Towards Interoperability in Heterogeneous Database Systems

A. Zisman J. Kramer

Imperial College Research Report No. DOC 95/11

Department of Computing
Imperial College
180 Queen's Gate, London SW7 2BZ - UK
Email: az@doc.ic.ac.uk, jk@doc.ic.ac.uk

December, 1995

Abstract

Distributed heterogeneous databases consist of systems which differ physically and logically, containing different data models and data manipulation languages. Although these databases are independently created and administered they must cooperate and interoperate. Users need to access and manipulate data from several databases and applications may require data from a wide variety of independent databases. Therefore, a new system architecture is required to manipulate and manage distinct and multiple databases, in a transparent way, while preserving their autonomy.

This report contains an extensive survey on heterogeneous databases, analysing and comparing the different aspects, concepts and approaches related to the topic. It introduces an architecture to support interoperability among heterogeneous database systems. The architecture avoids the use of a centralised structure to assist in the different phases of the interoperability process. It aims to support scalability, and to assure privacy and confidentiality of the data. The proposed architecture allows the databases to decide when to participate in the system, what type of data to share and with which other databases, thereby preserving their autonomy. The report also describes an approach to information discovery in the proposed architecture, without using any centralised structure as repositories and dictionaries, and broadcasting to all databases. It attempts to reduce the number of databases searched and to preserve the privacy of the shared data. The main idea is to visit a database that either contains the requested data or knows about another database that possible contains this data.

Keywords

Heterogeneous databases, global schema, interoperability, federation, information discovery, scalability, shared data, semantic information, autonomy, transparency.

Contents

Chapter 1

Introduction

1.1 Motivation

Before database systems appeared it was very difficult to share heterogeneous files created through multiple autonomous applications. It was a complex task to manage all of these files in a single application. They had a great number of differences, such as field naming, value types and file structures. To resolve these problems and difficulties the use of autonomous files were changed by the use of a central collection of data named *database*, managed by a centralised control system called *database system*.

During the seventies, centralised databases were widely used and after some years a similar problem appeared, but in a different environment. The development of database management systems has increased the utility of these systems, but has not solved the problem of having a great number of databases in a large company or community. Users need to access and manipulate data from several databases and applications may require data from a wide variety of independent databases. These databases are independently created and administered, differing physically and logically. Each independent database has its own schema, expressed in its own data model, accessed by its own retrieval language and designed by one or a group of persons.

The development of distributed computing and networking has provided the technical basis for remote access. However, the situation requires for a new system architecture designed to manipulate and manage different and multiple databases. We are faced with the challenge of trying to simultaneously manipulate different databases, without losing their autonomy and in a way that is transparent to the users and applications, that is, make them interoperable.

Kamel and Kamel [56] outlined some of the requirements and objectives that must be fulfilled and provided when interoperating with heterogeneous databases, such as:

1. Distributed transparency, that is, the users must access a number of different databases in the same way as accessing a single database.

2. Heterogeneity transparency; users must access other schemas in the same way they access their local database (using a familiar model and language).

3. The existing database systems and applications must not be changed.

4. Addition of new databases must be easily accommodated into the system.

5. The databases have to be accessed both for retrievals and updates.

6. The performance of heterogeneous systems has to be comparable to the one of homogeneous distributed systems.

Although the area related to heterogeneous databases has been hardly studied and various approaches proposed, it still asking for new solutions and approaches. One important issue when dealing with interoperability of heterogeneous databases is the discover of relevant data and where it is located. Most of the existing approaches in the literature assume that the relevant data is either already known or identified and try to process it efficiently. However, in a n environment with a large number of databases this assumption is not reasonable. The information discovery process is related to the fact that, during the resolution of a query, the system does not know the location of the correct information and which are the components that contain this information.

Another drawback when dealing with heterogeneous databases is concerned to the resolution of conflicts and representation of semantic aspects. It is very important to understand, capture and deal with the semantics of the data in order to permit the detection of the correct data and the assignment of conflicts that may appear. The unification of data is also a significant question. It is necessary to execute the unification as automatic as possible, preserving the databases autonomy and without interfering in the original data stored in the involved databases. On the other hand, transaction management is a complex task in the environment of heterogeneous databases. It is very difficult to guarantee the ACID and serializability properties because each system has its own mechanism to ensure the properties. The interdependencies among data in different databases have to be carefully studied. It is essential to allow consistency during the exchange and share of data. Thereby, it is necessary to find a way of representing, manipulating and maintaining interdependencies.

1.2 Goals and Contributions

Based on the problems of interoperating with heterogeneous databases and on the existing approaches we propose a new way of dealing with these systems. We suggest the use of an architecture to support interoperability of autonomous heterogeneous databases. The architecture permits the transparent execution of both read and write operations in remote databases, in the same way as manipulating local databases, without compromising their autonomy. It avoids the use of a centralised structure to assist in the different phases of the interoperability process (resource discovery, resolution of semantic heterogeneity and unification of remote and local data). The architecture also assures privacy and confidentiality of the data and supports scalability of the system, allowing component databases to join and leave the system. With this architecture the databases can decide when to participate in the system, what type of data to share and with which other databases. Thereby, preserving their autonomy.

We also propose a distributed algorithm to perform information discovery. This process is executed neither using a centralised structure containing the relationship and information about the shared data and its location, nor performing broadcast to all databases in he system. Each component database has structures containing information about other databases that possible contain requested data. In order to help in the discovery

process, we suggest the use of special structures possessing semantic information of the shared data. These structures are local for each component.

The approach also permits a dynamic evolution of the system by allowing databases to join and leave the system. We demonstrate the scalability property of the system by the specification of different cases. With the proposed approach it is possible to have the participation of databases that share and exchange data and databases that only consult and interrogate the other members of the system.

1.3 Definitions

There is a variety of definitions in the literature to describe terms like: *distributed databases, heterogeneous databases, federated databases, multidatabases* and *interoperable systems*. To avoid confusion and to facilitate the comprehension of the text we briefly describe these terms.

According to Ceri and Pelagatti [23], *distributed databases* are considered as a collection of data distributed over different computers of a computer network. Each node of the network has autonomous capability, performs local applications and may participate in the execution of some global application that requires accessing data at several sites, using the communication subsystem.

Heimbigner and McLeod [48] classified databases along two dimensions: (a) conceptual/logical structure and (b) physical organisation and structure. Each of these dimensions can be either centralised or decentralised. Therefore, we can identify four classes of databases: (1) logically centralised and physically centralised (this class includes the traditional integrated databases); (2) logically centralised and physically decentralised (including distributed databases[1] and some approaches to compose databases support); (3) logically decentralised and physically centralised; or (4) logically decentralised and physically decentralised. In a logically centralised system there is a single central conceptual schema, where in a logically decentralised one there are various distinct conceptual schemas. A physically centralised system contains the data stored in the same computer, where physically decentralised databases have data stored in many separate computers.

Database systems can also be classified as *homogeneous* or *heterogeneous*. Like [48, 91] we consider as homogeneous the databases where the local schemas and the global schema are defined using the same data model and data manipulation language. On the other hand, heterogeneous databases contain different data models and data manipulation languages[2].

We use the terms *component* and *database* interchangeable. These terms mean a database sharing and exchanging data. The terms *local database* or *local component* are used to denote a database with its original characteristics. That is, a database with its attributes before joining and participating in a system of heterogeneous databases.

The term *federated database system*, introduced by Hammer and McLeod [44] and extended by Heimbigner and McLeod [48], is a federation of loosely coupled databases without a global schema. That is, autonomous component database systems that coop-

[1] This is the way in which the term has been largely used in the literature.

[2] For Thomas et al. [104] a database is *heterogeneous* if the local nodes have different types of computers, operating systems, communication links and protocols, and database management systems, even if all local databases are based on the same data model.

erate with each other. The federated architecture has to resolve two conflicting requirements: (a) the maintenance, as far as possible, of the autonomy of the components, and (b) the achievement of a reasonable degree of information sharing. In this architecture each component contains three different schemas (private, export and import). The exchange of data is executed by performing negotiations among the involved databases (see subsubsection 2.3.1).

Multidatabases and *interoperable databases* are both considered here in the same way as defined in [68, 69, 71]. Like the federated databases they are also multiple autonomous databases, managed together without a global schema. The difference consists in the architecture. The multidatabase is basically a set of databases (heterogeneous) containing a multidatabase language (universal or common language), that allows the manipulation of the databases[3] (see subsubsection 2.3.1).

Unless otherwise specified, during the text we use the term *federation* for a group of autonomous heterogeneous databases, sharing and exchanging data, independently of the architecture in use. Hence, it is possible to have a federation of federated databases, a federation of multidatabases, a federation of interoperable databases, and so on.

It is also important to define what is meant by *transparency* and *autonomy* when handling with different databases. Transparency is the ability to access a data without knowing that this data is distributed over different heterogeneous databases. For the user and application the system is seen as a traditional centralised database. The data is accessed in the same way that the user access his/her database, using the same data model and language. The system has to be able to automatically propagate the information throughout the components of the system when necessary.

Autonomy refers to the capacity of a component to choose its own design and operational model. An autonomous component is able to: (a) define the data that it wants to share with other components, (b) define the representation and naming of the data elements, (c) determine the way that it views the data and combines existing data, (d) decide when to enter or leave a federation, (e) specify whether to communicate with other components, (f) execute local operations without interference from external operations, (g) add new data and withdraw access to any shared data.

1.4 Report Outline

The remainder of this report is organised as follow.

Chapter 2 contains the literature survey analysing and comparing the different aspects, concepts and approaches related to the area of heterogeneous databases. We divide the existing approaches that deal and manipulate with different autonomous databases into two groups. In the first group, a *global schema* is used to integrate the databases. The second group does not use a global schema making the databases *interoperable*. Thereby,

[3]Sheth and Larson [91] used the term federated database systems to describe systems with broader capabilities. For them a federated database system is a collection of cooperating autonomous component database systems. They categorised federated database systems into *loosely* or *tightly coupled*, based on who manages the federation and how the components are integrated. A *loosely coupled federated database system* is what is called in [68, 69, 71] multidatabases or interoperable database systems, and in [44, 48] by federated database system. A *tightly coupled federated database system* permits the existence of a global schema.

we present important concepts that appear in both group of approaches, such as: semantic and syntactic conflicts, choice of a canonical data model and the integration process. Following, we describe the characteristics of these groups, together with some approaches. Finally, we briefly present the interdependencies and transaction management characteristics.

In chapter 3 we propose an architecture to support interoperability among heterogeneous databases. The architecture permits the transparent execution of both read and write operations in remote databases, in the same way as manipulating local databases, without compromising their autonomy. It avoids the use of a centralised structure to assist in the different phases of the interoperability process, assures privacy and confidentiality of the data, and supports scalability of the system. To guarantee autonomy and transparency new structures are added to each database before joining the system. These additional structures provide the information necessary to achieve interoperability, without modifying the original structure of a database.

Chapter 4 introduces how to initialise a system of heterogeneous databases using the proposed architecture. A very important issue when dealing with heterogeneous databases is to permit a dynamic evolution of the system. We claim that the proposed architecture permits this evolution without suspending the execution of the whole system and preserving the autonomy of the databases. We present the scalability property of the architecture by identifying and describing all possible cases. This description consists in the specification of how to perform the evolution of the system.

Chapter 5 describes a process to execute information discovery in the proposed architecture. Most of the existing approaches in the literature assume that the relevant data is either already known or identified. The discovery process has the characteristic of not using any centralised structure as repositories and dictionaries, containing information about the shared data and its location. It also avoids broadcasting to all databases. To execute the information discovery process we propose the use of a hierarchical structure, local for each database. This structure contains information about the location of different databases by classifying them in groups. These groups are specified depending on the type of shared data of the databases.

Chapter 6 investigates the use of a special structure (SPD) containing semantic information about the shared data. The semantic of the shared data is important in all the different phases of the interoperability process. The focus of our study related to the semantic aspects of the shared data is to help on the information discovery process. This chapter is not complete. It is necessary to specify more details of how to represent and manipulate with the semantic characteristics.

In chapter 7 we present an example (case study) were we analyse the architecture, the

discovery process and the scalability property. This example is useful to illustrate all the proposed ideas.

Chapter 8 presents a summary and a critical review of the research findings. It also describes the conclusions and suggests some topics for further work.

