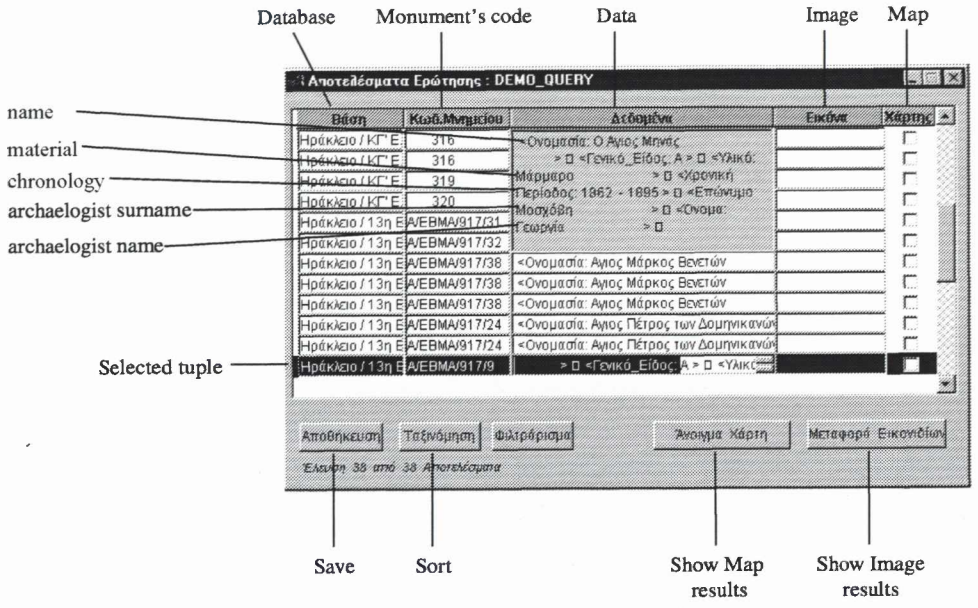

Figure 10

a. The investigation, design and implementation of further mappings.
b. the lack of adequate, transaction management algorithms, that provide a specified level of consistency (i.e., are correct with respect to a given consistency criteria) and fault tolerance with acceptable performance, within the heterogeneity and autonomy constraints of an FDBS.
c. a system for identifying and representing all semantics, useful in performing various FDBS tasks, such as schema translation and schema integration, and for determining contents of schemes, at various levels.

Bibliography

BATINI C. and LENZERINI M. (1984), "A methodology for database schema integration in the entity relationship model", *IEEE Transactions Software Engineering*, vol.SE-10, no. 6.

BATINI C., LENZERINI M. and NAVATHE S.B. (1986), "A comparative analysis of methodologies for database schema integration", *ACM Computing Surveys*, vol. 18, no. 4, pp:323-364.

CASTELLANOS M., (1993), "A Methodology for Semantically Enriching Interoperable Databases", *Proceedings, 11th British National Conference on Databases*, Keele, pp.58-75.

CASTELLANOS M., SALTOR F.and GARSIA-SALACO M. (1994), *an* "Semantically Enriching Relational Databases into Object-Oriented Semantic Model", in: KARAGIANNIS D. (ed.), *Database and Expert Systems Applications (5th International Conference DEXA'94, Athens*, Springer Verlag, pp. 125-134.

CHEN P.P. (1976), "The entity-relationship model - toward a unified view of data", *ACM Transactions on Databases Systems* , vol.1(1), pp.9-36.

DAYAL U. and HWANG H. (1984), "View definition and generalization for database integration in MULTIBASE : A system for heterogeneous distributed databases", *IEEE Transactions Software Engineering*, vol.SE-10, no. 6, 1.

HSIAO D.K. (1991), "Federated Databases and Systems : Part I - A Tutorial on Their Data Sharing", *VLDB Journal*,179, January, pp. 127.

LIEN E.Y. (1982), "On the Equivalence of Database Models", *Journal of the ACM*, vol.29(2), pp. 333-362.

LITWIN W., Mark L. and ROUSSOPOULOS N. (1990), "Interoperability of Multiple Autonomous Databases", *ACM Computing Surveys*, vol.22(3), pp. 267-293.

MANNAI D. N. and BUGRARA K. (1993), "Enhancing Inter-Operability and Data Sharing In Medical Information Systems", *Proceedings of the ACM SIGMOD International Conference on Management of Data*, vol.22, Washington, pp: 495-498.

MILLER R.J., IOANNIDIS Y.E. and RAMAKRISHNAN R. (1994), "Schema equivalence in heterogeneous systems : Bridging theory and practice", *Information Systems*, vol.19(1), pp. 3-31.

MOTRO A. (1987), "Superviews : Virtual integration of multiple databases", *IEEE Transactions Software Engineering*, vol.SE-13, no. 7, July 1987.

NAVATHE S.B. and GADGIL S.G. (1982), "A methodology for view integration in logical database design", in: *Proc. Eighth Int. Conf. Very Large Databases.*

NAVATHE S.B., SACHIDHAR T. and ELMASRI R. (1984), "Relashionship merging in schema integration", in: *Proc. Tenth Int. Conf. Very Large Databases.*

SHETH A. P. and LARSON J.A. (1990), "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys*, 22(3), pp.183-236.