Chapter 2

Related Work

This chapter contains a literature survey of the work that has been done to support the manipulation and management of different heterogeneous database systems.

We divide the current approaches in the literature that deal with the problem of manipulating and managing different and multiple autonomous databases into two groups. The first group uses the idea of having a *global schema* [1, 11, 14, 23, 26, 30, 62, 67, 91, 97, 104] integrating all the databases in a logical single database. Hence, users may manipulate data of the global schema or of an external schema derived from it. As defined by Litwin [69], it is the reapplication of the database approach principle one level up, i.e., all data has to be converted into a new (distributed) database. However, due to the necessity of preserving autonomy of the databases and the lack of a general solution to resolve the semantic conflicts it is difficult to create a global schema. The second approach assumes that the databases the user may access have no global schema [14, 23, 29, 33, 48, 50, 68, 69, 71, 91, 105]. The existing databases are *interoperated* instead of being separately manipulated or only components of a single global schema.

2.1 Dealing with Heterogeneous Databases

In this section we present some important aspects that appear in both groups of approaches. Following, the next two sections focus on the approaches which use a global schema and the approaches which make the databases interoperable, respectively.

2.1.1 Semantic and Syntactic Conflicts

One fundamental question when dealing with autonomous heterogeneous database systems[1] is related to the resolution of semantic and syntactic conflicts. This has to be addressed either when using the approach of integrating all of the systems in a global schema or when applying a method to interoperate between them, The conflicts appear whenever trying to identify semantically related objects in systems independently created and administered. That is, systems containing different constructs to model the same portion of the Universe of Discourse (UoD). Batini et al. [6] defined *conflicts* as being two (or more) not identical representations of the same concept. As outlined in [6, 29, 62, 108, 114] the

[1] This question is also important when dealing with Intelligent and Cooperative Information Systems [17, 34, 35, 81].

success in schema integration (and database interoperability) depends on understanding the semantics of the schema elements and on the ability to capture and deal with these semantics. Thus, it is necessary to have methods that make explicit the semantics of the different data models [109].

The semantic and syntactic conflicts can be originated by different reasons, such as: (a) the different viewpoints that groups of users and designers have about certain information during the design phase (*different perspectives*); (b) the existence of different kinds of constructs in the data models, permitting different modelling possibilities (*equivalent constructs*); (c) the fact that different design specifications can result in different schemas (*incompatible design specifications*) [6].

The fact that the same concept can be described in different schemas using different representations, generates several types of semantic relationships between these representations. In [6, 40] these relationships are classified as: *identical*, when the representations are exactly the same; *equivalent*, when the representations are not exactly the same, but it is possible to use either behavioural, or mapping, or transformational equivalences; *compatible*, when the representations are neither identical nor equivalent, but the constructs and integrity constraints are not contradictory; and *incompatible*, when the representations are contradictory. The equivalent, compatible and incompatible semantic relationships are defined as *conflict*.

Based on the classifications and frameworks that appear in the literature for the different types of conflicts [6, 14, 30, 40, 56, 59, 62, 99, 100, 108] we classify these types as follow. *Name*, involving homonyms, when two items have the same names but different meanings, and synonyms, when two items have different names but the same meanings. *Scale*, involving the use of different units of measurement. *Structural*, when the same facts are described in two schemas using different elements of the same data model (different choice of modelling constructs or integrity constraints). *Representation*, when the same data item has different representations in the schema. *Different levels of abstraction*, when one schema contains more information details than the other. *Schematic discrepancies*, when data in one database corresponds to metadata in another[2].

Saltor et al. [89] proposed a general solution to schematic discrepancies based on a framework of *generalisation* and *aggregation*. They suggested operations that transform data into metadata and vice-versa, in both relational and object-oriented data models.

In [94] the author developed a semantic taxonomy to characterise semantic similarities among two objects[3]. In this taxonomy, the degree of similarity is denoted by a qualitative measure named *semantic proximity*, that classifies and distinguishes the two objects as: *semantically equivalent, semantically related, semantically relevant* and *semantically resemblant* [92]. Two objects are: (a) semantically equivalent, when they represent the same real world entity or concept; (b) semantically related, when there is a generalisation or aggregation abstraction between the domains of the two objects; (c) semantically related, when there is a context dependency relation between them; and (d) semantically resemblant, when they cannot be related to each other by any abstraction in any context, but they have the same role in their respective context(s). On the other hand, *semantic incompatibility* does not mean the lack of any semantic similarity, it means that the objects are unrelated. This can only be established either when there is no context

[2] In [61] the authors proposed a higher order language for interoperability of databases with schematic discrepancies.

[3] Unless otherwise specify, by the word "object" we mean any construct of a data model.

and no abstraction in which the domains of two objects can be related, or when the objects being compared cannot have similar roles in the context(s) that they exist. This semantic taxonomy was also used to relate a structural taxonomy emphasising schematic (structural/representational) differences among the objects. That is, they identify semantic similarities between objects that have different types of schematic differences such as: incompatible problems of domain, entity definition, data value, abstraction level and schematic discrepancies.

Francalanci and Pernici [40] affirmed that to detect conflicts the first step is to discover *similarity* between schema portions, even when existing different meanings to similarity concepts. After detecting the similar schema portions, the second step consists in the use of transformational and mapping equivalence notions. These notions are used to ensure if the schema conflicts are similar or not. Nevertheless, to execute schema comparison it is necessary to use the experience of the designer and his knowledge about the model, schema and application context (human help).

In order to identify relationships (similarities) among objects and to detect semantic and syntactic conflicts some approaches have been proposed. Chatterjee and Segev [24] proposed a probabilistic technique for comparison of records across databases when the identifying attributes are structural or semantically incompatible. The idea is to compare not only the identifying attributes of two records, but all of the attributes that identify instances and are common to the two records being compared. Using this technique the probability of correct identification is improved. If the two records describe the same real world instance and their identifiers are different, then most of the other common attributes would match. Therefore, if two records sharing the same identifier refer to two different real world instances, then the common attributes are less likely to match. After comparing two records a *comparison value* is assigned to them. Depending on this value it is possible to conclude if these records correspond or not to the same real world instance.

Yu et al. [114, 115] proposed another approach to identify relationships. The idea is to use a global concept space (knowledge base - KS) that permits the specification of common concepts and relationships among unknown terms. The steps to determine relationships consist of: (a) automatic mapping of names to a set of common concepts by their description and creation of new concepts whenever necessary, (b) calculation of the similarity of each pair of names arranging them in descending order of similarity by using similarity functions, (c) conformation by the DBA of the relationships of each pair of related name.

Urban and Wu [108] proposed the use of a semantic data model to describe the structural semantics of the other component data models, that participate in an environment of heterogeneous databases.

The conflict solution consists in modifying either one or both conflicting schemas, in a way that the final representation satisfies the requirements of both original schemas (see [6] and [40] for a survey about the approaches to detect and resolve conflicts).

2.1.2 Choice of a Canonical Data Model

In both groups of approaches it is necessary to choose a canonical data model to permit cooperation and communication between the heterogeneous databases.The canonical data model is used to bridge the gap among the different data models of the local components. It facilitates the detection of interdatabase semantic relationships and provides

transparency to the access of different components. The canonical data model is used either to describe the global schema (see subsection 2.2), or to express the shared data of the involved components (see subsubsections 2.3.1 and 2.3.5), or as an intermediate data model (see subsubsection 2.3.3).

The task of chosen a canonical data model is not simple. The characteristics of a data model are responsible to make a certain model suitable or not suitable for this goal. Many of the data models that exist in the market are used as canonical data models in the existing systems, such as: Entity Relationship (DDTS [23, 91]), Relational (MERMAID [14, 91, 104], DATAPLEX [14, 26, 104], ADDS [14, 104]), Functional (MULTIBASE [14, 23, 91, 97, 104]), Object-Oriented (PEGASUS [1, 57], Heimbigner [14, 48], Hammer et al. [44, 45, 37, 38]), ERC++ (Tari [103]), and so on.

As outlined by Saltor et al. [88] a data model is responsible for the representation ability of a database which is composed of two factors: *expressiveness* and *semantic relativism*. Expressiveness means the degree that a data model can represent the conceptualisation of the reality aspects. It is composed of structural and behavioural parts. Semantic relativism of a database is the degree where a database can accommodate all of the different conceptualisations that distinct persons have about the same aspect of reality. Therefore, semantic relativism of a data model is the power of its operations to derive external schemas from its database schemas. That is, multiplicity of possible representations of a given real world.

The canonical data model must have an expressiveness equal or greater than the data models of the other components in the system. It may allow simple translations (mappings) between the data models of the existing component databases. Due to the possible limited expressiveness of the original data models, sometimes, it is necessary to include structural and behavioural semantic enrichment before translating schemas.

Eliassen and Karlsen [33] affirmed that the adoption of a canonical data model is a requirement for providing data model transparency, since it can permit uniform access to the heterogeneous databases (homogenisation). They believe that an o-o data model is suitable as a canonical data model. At the federation level it is necessary a strong notion of identity (the property that distinguishes an object from all other objects in the system), and object-oriented data models support this identity notion.

Saltor et al. [88] developed a framework to analyse what are the characteristics that a data model should have to be suitable to be used as a canonical data model. They conclude that the hierarchical and network models are not suitable. On the other hand, the ER model and its extensions are inferior when compared to the existing o-o models. However, in their opinion, the functional and some o-o data models supporting views seam to be the best data models for this task.

Tari [103] affirmed that an object-orientation of a canonical data model is generally advised. The relational model is not suitable, since it does not contain the necessary semantics for defining all essential mappings. In his opinion the object-oriented models have some advantages such as: complex objects, shareability, abstraction hierarchy, and object identity. However, the o-o models fail when representing explicit information of databases applications as: (a) representation and manipulation of complex relationships of the database applications (in particular n-ary relationships); (b) lack of representation and management of the constraints as an element of a system; (c) permission to model behaviour of database applications in procedural language, causing problems for integrating and translating operations in local databases. On the other hand, the entity relation-

ship models solve the problems related to the o-o models by allowing descriptions of all database information. Therefore, they fail to express behaviour information of database applications. Based on these aspects Tari proposed the ERC++ model to be used as a canonical data model. ERC++ is an extension of the semantic entity relationship ERC+ data model (S. Spaccapietra and C. Parent) that uses rules (first order logic formulae) to represent general and behaviour aspects.

In [21, 22] Castellanos et al. proposed a semantic extension of an o-o model named BLOOM, to be used as a canonical model. BLOOM contains objects, types, classes and semantic abstractions. *Objects* represent the real world objects; *types* describe the structure and behaviour of the object instances; *class* is a set of objects associated with a type; *semantic abstractions* are based on a rich set of semantic concepts, which permits the distinction of different types of dependencies, specialisations, and aggregation and captures more semantics than most of the existing data models. The model has a set of integration operators to support multiple semantic in a federated schema and to overcome schematic discrepancies. It also contains upward inheritance where global types, formed by the integration of local types, inherit their structure and behaviour. This facilitate the task of defining federated schemas. BLOOM was developed with the objective of coupling relational and o-o databases. It achieves this goal after the conversion of relational and o-o schemas to BLOOM schemas. In BLOOM the comparison task in the integration process (see subsubsection 2.1.3) is guided by the structures of generalisation and aggregation semi-lattices. Normally, the searches are not guided at all. However, when they are guided many comparisons are eliminated causing reduction of the complexity.

We affirm that the choice of a suitable common data model must depend on the data models of the components that participate in a federation. Actually, it is necessary to make an engineering task to analyse the best canonical data model depending on the combinations of the existing component data models. It is also very difficult to translate a database schema in one model to a schema in another model when considering a general level. That is, it is possible to find some approaches to perform translations between some specific data models, but there are no general techniques to do this. The semantic and behaviour information aspects are very important when trying to translate the specifications of data models. However, the great majority of the current data models suffer from the lack of semantic and/or behavioural information.

2.1.3 The Integration Process

Heiler et al. [47] defined *integration* as a means of combining or interfacing data and functions of a system into a cohesive set. The integration process requires the identification of relationships and dependencies among data and procedures. A great number of methodologies, tools and solutions to integrate database schemas have been proposed and presented as survey in the literature as can be found in [5, 6, 20, 27, 30, 31, 39, 40, 62, 64, 77, 78, 80, 93, 101, 113]. Other solutions have been suggested by commercial and prototype systems such as: DATAPLEX [14, 26, 104], DDTS [23, 91], DHIM [25], MERMAID [14, 91, 104], MULTIBASE [14, 23, 91, 97, 104], PEGASUS [1, 57]. However, despite the methodologies, algorithms and heuristics that help the integration process, it is necessary to have human assistance to support the resolution of conflicts (there is a lack of automatic processes). This assistance is only reasonable when the number of components is small. There are some systems [11, 14, 23, 30, 91, 97] that use an *auxiliary database* to assist

in the resolution of these incompatibilities. As outlined in [93] the results of a schema integration activity are not unique and cannot be generated totally automatically. The integration process needs information that goes further than synthesising dependencies and involves a subjective activity. The data models are unable to capture the semantics of real world objects in terms of their meaning and use. Besides, the meta-data information about the modelled objects that is normally captured in a schema is not enough.

Larson et al. [62] discussed the use of attribute equivalence for schema integration. They affirmed that two attributes are considered to be equivalent when there is a certain mapping function between the domains of two attributes. Sheth et al. [93] outlined that this attribute equivalence definition is incomplete and inadequate. They believe that a mapping does not imply attribute equivalences. There is no particular set of descriptors to define an attribute that is proper to help with attribute equivalence discovery.

Batini et al. [6] compared some of the existing methodologies that deals with integration and described the following steps to be performed.

1. Preintegration: consists in analysing schemas before integrating them, in order to decide upon the integration policy to be used. This policy is related to the choice of schemas to be integrated, the order of integration and preferences to entire schemas or portions of schemas.

2. Comparison of schemas: in this stage the schemas are compared and analysed to determine relationships among concepts and to detect conflicts and interschema properties.

3. Conforming the schemas: corresponds to the resolution of conflicts to allow the merging of various schemas. In this phase it is necessary close interactions with users.

4. Merging and restructuring: perform the unification of schemas originating some intermediate integrated schema(s). The global schema has to possess the following characteristics: (a) *completeness and correctness*, the original component must contain all the information presented before the unification process; (b) *minimality*, guarantees that the concepts are represented only once; and (c) *understandability*, allows the end user to easily understand the unified schema.

Spaccapietra et al. [100] divided the integration process into two phases. The first phase, named *investigation phase*, consists in determining commonalities and discrepancies among schemas. The second phase consists in the semi-automatically *integration*. In this phase the DBA interacts whenever the integrator does not have knowledge to resolve conflicts.

2.1.4 Conclusion

We believe that it is not possible to allow different databases, independently created and administered, to share and exchange information without addressing the similarity and equivalent aspects of the involved data. Hence, the semantic related to the data is a fundamental question and still needs to be solved. It is necessary to find a simple and automatic way, as much as possible, of representing and manipulating with the semantic.

These semantic aspects reduce and distinguish the differences among data of databases independently created, identify the equivalence and similarities between different data and permit comparison of distinct data.

The semantic of the data is significant and used in various phases such as: the discovery of the correct information, the integration and unification process, the conflict resolution, and so on.

The conflict resolution is another aspect that needs attention. The goal is to create a way to deal with this feature without having to ask for human help (or ask only when strictly necessary).

On the other hand, it is also important to make a comparative study of the different existing data models, using different possible combinations, to define the suitable canonical data and in which situation. Notice that this is a laborious engineering task, but necessary to be done.

2.2 Global Schema

The first approach that appeared in the literature to the problem of managing with heterogeneous database systems was based on the design of a global schema. This global schema is produced by selecting the independently developed schemas of each component (local schema), resolving semantic and syntactic conflicts among them, and creating an integrated schema with all their information (see subsubsection 2.1.3). This process is also called *view integration* by Bright et al. [14] and is more difficult than just creating a union of the input schemas. It has to deal with ambiguity and similarity of terms, and the differences in the representation and interdependencies of the data. The global schema is simply another layer above the local schemas and can be usually replicated at each node for efficient user access [14][4]. As presented in figure 2.1, some of the architectures that use a global schema convert each local schema to another semantically equivalent schema, defined in terms of a common data model, before integrating them and forming the global schema. Therefore, providing data model transparency, since the global schema permits uniform access to the heterogeneous databases (homogenisation). Examples of these architectures can be found in systems and approaches like: MERMAID [14, 91, 104], DATAPLEX [14, 26, 104], DDTS [23, 91], MULTIBASE [14, 23, 91, 97, 104], Dayal [30].

The building of a global schema formed by the integration of different schemas (local) is not a simple task. It has to resolve problems of semantic and syntactic conflicts, equivalence and ambiguity of data, and differences in modelling and viewing aspects of the real world. All of these have to be executed without losing the meanings and goals where the data was originally created.

One important drawback when using a global schema is related to the fact that the autonomy of the local databases cannot be guaranteed. Bouguettaya [7] affirmed that there is a tradeoff between sharing and autonomy summarised as: "the more there is sharing, the less autonomous databases are". With a global schema the databases do not have self-government to define the representation and naming of the sharing data, to determine the way that it views the data and combine existing data, to specify whether and how to communicate with other components, to execute local operations without

[4]Sometimes, when the nodes have limited storage facilities, it is either difficult or impossible to replicate the global schema.

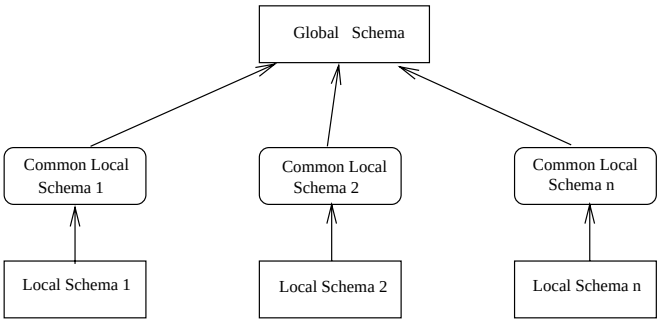

Figure 2.1: Global Schema Architecture

interference from external operations, and so on, compromising the autonomy of these components. Another problem is related with the addition of new component databases into the system and with the modification of the existing local schemas. Both operations have to be reflected into the global schema causing alterations. In some cases these alterations can force the reconsideration of many design decisions [14].

In the global schema approach it is also necessary to define a data model and a data manipulation language (canonical) that is used in the global schema. Every access to the databases presented in the federation is first executed in the global schema, via a common data language (universal). Following, it is distributed to the appropriate components in such a way that is transparent to the users. Therefore, breaking query into subqueries and relating each subquery to a proper local database are also difficult tasks.

2.3 Interoperability

Due to the difficulty of integrating the component databases of a federation into a global schema other approaches appeared in the literature. These approaches are based on the idea of making heterogeneous databases interoperable without using a global schema. Litwin and Abdellatif [68] affirmed that: "a single schema for the thousands of databases on future open systems is a dream". Examples of existing interoperable commercial systems are: MRDSM (Multics Relational data Store Multiple) [14, 68, 69, 91], Calida [14, 69], Oracle V5 [68, 69], Sybase [68, 69, 104].

In the next section we first present some architectures and methodologies that were initially proposed in the literature and that were used as foundation to other approaches. We classify the interoperability approaches into direct and indirect depending on the way that the databases communicate between each other, as introduce in subsections 5.2 and 5.3 respectively. Then, we show different phases of the interoperability process. Following, we exhibit some existing approaches classified into two subgroups. The first subgroup uses a centralised structure like dictionaries and/or repositories containing information about the objects that are exported by the components. The second subgroup does not make use of any centralised structure. Finally, we present a conclusion with some outstanding problems.

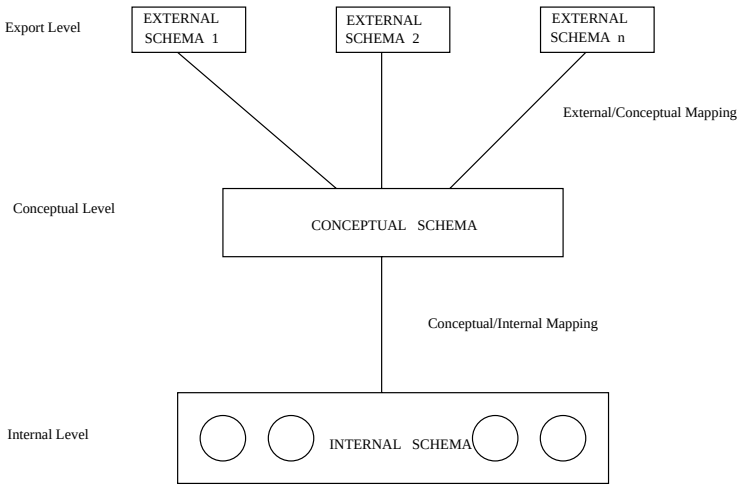

Figure 2.2: ANSI/SPARC three-Level schema architecture

2.3.1 Initial Architectures

We present a general description of some architectures and methodologies that are used in distributed database systems. These architectures are applied by certain commercial and prototype systems to solve the problem of interoperability among heterogeneous databases. For historical, informative and comparative reasons the standard three-level schema architecture for centralised database systems (ANSI/SPARC Study Group Data Base Management System) [28, 106] is also reviewed.

ANSI/SPARC Three-Level Architecture

This architecture is divided into three general levels: external, conceptual and internal as shown in figure 2.2.

External Level: Is concerned with the way that the data is viewed by individual users. The subset of the database that may be accessed by a user or a class of users is described by an *external schema*.

Conceptual Level: Consists of objects that provide a conceptual or logical-level description of the database. It is described by a *conceptual schema* consisting of conceptual or logical data structures and the relationships among these structures.

Internal Level: Is concerned with the way that the data is stored. It contains physical characteristics of the logical data structures in the conceptual level and is described by an *internal schema*.

According to figure 2.2 there are two levels of *mappings* defining the correspondence between the objects of the external and the conceptual schema and between the objects of the conceptual and the internal schema.

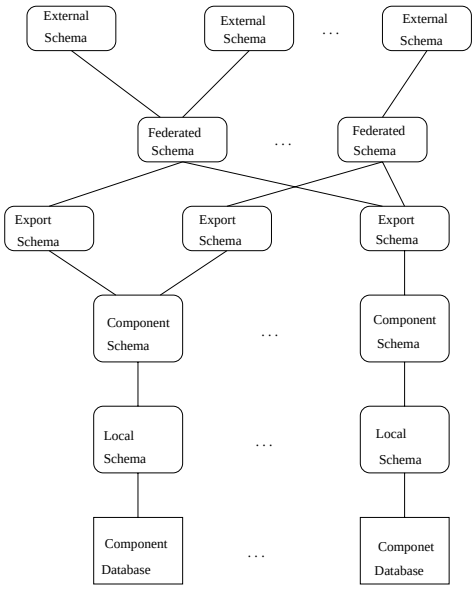

Figure 2.3: Five-Level schema architecture

Five-Level Schema Architecture

Sheth and Larson [91] proposed a five-level schema architecture for describing the architecture of a database system that is distributed, heterogeneous and autonomous[5]. This architecture is an extension of the three-level architecture as shown in figure 2.3 and figure 2.4 and includes the following.

Local Schema: Is the conceptual schema of a component database system and it is expressed in the same native data model of the component, that is, different local schemas may be expressed in different data models.

Component Schema: Is the result of the translation of a local schema into a common data model previously chosen; this translation is done by the *transforming processor*.

Export Schema: Represents the subset of a component schema that is available to the federation and its users. Notice that not all data of a component are available. The *filtering processor* is used to provide the access control and to limit the set of operations that can be submitted on the corresponding component schema.

Federated Schema: Is the integration of multiple export schemas and includes the information of data distribution that is generated when integrating export schemas[6]. It is possible to have multiple federated schemas in a system, one for each class of federation users (a group of users and/or applications performing a related set of activities). To transform commands from the federated schema into one or more export schemas, the architecture uses the *constructing processor*.

[5]For Sheth and Larson [91] a distributed, heterogeneous and autonomous database system was called as *federated database system*.

[6]There are systems that use a separate schema to contain these information called *distribution schema* or *allocation schema*.

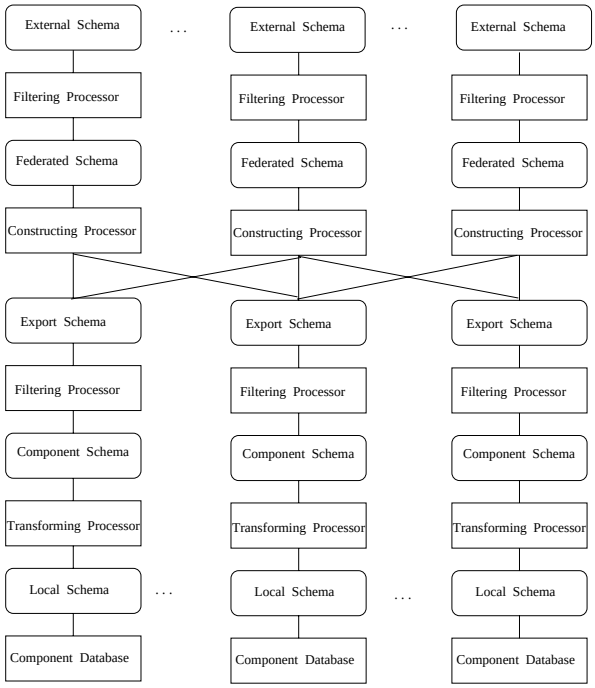

Figure 2.4: System architecute of the five-level schema

External Schema: Defines a schema for a user, or application, or a class of users and applications. A *filtering processor* is used to analyse the commands on an external schema and to ensure their conformance with access control and integrity constraints of the federated schema. When the external and the federated schemas present different data models it is necessary to use the transforming processor to transform commands between these schemas.

The federated, export and component schemas must be represented using the same data model named *canonical* or *common data model*, previously chosen. All of their commands must be expressed using an *internal command language* (see subsubsection 2.1.2). Thereby, this architecture poses the problem of choosing a common data model and an internal command language that are suitable for the federation. The component schema contains unnecessary translated information since it comprises all the local schema in a canonical data model. Generally, only part of this schema is available to the federation and users. Another drawback is the integration of schemas in the federated schema level, even when these schemas are represented in the same data model, because of possible semantic and syntactic differences. Notice that in the five-level schema architecture the export and federated schemas avoid the access of a certain data by unauthorised users, guaranteeing more privacy and confidentiality on the system.

Sheth and Larson [91] outlined the possibility of redundancies in this five-level schema architecture (notice that the local, component and export schemas embody the same information). Therefore, various alternative architectures can be derived from the five-level schema architecture by adding or removing some basic components.

Federated Database Architecture

Hammer and McLeod [44] proposed the notion of a federated database that is a loosely coupled set of component databases. Heimbigner and Mcleod [48] extended this principle to the notion of a loosely coupled database without a global schema. The federated architecture was initially proposed for a federation of databases using the same data model (object-oriented). However, it is possible to apply it for a heterogeneous federation (translating the original database schemas in schemas described in the object-oriented data model). The key points in a federated approach are autonomy and cooperation in interdatabase sharing. It is important to remain that federated database approach appeared to provide a better alternative to the global integration approach (see subsection 2.2) and was not designed to address the issue of large multidatabases [7].

The federated database model used in [48] is based on three basic data modelling primitives: *objects*, corresponding to some entity or concept of the real world; *types*, that are time-varying collections of objects sharing common properties; and *maps*, "functions" that map objects from some domain type to sets of objects in the power set of some range type (e.g., a map where the value of all objects in the domain type has cardinality between zero and eight).

In the federated architecture the basic elements are *components* which represent individual information systems that desire to share and exchange information. It is possible to have any number of these components in the federation. A component may be viewed as an autonomous database and has three schemas: *private, export* and *import*, as shown in figure 2.5.

Private Schema: Describes the portion of data of a component that is local to the component. This portion of the schema and the data that it describes correspond to a normal database in a nonfederated environment. It also contains some information and transactions relevant to the participation of the component in the federation as: descriptive information about the component, primitive operations for data manipulation, and the import and export schemas.

Export Schema: Specifies the information that a component is willing to share with other components of the federation, that is, the information to be exported to other components. It is a metaschema consisting of a set of types and maps in the component schema that contains the definitions of types and maps that are to be exported.

Import Schema[7]: Specifies the information that a component desires to use from other components grouping data from several export schemas. After the importation of some types and maps a component can restructure this information to adapt for its purposes. This restructure is made by the *derivation operators* that manipulate definitions of types and maps to produce new ones.

Each federation has a single distinguished component named *federal dictionary* that contains information about the federation itself and describes available data items and services in the federation. Thus, the difference between the federal dictionary and any

[7] This concept is similar to the federated schema of the five-level architecture [91].

other component is the database that it contains. The federal dictionary does not mediate communications among other components and has no direct control over them. As denoted by Milliner and Papazoglou [74, 75], in a large unstructured network of data dictionary it is very difficult to locate the desire information. When the federation contains a large number of components, the use of the federal dictionary is a performance bottleneck. In a centralised structure like the federal dictionary it is also necessary to resolve the problem of failures and to permit an easy way of making updates. Notice that with the dictionary any component willing to participate in the federation has to share data and this data can be accessed by any other component. Hence, it is not possible to guarantee privacy and confidentiality of the information without using special strategies. It is also impractical to allow a component to participate in the federation only interrogating the other components, that is, without having to share any data.

In the federated architecture the process of accessing a data from another component is executed in several steps. First, a component consults the federal dictionary to discover about existing databases and schemas. Second, it contacts those components, examines their export schemas and starts the "importation process". During the importation phase the federated architecture uses a mechanism of negotiation to coordinate the sharing of information between two components. The negotiation is a multistep, distributed dialogue among two components that establishes the right to access some data elements (a kind of contract). The importation of a type or map is a separate process from the process of accessing a data. The type (map) is imported once and introduced in the import schema of the importing database. The accesses to the contents of that type (map) are carried out directly without any negotiation. Notice that during the importation, the importing database has to understand and comprehend the imported schema. This is difficult in an environment formed by heterogeneous databases, where the importation may force translations of the imported schema.

A drawback in the federated architecture is that it does not have the concept of interdatabase dependencies between the export schemas. That is, consistency constraints among data of two or more different components. In this case, to achieve the interdatabase dependencies we suggest the addition of an extra element (component) to the architecture, containing this information. Another difficulty in this architecture is related to the unit of data sharing that in this case is the *schema*. The components of the federation record their export schemas in the federal dictionary. As outlined in [7] the databases have to show all information they export to the whole federation violating the autonomy and privacy, and the importing databases have to understand the organisation of the imported schema. Notice that in the five-level schema architecture the unit of data sharing is also the schema. However, the export and federated schemas avoid the problem of showing all the information that a component export to the whole federation.

Multidatabase Architecture

The multidatabase architecture proposed by Litwin et al. [69, 71] extends the ANSI/SPARC architecture as can be seen in figure 2.6. The architecture is divided into the following levels.

Internal Level: This level contains the existing databases, each with its *physical schema* (PS_i), $1 \leq i \leq n$, where n is the number of existing databases.

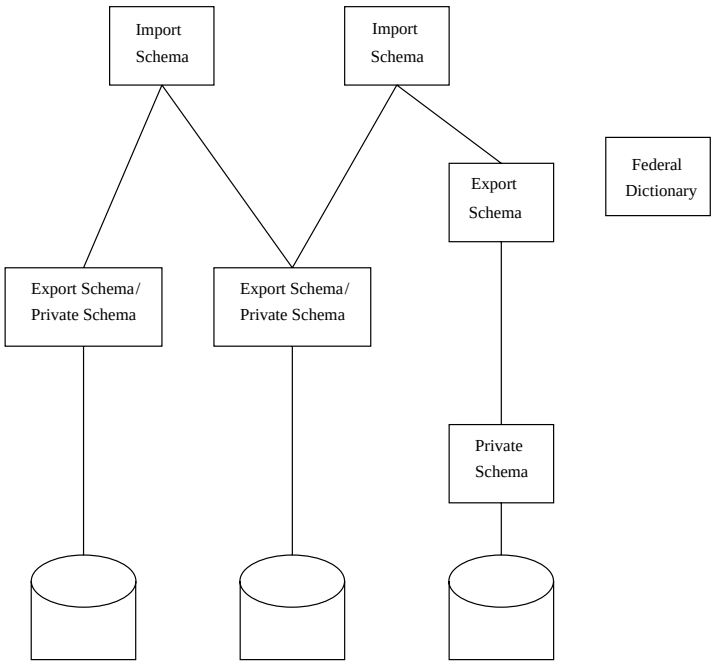

Figure 2.5: Federated database architecture

Conceptual Multidatabase Level: This level contains the *conceptual schema* (CS_i) of the database willing to cooperate. This schema may be the actual conceptual schema or a local external schema. In the latter case, the actual conceptual schema is called an internal logical schema (IS_i). The conceptual schema may support different data models and may hide private data. Unlike the federated architecture, it is possible to have definitions of dependencies between sub-collections of databases. These dependencies are expressed as *dependency schemas* (DS_i) consisting of the only tool to preserve the consistency of data in different databases (note that in this architecture we have the absence of a global schema). Generally, the schema in this level are represented in a common data model. Thus, the problem of choosing a suitable data model to be used as a common one persists[8].

External Level: This level contains the construction of *external schemas* (ES_i), that are mono or multidatabase schemas, presented as a collection of databases integrated as a single one. A database may participate in different external schemas and be manipulated locally. When the appropriate interdatabase dependencies are not enforced it is not possible to guarantee the consistency of data from different databases presented as a single database, since there is not a global schema in the architecture.

The access of the multiple databases is performed directly at the multidatabase level, using a multidatabase universal language or common language), or through an external view, using either a multidatabase language or a (local) database language, in the case that the external schema defines a single database. The multidatabase language must allow the

[8]In [71] the authors suggested the use of the relational data model to be used as a common model since it contains a nonprocedural manipulation language.

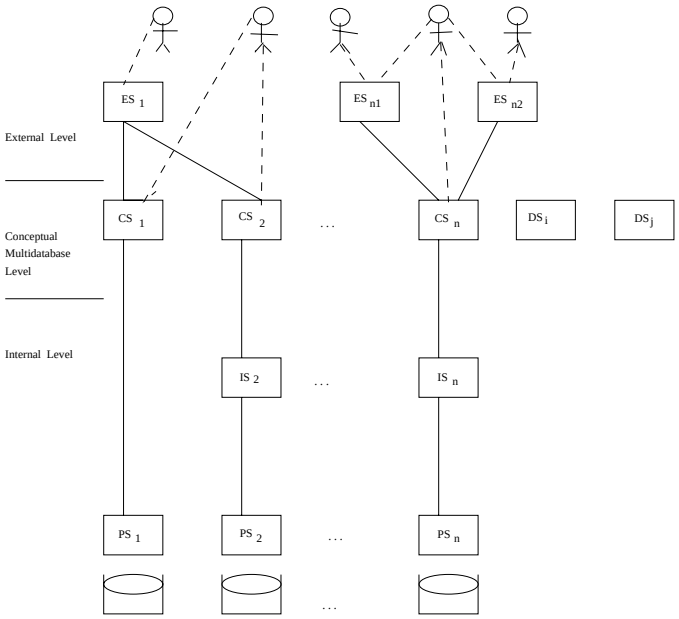

Figure 2.6: Multidatabases schema architecture

users to define and manipulate a collection of autonomous databases in a nonprocedural way. In [68, 69, 71] they assume that a set of databases becomes a multidatabase only when a multidatabase language is provided.

Unlike distributed databases, in the multidatabase architecture we can have the distinction between the notion of *database* and *site*. The former is a logical collection of data bearing a semantically meaningful name; the latter is a distinct physical network node at which some number of databases reside.

Litwin et al. [71] compared the federated architecture with the multidatabase one and affirmed that they are very similar. An import schema corresponds to an external schema, a private schema corresponds to either the internal schema or the conceptual schema at the multidatabase level, and an export schema could be equivalent to a conceptual schema at the multidatabase level. In our opinion the federated and multidatabase architectures differ in the sense that the multidatabase architecture does not make distinctions between the public (shared) and private data. In the federated architecture the public data is described in the export schema. In the multidatabase architecture the public data is specified in the conceptual schema (or internal local schema) which also describes all the private data of a component. We believe that this distinction is very important. It avoids the access of not allowed data and the necessity of using different ways to make distinctions between the public and the private data (using the data manipulation language, for instance).

The multidatabase architecture does not contain mechanism of negotiation to coordinate the sharing of information. It permits users and/or applications to access the data of the different databases either directly or using an external view, formed by the integration of different parts of distinct databases (external schema). Notice that the construction of the external schemas forces the resolution of semantic and syntactic conflicts during the integration of some schemas.

Conclusion

In this subsection we presented some architectures and methodologies proposed to permit the access and manipulation of heterogeneous, autonomous databases, without performing a global integration of these databases. In subsection 2.3.5 we exhibit other existing approaches. However, even having some advantageous aspects and good features, these architectures contain some reminding problems outlined below.

The five-level and multidatabase architectures have the problem of choosing and using a canonical data model. Notice that this problem does not appear explicit in the federation architecture, since it was initially proposed for a federation of databases using the same data model. However, when this architecture is applied to databases containing different data models it is necessary to use a common data model to describe the export and import schemas. The distinction between the shared data (public) and the private data is very significant to guarantee privacy of the data. Therefore, it is necessary to be careful on the type and the way that the data is shared. It is important to avoid a component to show all of its exported information to all components of the federation compromising the privacy of its data. the approaches applied to manipulate heterogeneous databases should not allow the use of centralised structures as dictionaries and repositorious, containing global information about the involved components. They guarantee the interdatabase dependencies in order to preserve the consistency of data in different components. Other question that these architectures do not solve is related to the fact of preventing the access of a certain data by the same component more than one time, generating a great amount of traffic in the system. That is, the lack of "cache structures" containing data frequently accessed.

In an environment of different autonomous databases independently created and administered it is necessary to have semantic related to the data. Since the existing data models are not rich enough in semantic it is essential to find a way of representing this semantic. Notice that the present architectures do not mention anything about this problem.

2.3.2 Direct Interoperability

The direct interoperability between autonomous heterogeneous databases consists in the direct mapping (translation) among the components, as assigned in figure 2.7. The task of direct mapping two different databases may be extremely complex or sometimes impossible, specially when one of the components is semantically more expressive then the other one. Another problem is related to the high number of transformators and translators of schemas, where in this case can grow with the square of the number of different types of data models existing in the federation ($O(n^2)$).

2.3.3 Indirect Interoperability

When applying this strategy different and multiple autonomous heterogeneous databases are manipulated and managed by the use of an intermediate (canonical) data model and data manipulation language. The canonical data model is used to represent other models, bridge the gap between local models, detect interdatabase semantic relationships and achieve interoperability. The original schemas of the components are converted into

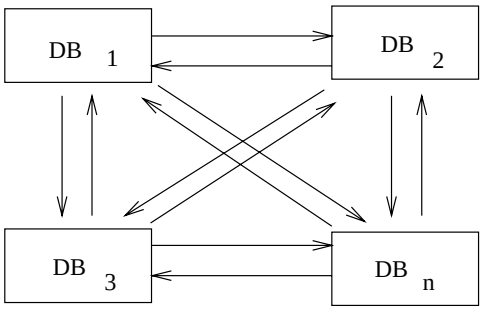

Figure 2.7: Direct interoperability of components

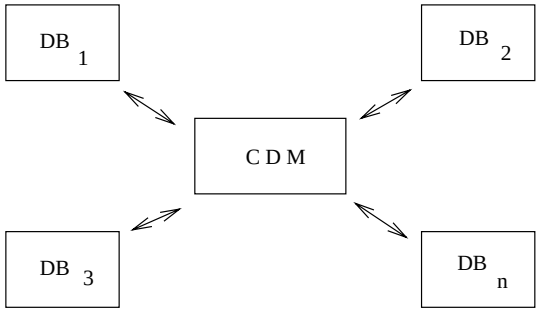

Figure 2.8: Indirect interoperability using canonical data model (CDM)

schemas in the canonical data model originating a certain homogenisation. This process is called *schema transformation* or *generation* [51]. Hsiao [51] declared that the capability of having various database schemas in different data models for the same database system is important for data sharing. However, he affirmed that it is not possible to access and manipulate a database, via a new schema, in a data model different from the original one. Associated with the new data model of this schema there is a data language and the database system has to be able to understand the semantic of this language. Hence, it is necessary to have *transaction translations*, that is, a transaction written in the data language of the new data model is translated into an equivalent transaction in the original data language.

The approach is executed in two steps. The first phase is related to the transformation (mapping) of a source database schema in one data model into a database schema in the common data model, and to the translation of transaction written in the source data language into transaction in the common data language. The second phase lies on the translations of schemas and transactions in the common data model and data language into equivalent schemas and transactions in the target (remote) database schema and language, respectively. The process must guarantee that the output schema is a valid schema for the target database. Figure 2.8 presents this situation. Notice that the intermediate data model is conceptual and virtual. There is no database related to the intermediate data model. In [54, 55], Johannesson proposed a method for translating schemas in the context of a logic based modelling approach. In particular, the proposed method executes transformations from relational schemas to conceptual schemas.

With an intermediate data model/data manipulation language, the number of schema

transformators and language translators is linear with the heterogeneity of the federation. For n different types of data models in the federation, $2n$ transformators and $2n$ translators ($O(n)$) are necessary. Notice that this approach permits the inclusion of any database containing new data model and data language. The addition of new data models and languages into the federation causes only the generation of new transformators and translators between the data model and language of the new database and the canonical one, and vice-versa.

The problem of resolving semantic and syntactic conflicts during the mappings is also presented. It is not possible to guarantee consistency when the architecture does not contain an appropriate method to delineate interdatabase dependencies (see subsection 2.4). The use of a canonical data model (language) may produce a situation where two different components of the federation, DB_1 and DB_2 for example, want to share data and both of them are specified in the same data model. In this case, the mappings between the data models of DB_1 and DB_2, and the intermediate data model generate unnecessary work.

Unlike the global integration approach, when using a canonical data model is possible to maintain much of the autonomy of the heterogeneous databases of the federation. Hsiao [50] mentioned that despite the linear number of transformators and translators, this approach is also important to the specification and validation of the access and concurrency controls. To perform specification, the schemas and transaction in the intermediate data model and language, respectively, are translated into schemas and transaction in the local data models and language of a local database. Then, to execute validation, the access and concurrency control mechanisms of the local database must check the translated specification against data in the system. Therefore, the autonomy of each component is guaranteed by the local access and control mechanisms. However, for concurrent accesses to databases that are in different places it may be necessary to specify, in the canonical data model, which are the integrity constraints, application specifications and security requirements. Otherwise, the system cannot guarantee reliability and deadlock free accesses.

We believe that the best concept is to use both the direct and indirect interoperability approaches interchangeable. That is, use a canonical data model/data manipulation language to intermediate operations between databases having different data models (data manipulation language). On situations where the interoperation has to be done between databases containing the same data model (data language), we allow them to interoperate directly without making unnecessary work. This is a good approach since it avoids the problems of having a high number of translators and transformations, avoids differences in semantic expressiveness of the data models and unnecessary translations, and guarantees the maintenance of the components autonomy. Nevertheless, the problem of choosing a suitable canonical data model/data manipulation language remains (engineering task, see subsubsection 2.1.2).

Use of Metamodel

Instead of using an existing commercial data model as a canonical data model, some approaches use a metamodel[9] to execute this task.

Atzeni and Torlone [4] proposed a metamodel which is a framework for the definition of different data models and for the management of translations of schemas from one model

[9] A metamodels is a formalism for the definition of models.

to another. This approach does not cover all of the possible models, but covers all of the widely used conceptual models and can be extended to be applied for other models[10]. The idea consists in reducing the constructs of the conceptual models to few categories as: *lexical, types, abstract types, aggregations, grouping constructs, functions, generalisations*, allowing a model to be defined with respect to these metaconstructs. Notice that if a model with a completely new construct is proposed this construct can be easily introduced into the metamodel. The definition of a model with respect to the metaconstructs is done by a language named *model definition language* (MDL). The definition of a model by MDL generates a *schema definition language* (SDL).

After defining two different models by means of the metamodel it is necessary to permit the transformation of schemas in these two models. It is assumed that there is an intermediate "supermodel" which involves a construct for each metaconstruct in the metamodel. Thus, the transformation process is executed in two steps via the "supermodel". The process must guarantee *validation*, that is, the output schema has to be a valid schema for the target database. Other requirement is the preservation of equivalence, although there is no accepted definition of equivalence and some problems may appear independently of the notion of equivalence assumed. For instance, some of these problems can be the *loss of information*, when the data model of the source component allows a finer representation of features than the data model of the target database; or *degradation*, when it is possible to have two equivalent schemas, but these schemas do not represent the same semantics. The authors affirm that since it is very difficult to find general automatic transformations that work in every case, part of the process has to be specified by a specialist called *model engineer*. The model engineer has to specify a transformation for each aspect of metaconstruct for which the model being defined has no construct. This work can be supported by a set of predefined functions implementing the standard transformations between the basic constructs.

On the other hand, Papazoglou et al. [82] affirmed that the use of a canonical data model to achieve interoperability suffer from two drawbacks. The first drawback is the assumption that different data models can be mapped directly into a central more expressive data model. The second problem is related to the impossibility of using the canonical data model for incremental transformations. That is, the translation of a query from a source to a target model requires the existence of an entire equivalent canonical schema for both databases. Thus, to avoid the complex task of choosing a suitable canonical data model and the existence of an entire canonical schema for the involved databases, the authors proposed the use of an *intermediate meta-model* as a generic technique for translations between diverse data models.

The propose approach is based on the fact that the intermediate meta-model is able to capture and describe typical constructs (elements of models) that are found in conventional data models in a generic format. This allows a data model to be defined in terms of these meta-constructs (elements of metamodels). In their approach the intermediate metamodel views a meta-model as a collection of high-level abstractions (metaclasses) and a schema as an instantiation of the abstract constructs. An instance of the intermediate metamodel is known as an intermediate schema meta-graph or ISMG[11].

[10] In [4] the authors do not want to solve the whole problem, but want to show the feasibility of the approach.

[11] It is worth mention that this approach has some similarities with the idea proposed by Gangopadhyay and Barsalon [42], even not having the same goal. In [42] the syntax and semantics of data models,

The transformation of schemas into a collection of populated ISMG meta-classes is performed in three steps. The first step consists in the enrichment of given schemas to guarantee semantic consistency. In the second step, the enriched schemas are passed through a class library that together with some generation rules generates the ISMG meta-classes. On the third step, the ISMG descriptions of two different schemas generated in the second step are compared and mapped into each other by applying transformation rules. The generation and transformation rules specified in [82] are for the relational and object-oriented data models.

Therefore, whenever a query made in one component (source) has to use any other different component of the federation (target) it is necessary to identify the partition of the source ISMG related to the query, called *query graph* (QG). Then, the QG together with the query are sent to the target database. Notice that the authors assumed that the target component is previously known. When the QG and the query arrive in the target component the QG is superimposed on the ISMG of the remote component to identify the differences between them. After the identification of the differences the system uses transformation rules to convert the query graph into a form that most closely semantically matches the target ISMG. In the next step the received query is translated into a form that makes it executable at the (target) component. When the execution is concluded the results of the query are sent (returned) to the source database.

In our opinion the idea of using either a canonical data model or an intermediate meta-model is actually to achieve the same purpose, that is, interoperability of heterogeneous databases using an intermediate framework to execute transformations and translations. The use of an intermediate meta model only avoids the hard task of chosen an existing data model suitable for the role of a canonical model. Notice that in the approach proposed in [82], instead of having an entire canonical schema for each involved database the approach has the ISMG. We believe that in a federation of heterogeneous databases it is important to have an intermediate structure (data model) to facilitate the transformations and translations. However, the choice of which model to be used depends on the heterogeneity of the system, i.e., on the kind of data models that are trying to interoperate (engineering task). One approach to avoid the problem of translations is to use the idea of maintaining a copy of the different schemas to be shared in the canonical data model previously chose.

2.3.4 Phases of the Interoperability Process

Hammer and Mcleod [45] divided the interoperability process into three subtasks (phases) that can be executed during different operational phases, outlined below:

1. Resource discovery and identification:

   This first phase is also known as *information discovery*. It consists in locating and identifying information in the other components of the federation that is relevant, identical, similar, or related to the requested data. Sheth [92] affirmed that in the future the primary issue will not be how to efficiently process the data that is known to be relevant, but to determine which data is relevant and where it is located. The

schemas and databases can be uniformly described by the use of a metalevel system called M(D)M. The data modelling constructs are mapped to this formal common metamodel which formalises the syntax and the semantics of the constructs as a collection of second-order logic formulae.

great majority of the approaches related to the resource discovery are based on the idea of using a federal dictionary or repository containing the exported data [15, 16, 45, 46, 110]. They assume that complete descriptions are specified a priori and do not normally change. However, this is not what happens in a federation of numerous databases (components).

2. Resolution of semantic heterogeneity:

   After identifying the remote information suitable for a certain request it is necessary to detect and resolve some conflicts that may exist between the local and non-local information. Thus, the basic problem consists in determining the relationship that exists between a remote and local object. To compare these objects it is necessary to use the notion of *equivalence* and *similarities* (see subsubsection 2.1.1).

3. Unification of remote and local data:

   The third and last phase consists in the importation of the foreign information into the local component, as natural as possible. However, this unification task is not simple and in some cases the local component has to be restructured to guarantee the *completeness, minimality* and *understandability* of the final result (see subsubsection 2.1.3). During the importation of the remote data it is possible that some structural conflicts appear, forcing their resolution.

2.3.5 Other Approaches

We present below some of the existing approaches that permit interoperability in heterogeneous databases. Some of these approaches are derived from the initial architectures presented in subsubsection 2.3.1. We divide these approaches into centralised and decentralised, based on the existence of a central structure to help with the information discovery and relationship identification processes.

Centralised

This group of approaches [15, 16, 37, 38, 45, 46, 110, 111] uses a centralised structure, such as a dictionary or repository, containing information about the objects that are exported by the components of a federation (shared data) and the location of those objects[12]. A centralised structure is not suitable for a system where a component can often join, leave and change their specifications, since these tasks generate too many updates. Other problems are related to the performance bottleneck that may exist when a great number of components access the centralised structure, to the failures that can occur in this structure, and to the fact that not all sites want to advertise their data to everyone. Notice that when an information (shared data) is introduced into a dictionary or repository it can be used by all components that access these structures not guaranteeing privacy and confidentiality. With a centralised structure it is also difficult to allow a component to participate in the system without sharing data, i.e., only interrogating other databases. We present below some architectures that can be classified in this group.

1. **Remote-Exchange**

[12] The federated architecture [48] is also classified in this group.

In the Remote-Exchange system [37, 38, 45, 46] the discovery of the appropriate remote data relevant to the request of a component is performed by a special component, named *sharing advisor*. It uses a centralised structure called *semantic dictionary* containing knowledge about the objects that are exported by the components of the federation (shared data). The semantic dictionary is accessed by the sharing advisor. It represents a dynamic *federated knowledge base* about shared information in the federation. The sharing advisor also uses *sharing heuristics* that allow it to identify whether either the meaning of a type object being registered can be determined based upon its properties or it is necessary additional assistance from users. Thus, whenever a component inquires for a certain information the sharing advisor uses the information of the semantic dictionary and a set of heuristic rules to identify the concepts that it considers to be relevant. The approach uses a functional object database model called *Minimal Object Data Model* (MODM) to describe the shared data (structure, constraints and operations).

When a database is registered into the system this (new) component informs the sharing advisor about the data that it is willing to share with the other components. Notice that in this case the shared data of a component is available to the whole federation, violating the privacy and confidentiality of certain data. It also does not allow a component to participate in the system only interrogating the other components. In the semantic dictionary the information are represented by a hierarchy. In this hierarchy similar types are classified into a collection called *concepts* and subcollections called *subconcepts*. However, the relationships express in this concept hierarchy are only approximations of the true, they do not express strong relationships. Thereby, they ask for mechanisms that provide more exact relationships and human help.

Hammer et al. [46] classified 3 basic kinds of discovery requests combined allow a component to identify non-local information: similar concepts, complementary information and overlapping information.

The sharing advisor uses some techniques to access remote data such as: (a) *semantic and behavioral equivalence*, used to identify semantically related objects; (b) *constraints analysis*, where the knowledge of constraints about remote objects suggests their relevance and relationships among local objects; (c) *probabilistic approach*, where a probability index is assign to each object reflecting the probability that the object is relevant to the requested information; (d) *user feedback data*, the user return a feedback data for each retrieved information based on the relevance of the information with respect to the query; (e) *heuristics*, some rules used to help with the identification of semantically similar objects that can be incremented due to the feedback of data from users; (f) *machine learning techniques*, which can be applied based on heuristics and the feedback data of the user, causing the advisor to identify semantically similar objects more accurately.

Hammer et al. [45, 46] proposed the use of a combination of several different structures to resolve the problem of conflict resolution . The first idea is to use *meta-functions* that return meta-data information about the remote data. The meta-functions contain all the information about the structure of a type object. Instead of encapsulating behaviour they encapsulate the structure of a type object. The second aspect is related to the use of a *local lexicon*, for each component, contain-

ing semantic information (precise meaning) about the shared data exported by this component. This information is represented in a uniform way as a static collection of facts. The local terms are described by concepts, presented in a dynamic list that contains common concepts of the federation, related through a set of predefined relationship descriptors. The authors affirmed that since interoperability only make sense among components that model similar or related information, it is reasonable to expect a common understanding of a minimal set of concepts taken from the application domain. However, we believe that the current goal is to permit interoperation between databases that do not necessary have similar or common information. Thus, this idea is not suitable for the case where, for instance, a user from a database containing information about university wants to discover the address of a certain restaurant in the city of London. The third structure is the semantic dictionary that maintains partial knowledge about the relationships of all terms in the local lexicons of the federation. In some sense it complements the local lexicons, since these structures contain only semantic information and do not have knowledge about the relationships of their objects. The strategy of execution is characterised by performing the majority of the user inputs before the resolution step and not during this step.

The Remote-Exchange system can be considered as a variation of the federated architecture [48] (see subsubsection 2.3.1). It makes the distinction between the private and shared data and uses a centralised semantic dictionary with similar roles to the federated dictionary.

We believe that despite of the problems related to the use of a centralised structure to perform the information discovery process the approach contains drawbacks related to the conflict resolution. Sometimes the conflicts cannot be resolved automatically and ask for user help. The idea of using a structure for each component with semantic information about the shared data is feasible. However, it is necessary to specify a way of representing and accessing this information. Other weaknesses are related to the lack of strong relationships in the contents of the semantic dictionary and to the maintenance of this structure in a dynamic environment, with a large number of components. Notice that the use of a sharing advisor trying to "coordinate" some of the interoperation steps can become impractical for a large number of components. In this case it is necessary a good concurrency control protocol.

In the Remote-Exchange system, different cases were considered in details to allow the execution of the unification phase in [37, 45]. Depending on the situation, the idea is to either add a new class related to the new data being imported or create a superclass adding necessary attributes.

2. **InHead**

   The InHead (Intelligent Heterogeneous Autonomous Database Architecture) proposed by Weishar and Kerschberg [110, 111] incorporates an object-oriented Knowledge/Data Model (KDM). The KDM integrates Artificial Intelligence problem-solving techniques and advanced semantic data modelling techniques.

   The idea is to construct object-oriented *domain models* and an *overall domain model*. The domain models are representations of data and knowledge of the constituent databases. The overall domain model contains semantic relationships among the

objects of those domain models. The domain models are represented as *Knowledge Sources* (KSs) in a *blackboard* architecture[13]. They have global and local domain expertise and are comparable to the local schema. The KSs work together providing users with simultaneous, multiple viewpoints of the system. The overall domain model is a *global thesaurus KS* executing the same role of a data dictionary.

The part of the local KS that is used by other components (shared data of a database) comprises object structure, relationships, constraints, operations and rules that are encapsulated into an abstract object type called *Data/Knowledge Packets*. Thus, not only object structure semantics are exported (shared), but also object operational semantics. The components of the federation remain the same and the data/knowledge packets with the local domain model can be used to resolve semantic problems at the local level, before sending a query to the global knowledge source. For instance, suppose the situation where there are two different databases having the same attribute "price", but in one of them the "price" includes VAT and in the other it does not. Thus, to resolve this situation, it is possible to use a local domain KS together with a data/knowledge packet specifying the meaning of "price".

The propose architecture provides a natural way for knowledge discovery and solving problems cooperatively. This natural way consists in allowing the knowledge sources to share the information in their data/knowledge packets with other KSs, on one or many blackboards.

The InHead global thesaurus KS addresses semantic heterogeneity issues to data items. These data items are similarly named, related, subclass and/or superclass of data items located elsewhere within the federation. It also *actively* works with users to reformulate queries. The thesaurus acts as a repository of knowledge of data-item terms and of their usage, and as an active participant in formulating improved query specification (provides global data-item definitions and locations).

Whenever a query is performed the system consults the global thesaurus using another element called *controller*. If the solution to the query is obvious, the controller sends it to the appropriate components of the federation. Otherwise, the controller sends the query to the blackboard where the KSs try, cooperatively, to find a solution to resolve the semantic ambiguities of the query. Whenever a solution cannot be found (the system cannot resolve semantic ambiguity), the user is consulted to clarify and resolve the problem. Notice that this approach is similar to the Remote-Exchange system. Both approaches use a central structure containing the relationships between the data in the federation and local structures possessing semantic information about the shared data of components. The controller is also responsible to conduct the necessary query translation and optimisation, to send the query to the components, to integrate the query results and to provide the answer to the user. In [110, 111] the authors do not give details of how the semantic conflicts are resolved and how the results are unified, providing the answer to the user.

We affirm that this approach suffers from the same drawbacks presented in the Remote-Exchange approach. When dealing with a dynamic environment contain-

[13]The blackboard framework is regarded by AI researchers as the most general and flexible knowledge system architecture and offers expert system programming techniques.

ing a great amount of components, it is difficult to use a controller coordinating the execution of the system and to update the global thesaurus. The idea of sharing (exporting) not only object structure semantics, but also object operational semantics is feasible. Each component is responsible to define, organise and manipulate with the semantic of their local data, preserving the autonomy of the components. However, it is not clear how to represent and update this semantic information.

3. **Summary Schema Model**

Bright and Hurson [15] and Bright et al. [16] proposed the use of SSM (Summary Schema Model) to provide automated support for identification of semantically similar data in a federation of autonomous, heterogeneous databases. The SSM uses linguistic tools and information retrieval theory to build a global data structure to achieve data discovery automatically. The semantic power of the SSM comes from the linguistic knowledge represented in an on-line taxonomy, that combines information found in dictionaries and thesaurus[14]. The DBA of each component is responsible for linking local shared data to terms in the federated taxonomy. That is, for each local term the DBA chooses the closest dictionary definition in the taxonomy.

A *summary schema* is an abstract, concise description of the local shared data. The summary schema retains most of the semantic contents of the shared data using terms with broader meanings, called *hypernym*, instead of the original data.

In the SSM the components of the federation are structures in a hierarchy. Each internal node contains a summary schema created from the schemas of its children, maintains a copy of the taxonomy and has entries for specific word meanings. The leaf nodes contain database schemas in the relational data model. So, the highest level of the hierarchy contains summary schemas, representing all of the information available in the federation in a concise way. The hierarchy is kept short, but bushy (when the number of components is high). The authors affirmed that even having a global data structure (the hierarchy), the idea of using SSM is better then employing a global schema integrating all of the components. The hierarchy is smaller than a global schema and its creation and maintenance are partially automatic.

In SSM two terms that are semantically related have a path between them using different links in the taxonomy. To measure the semantic similarity of data the SSM uses a Semantic-Distance Metric (SDM). The SDM is a weighted count of the links in the path between two terms. Terms with few links separating them (small SDM value) are similar and terms with many links (large SDM value) are less similar.

When an imprecise query is submitted to the system, that is a query describing data in the own terms of the user, the SSM query processor takes the imprecise terms and maps them to entries in the taxonomy. After the mapping, the system tries to match these terms to terms that are semantically close in the local summary schema, using specified SDM value. When the terms cannot be matched locally, the query processor moves to a higher level in the hierarchy trying to match these terms. If a match is found, the terms are replaced by the shared data references of

[14]The authors used two taxonomies as bases for the SSM: the Roget's and the Webster's 7^{th} New Collegiate Dictionary. A comparison between these two taxonomies is presented in [15].

the local component represented in the matching summary schema. Otherwise, if the terms cannot be matched in the highest level of the hierarchy, then there is no matching available in the federation.

Besides the disadvantages that appear in a centralised structure this approach adds the problem of finding general terms (hypernym) to represent the local access terms. It also has the complexity of the search space based on the complexity of the taxonomy structure.

Decentralised

In this second group, the approaches [7, 8, 9, 18, 35, 34, 74, 75, 81][15] are characterised by not using a centralised structure containing the exported data. We present below some of these approaches.

1. **Global Concepts**

   Milliner and Papazoglou [74, 75] proposed a decentralised approach focused on the step of *inter-node relationship discovery*, as opposed to the step of resource discovery. They affirmed that, in spite of the size and complexity of interoperable databases, it is impractical to use a network-wide global directory and, to submit a circumforaneous query that traverses the entire network to retrieve descriptions of the requested data objects. Based on the idea that relationship information must be "data-driven" and neither specified nor explicit classified, these authors proposed an architecture for dynamic, incremental inter-node relationship formation.

   The information about the relationships between remote nodes[16] (seeding knowledge) is stored in a special node, associated with each database node, called *node knowledge-base*. This special node contains naming conventions and aliases, services descriptor for invoking remote service, definitions of terms in the form of thesaurus, and so on. This information is stored in the form of facts. The sharing and exchange of information is achieved through a common minimal object-oriented data model, describing the structures, constraints and operations of the shared data. The distributed information environment is based on a client/server-oriented multidatabase (CSOMD) architecture, where each node contains an *information broker* able to accept client requests and locate the place where these requests can be executed. To guarantee transparency in the process of the broker, the architecture contains, for each node, other components such as repositories, directories and dictionaries. These components provide information about identity and interfaces to allow the location and execution of remote operations.

   Milliner and Papazoglou proposed a high level "context abstraction" to maintain the organisational autonomy of the nodes. The context abstraction defines objects using *Global Concepts* (GCs). These GCs divide the Universe of Discourse (UoD) into dynamic clusters of database nodes, based on areas of interest, causing reduction in the search space. The databases are related to the GCs by *link weights*. Thus, a database belonging to a cluster has a strong link to the GC that this cluster

[15] Some of these approaches were suggested to the problem of integrating many systems (not only databases), which comprise the emerging field of Intelligent and Cooperative Information Systems (ICIS).

[16] In [74, 75] a *node* or *database node* may contain a collection of databases.

represents and can be referred to another GC with a weak link. In order to allow a consistent degree of granularity, the system may allow a dynamic merging and splitting of the GCs and updating of the link weights. Associations between nodes and GCs in the node knowledge-base are represented in the form of weighted facts.

There are several phases of negotiation (node interaction) in the approach being presented. The first phase consists in the formation of clusters or nodes based around areas of interest (GCs). The second phase involves the search of the inter-node relationship space (resource discovery) by search heuristics. Thus, when a database needs to search an object it can start by looking first into the databases that are strongly linked to the same GC that it belongs to. After this initial phase, the search is executed using one or a combination of the heuristics explained below.

In the *node based* heuristic the search is performed based on the link weights with which a node is related to the various clusters. That is, the search is executed in the nodes of the clusters from the stronger to the weaker links.

In the *concept based* heuristic the search is accomplished using the GC's view of node groupings (not the view of the node). Each GC has knowledge of nodes in all clusters and the search is based upon the strength with which the various clusters are linked to a GC (concept server).

The *token based* heuristic is characterised to have a number of nodes involved in the organisation of the search. First, a node identifies the closest cluster and explores it. Then, the control of the search is not remained with the node or the related GC. It is given to a node in the cluster that consults the GC with which it has its second strongest link. So, the search is performed from node to node in a *breadth first* manner. To avoid cycles (the exploration of a node for a second time), the information of which node has been already explored is passed from node to node along the search.

The other phases of negotiation include: (a) determining the access rights (restrictions) to remote node information, (b) delivery of integration/interoperability facts that are stored in the knowledge-base, and (c) reviewing of the weights of the links between a node and the GCs. A node can disseminates information either by broadcasting its new fact to relevant node knowledge-bases, or contacting and interacting with relevant nodes in a cluster in a background fashion. The original knowledge for interoperation is provided by seeding knowledge, but grows as queries are being processed, resulting in a dynamic assembly. The incorporation of facts may be performed either automatically or semi-automatically. This depends on the existence of sufficient local knowledge to resolve ambiguity.

In the propose approach there is a centralised initialisation phase where the databases are registered, building the system. After this phase, the system "grows" by the automatic merge and split of the existing global concepts and the update of link weights. The updating of global concepts and link weights are executed by concept server processes, running in some predefined nodes. A concept server process indicates whether the number of nodes clustered around their GCs exceeds (falls below) a specific value. In the splitting process, new groups based on node link weights are determined and new concept servers created. The merging process is based on link weights to locate the appropriate concept servers. Thus, the link weights assigned

during the initialisation phase is constantly updated. After the merging/splitting process, messages are sent to update the concept server address and GC-node link weight information, possible causing effort on communication. However, in [74, 75] the authors do not specify how new databases are added into the system and how existing components are removed. They assume for simplicity that all nodes in the system are registered during initialisation, that is, there is neither addition of new components nor remotion of existing components.

We affirm that it is a composite task to maintain the update of the clusters and the node knowledge-base, specially in the situation where a database is free to enter or leave the federation whenever it wants to. It is suitable to apply the idea of grouping databases, trying to make the universe of search smaller. In this case, the search can be guided by the different types of groups. Nevertheless, we believe that the criterion of clustering based on the area of interest is not sufficient, due to the fact that these groups may still have a great number of components. Notice that for each different query, containing a distinct area of interest, it forces the creation of a new cluster. It is also necessary to specify criteria for the link weights, that is, which values must be given and in which situation.

2. **InterBase**

   The InterBase system [19, 34, 35] is a prototype developed at Purdue University to integrate pre-existing systems in a distributed, autonomous and heterogeneous environment. InterBase can be used to integrate not only database systems but also non-database systems. It supports system level integration instead of logical level integration like MERMAID [14, 91, 104], DATAPLEX [14, 26, 104], MULTIBASE [14, 23, 91, 97, 104], MRDSM [14, 68, 69, 91].

   In the system the user can either write *global applications* (global transactions), that are translated to a uniform language called *InterBase Parallel Language* (IPL), or write their queries directly in IPL. IPL permits *Flex Transaction Model*, where the correct execution of a transaction is beyond the traditional either *all* or *none* execution. It also allows *time constraints* on subtransactions, improving the flexibility of subtransaction scheduling. The model admits that users decide which of its subtransactions (subtasks of a global application being executed) can be committed or aborted in the final stage of a transaction.

   The global transactions are sent to a Distributed Flexible Transaction Manager (DFTM) that interprets and coordinates their execution over the entire system. For each global transaction, DFTM generates an image of itself allowing various global transactions to run concurrently. Each image disappears after the execution of the related global transaction. The concurrency execution of the global transactions is achieved by the Distributed Concurrency Controller (DCC). It determines the global transactions that can be executed simultaneously and the transactions that must wait until specific conditions are satisfied. The DCC uses the theory of Quasi Serializability[17] because serializability is incompatible with the preservation of local autonomy (see subsection 2.5). However, this theory defines a weaker consistency criterion than the traditional serializability.

[17]Quasi Serializable schedule was proposed by W. Du and A. Elmagarmid (1989) and consists in serializabling local transaction schedules and serially executing global transactions.

For each component of the system, named *Local Software System* (LSS), there is a *Remote System Interface* (RSI) located between DFTM and LSS. The RSI translates a command from DFTM into a format understandable by the LSS and sends it to the LSS to be executed. Thereby, the autonomy of a LSS is preserved. Notice that the interfaces between DFTM and RSIs are similar, and between RSIs and LSSs are dissimilar. Each RSI coordinates the execution order of the subtransactions in the LSS that interfaces with it.

RSI supports client/server model for LSSs, that is, LSSs can be treated as servers that provide services for subtransaction. It consists of two components: RSI server and RSI services. The RSI server receives requests from global applications to execute their subtasks, interacts with these global applications to arrange a subtask order and, based on this order, schedules the execution of the subtasks. The RSI service is created by the server and implements the execution of a subtask. This division permits subtasks of different applications to be executed at the same time on the same LSS (multiple RSI services created by the same server). The RSI server can also run in different platforms of its corresponding services and the system permits some standardisation of RSI servers.

The system contains a *RSI Directory*. It provides users with location and distribution transparency of RSIs and necessary connection information to global applications.

The system allows decentralisation of the resource discovery by creating a copy (image) of the RSI Directory and DFTM for each global transaction. Although this approach avoids the problem of performance bottleneck, the creation of these copies can be very expensive. On the other hand, all of the other problems that appear when using a centralised structure (updates, failures of the structure, no privacy of information, and so on) can be presented. The use of a uniform language as IPL is not very useful since this can generate unnecessary work in translations and retranslations of queries. In [19, 34, 35] it is neither clear and specified how DFTM divides a global transaction into subtransactions, nor how it identifies which LSSs are involved in the execution of this global transaction, nor how it deals with the problem of resolving conflicts and unifying the results of a global transaction. Notice that InterBase approach does not distinguish the public data from private data. It also does not specify semantic information related to the component data.

3. **ICCS - Agents**

In [81] Papazoglou et al. proposed an approach to the emerging field of Intelligent and Cooperative Information Systems (ICCS), trying to solve the problem of integrating heterogeneous information systems (ISs)[18]. That is, operating units (information sources) that are similar in functionality and goals, but differ in design and implementation. They affirmed that to achieve interoperability and cooperation it is necessary to resolve the problem of semantic heterogeneity. This is done by allowing the ISs to contain sophisticated knowledge-based capabilities.

[18]Papazoglou et al. defined information system as a conglomerate of applications that implement the necessary functions over a collection of data and knowledge. It consists of a conceptual schema. That is, a formal description of the structure and semantics of the part of the universe represented by the data/knowledge. In this part of the text we use IS interchangeable with *components*.

The goal of ICIS is to find a decentralised approach to the next generation information systems. The approach should imply that knowledge, control and data are logically and spatially distributed. The approach must have neither global control nor global data storage. No component has a global view of the problem domain and of the activities being performed in the system. To perform this goal they proposed the use of a community of *information agents*. The information agents are the ISs wrapped with a "skin", which make them "conversable" with each other, and allow the automatically execution of a common task without human intervention. Thus, equivalent to a logical front-end.

The information agents are organised in groups in terms of their subject areas[19] and have two roles. In the first role, named *retrieval expert*, the agents help the user to formulate the correct request, to find the appropriate information sources and to retrieve the information from these sources. In the second role, named *knowledge-driven system*, the agents maintain knowledge about the application domain, the structure and semantic of individual sources, the method of cooperation (communication) between them, the translation methods required to homogeneize heterogeneous data, and so on.

Due to the functions and capabilities of the information agents, it is necessary to have a set of high-level modelling tools. This is performed by creating a common knowledge-level model (KLM). The KLM must help with the homogenisation of the ISs, must provide that knowledge and data are represented in a unified representation schema, and must permit conditions to support inter-agent communication and cooperation. Thus, a knowledge model of an agent is a collection of tools for describing data/knowledge properties, object relationships, domain semantics, inter- and intra-object constraints and general means for inferencing. The KLM may be materialised by the use of object-oriented/knowledge-based technology. The complete knowledge of an information agent can be modelled by the definition constructs of the KLM. The definition part (schema) of the domain knowledge of an agent is represented as a hierarchy of knowledge-objects. The assertion and inferential parts are represented through a set of rules embedded into the knowledge-objects. The agent schemas developed in KLM describe not only data of the various systems, but also the meaning or semantics of information, rules for their composition and semantics of application domains. The requests for information are made in the KLM data language. This forces the KLM queries to be translated into the specific model and data language of an agent involved in the process.

Each information agent has a *knowledge directory*. The knowledge directory provides a decentralised way to locate the correct agents and contains the necessary information to resolve a problem. Nevertheless, it seems that it is a complex task to update and maintain the knowledge directories in a dynamic environment, where components can join and leave the system constantly. The information agents contain two types of knowledge: *autoepistemic* and *epistemic*. Autoepistemic consists of the knowledge that a component has about its own capabilities. Epistemic refers to meta-knowledge capabilities and behaviour of other components. Epistemic knowl-

[19] This idea of clustering the information agents by means of their subject areas seams to be similar to the approach proposed by Milliner and Papazoglou [74, 75], where the components of a federation are defined using Global Concepts.

edge is divided into *group* or *cluster knowledge*, and *global knowledge*. The group knowledge consists of the information shared by a community of information agents which is dynamically formed. The global knowledge consists of the information that is not into the autoepistemic and group knowledge. It is less detailed than the group knowledge. When a query cannot be handled by an information agent and the agents in its group, the global knowledge is used to identify the information agents that cooperate with the current agent.

When receiving a query an information agent may use its autoepistemic and epistemic knowledge to resolve it. It is assumed that an information agent has sufficient knowledge, resources and capacity to divide a query into partial queries and to match them with information resources of different information agents. This process is called *task* or *problem decomposition* and needs to be better detailed. It is not specified how the information agents use its knowledge, capacity and resources to achieve this task. This also forces the requirement of sufficient knowledge, resources and control in all the system to allow a good solution of the queries. Another process related to the problem of resolving query is the *negotiation*. It consists in the sequence of committing each information agent to perform a series of problem-solving actions. In the current approach the component that receives the query acts as its *manager* (*primary agent*), using "contracting" information agents to resolve the partial queries into which the original query was decomposed. Each partial query corresponds to a single contracting agent that may act as *master agent*, having to report back to the primary agent. So, master agents integrate the results produced by its contractors forming a partial solution. The complete solution is formed by the primary agent. In this case, we affirm that there is the problem of defining how different subtasks are related to each other, requiring strategies for ordering sets of subtasks and creating partial results.

The queries are decomposed into *subqueries* and *negotiable queries*. The subqueries (negotiable queries) correspond to the tasks that are handled by components in the same (outside the) cluster of the primary agent. The primary (master) agent generates a plan which consists of the local organisation of commitments for the agents involved in the computation. After the generation of the plan, each agent is responsible for two kinds of commitments: *shallow* and *deep commitment*. These commitments result in subqueries and negotiable queries, respectively and are removed in the end of the plan.

The solution of a query can be performed by accessing other agents recursively, forming a chain. These agents are chosen and specified based on the autoepistemic and epistemic knowledge. The approach of passing a query (or subqueries) to other agents that can possible know where to find the correct information is suitable to the discovery process. However, it can generates the problem of having a data of a component accessible by all components in the system. There are cases where a component wants to share its data only with a specific group of components in the system, but not with all the other components. It is also necessary to use a mechanism to avoid cycles.

4. FINDIT

In the PhD thesis of Bouguettaya [7] and in [8, 9, 10] a decentralised approach was proposed to achieve interoperability among a large number of databases. It is a two-level approach based on the idea of *coalition*[20] (first level) and *service* (second level), allowing databases to be grouped into tight and loose conglomerates. The goal is to allow the databases to know dynamically what other databases contain. It is performed by a proposed framework and system called FINDIT. The first prototype for this approach is currently operational.

A coalition is a group of databases that share some common interest about an information type. A service is provided as a means to share minimal information description. The services can be between two coalition, between two databases, and between a coalition and a database. Notice that a database can simultaneously be member of different coalition and also a servicer. Coalition and services use *single information type* as basic unit of sharing. They differ from the federated architecture [48] (see subsubsection 2.3.1), where the unit of data sharing is a whole schema containing *all* information types. The databases organise themselves based on information interest and optionally in geographical proximity. A coalition involves an arbitrary number of databases that share certain information type with all other databases of the coalition. It provides more information with a relatively high overhead. On the other hand, services provide less information with a less overhead. The exchange of information in a service is directional, that is, two databases can be entities of a service, but only one of them provides information to be shared. As denoted in [7], unlike the federated [48] and the Remote-Exchange [37, 38, 45, 46] approaches, the FINDIT architecture permits the distinction between components that are participating (consulting), from the components that are sharing information with other databases. That is, the use of coalition and services and the non-existence of a centralised structure with the export schemas of the participants, permit the presence of a component that only consults other components.

Each participating database contains an object-oriented *co-database* storing information about coalition and services that it is involved on. The co-database is used by a component to allow the users to know about the information that they may have access to and about the database itself. A co-database contains subschemas representing coalition and services. Each of these subschemas consists of classes. These classes store descriptions of the databases that participate in those coalition and services and of the type of information that they contain.

In the system, the answer of a query is executed in two phases. The first phase consists in educating users about the space of information and finding the target databases that are most likely to have the type of information being searched. The second phase consists in dynamically specifying and querying the actual information[21]. The author assumed that only one information type may be requested within one query.

The system uses information type name, structure, behaviour and graphical representation to achieve the goal of the first phase. The framework makes use of a

[20] Database coalition has equivalent notion of *political coalition* presented in democracies, where a group of people stay together based on some common objectives for a limited period of time.

[21] This phase was not focused in [7, 8].

special query language called TASSILI. It has constructs to educate users about the space of information and formulate their queries. With TASSILI it is possible to connect databases and to perform remote queries. It is a data definition and a data manipulation language.

The resolution process of a query consists in the dynamic education of users about the space of information and databases know the information that other databases contain. It is performed interactively with the user who is responsible to clarify and resolve any doubt of the system.

FINDIT uses a *documentation* for each information type to solve the problem of understanding a foreign information in a different component environment. The documentation is shared with other components. It consists of a demonstration of what an information type is and what it offers.

The execution of a query maps a set of information type names to a set of classes. The first set contains synonyms of an information type and the second set has names that are either a coalition or a servicer class. The default is to try to conduct a query first in a coalition class, and in the coalition that contains the database where the query was performed. A servicer class is chose only when there is no coalition class name available.

When a query cannot be resolve in those coalition and services that the component is member of (failure case), then the database tries to find another possible component that may have the information. This can be done in two different ways. One way is to verify in its co-database if there are coalition or services that know about the information. The other way is to request a subset of databases in the same coalition where the database is member of, that know about other coalitions that they participate that possible contain this information. In the case where there is no coalition, the database tries to find a service with other databases that know about the information. Whenever a database that seams to have the information is discover, then the query is sent to this database and the process of resolution executed. Otherwise, the query is considered to have failed.

Notice that the idea of sending the query from one component to the other was also proposed by Takizawa and Hasegawa [102] and by Papazoglou et al. [81] (primary and master agents). For a discovery process in a decentralised environment this is a good approach. However, there are cases where a component wants a restrict group of components to have access to its data, because of security and protection reasons. In FINDIT, the way in which the information is discovered does not avoid other components, outside a restrict group, to access this data. Other coalition and services are eventually searched to identify the information. Thus, to guarantee protection of the information it is necessary to check if the database containing the requested information can tell about this information to the user.

The failure of queries can be considered as a suitable task to the system. It increases the system knowledge about other information that it does not know. The resolution of queries out of the local database domain interest extends the system. In this extension new coalition and/or services are formed. FINDIT also permits the addition and deletion of components, providing a scalable system. However, in [7, 8, 9, 10] the authors deal with the scalability aspect from the situation where it

is already known the correct coalition and service related to that component.

In our opinion, the high interaction with the user during the resolution process of a query does not guarantee transparency of its execution. Notice that in many cases the user is responsible to guide this resolution process, giving necessary information to the system to locate the correct data. Another problem is related to the difficulty of manipulating with queries that contain more than one type of information. This is a normal situation when dealing with heterogeneous databases. However, in FINDIT this makes the resolution process a very complex and confuse task to the user. It is necessary to break the query into subqueries, to try to resolve each of them with the help of the user and, in the end, to place the results together. In [7, 10] the authors outlined that it is necessary and important to complement FINDIT to allow the direct use of the actual data after locating it. The resolution of conflicts and the unification of the located data is not yet defined in FINDIT.

2.3.6 Conclusion

Although the area of heterogeneous databases has been hardly examined and various approaches proposed it reminds some important problems not yet solved. One important problem is related to the information discovery of a data, in particular when dealing with a great number of components. Most of the existing approaches assume that the system knows where the correct data is located. The problem of information discovery is related to the fact that during the resolution of a query the system does not know where the correct information is located and which are the components that have this information available.

When dealing with a vast collection of databases it is not suitable to use a centralised structure to help with the execution of the different phases of the interoperability process (information discovery, conflict resolution and unification of the results). The use of a centralised structure does not guarantee autonomy of the involved components; generates difficulties in executing some operations like addition, remotion and modification; does not assure privacy of the information; and does not guarantee good performance in the execution of the different processes (performance bottleneck). When using a centralised structure it is also necessary to solve the problem of failures. On the other hand, the idea of either broadcasting for all of the components or performing queries that visit all the databases to locate and/or identify correct data (resources) is also impractical. Both of these approaches can generate a great amount of traffic in the network. Notice that these ideas can also lead to the generation of a large amount of unnecessary translations. That is, a query is sent to various components that does not have the requested data and that contain data models different from the data model of the component where the query was performed. Therefore, it is necessary to propose a way that allows all of the phases in the interoperability process to be done neither using centralised structures nor broadcasting for all of the components. The proposed approach has also to permit new and existing components to join and leave the environment without difficulties. That is, it has to guarantee the scalability of the system.

We agree with approaches like: Global-Concepts [74, 75], ICCS-Agents [81] and FINDIT [7], where the databases are grouped in order to diminish the universe of search. Therefore, it is necessary to determine the criteria to be used to form the different groups.

Another complication when dealing with different autonomous databases is concerned to heterogeneity. It is not easy to make a query understandable to different components (data models). One important aspect to assign with the lack of semantic and behavioural information and with the inability of reason about semantic differences among the different data models. Therefore, it is essential to understand, capture and deal with the semantics of the elements. It is also indispensable to extend the mechanisms of defining type specific behaviour (procedures, functions, or methods), to identify similarities of the data and to extend the data models to include more semantic information about the meaning of the data. These are very important aspects to permit the detection of the correct data and the assignment of conflicts that may appear.

In our opinion, the use of an additional structure for each component containing semantic information of the shared data (Remote-Exchange [37, 38, 45, 46], InHead [110, 111] and ICCS-Agents [81]) is reasonable. However, it is necessary to determine a way to define, represent and manipulate with the semantic aspects of the data.

We also defend the idea of separating the data that a component wants to share (public) from its other data (private). Nevertheless, it is important to determine the type of data to be public and how to represent and access this data. The use of a recursive procedure to execute the information discovery process, starting in a certain point (component) and moving to other one, depending on the information found inside the last component visited, is feasible. However, it is necessary to avoid unauthorised access of data and cycles during the process.

The inclusion of a certain data in any requester component is another feature that requires specifications. It is not easy to perform unification of data, as automatic as possible, preserving the components autonomy and without disturbing the original data stored in the involved components.

The managing of heterogeneous databases has to guarantee that the users of the different components are able to perform both read and write (consult and update) operations, in a transparent way for the users and applications, preserving the autonomy of the components and guaranteeing privacy on the data. It is also necessary to permit the participation of databases in the system that share and exchange data, and of components that only consult and interrogate the other members of the system.

2.4 Interdependencies

One important aspect in facilitating information sharing and exchange in federated database systems is the management of interdependencies among data in different databases.

Li and McLeod [66] affirmed that the data interdependency can be of various forms and complexities. It can goes from simple value dependencies to more complex structural and behavioral dependency relationships. They believe that data interdependency management is a sub-issue of the general area of distributed integrity constraint management. In [66] the authors proposed an approach to deal with interdependencies among data in different databases based on an object-oriented federation model and architecture named ODF [65] (*objectified database federation*).

ODF is considered to be an extension of the federated architecture [48] (see subsubsection 5.1.3), using an object canonical data model (see subsection 3.2) and applying an object-oriented methodology as a design and implementation approach. In this ap-

proach databases are inter-relatable through predefined relationships and interoperable through message-passing, which is used to exchange information and to negotiate sharing/exchange agreements. The databases are viewed as objects and names OCs *objectified components*. Each OC has associated with it a set of methods that support requests for data and/or services from a remote OC.

An OC has three areas (private, protected and public) and a derivation dictionary. The *private area* corresponds to the data that is local to the OC; the *protected area* contains data that is accessible by some selective OCs; the *public area* has the data that is accessible to every OC in the federation. The derivation dictionary maintains information of where and how imported data is derived and inherited from other OCs[22].

Li and McLeod identified four different types of interdependencies that may exist between two objects (O_i, O_j, for instance) from two different databases (DB_1, DB_2, for instances), outlined below. *Existence dependency*, where the existence of O_i depends on the existence of O_j; *structural dependency*, when structural changes of an object in one database may affect the structure of another object in another database; *behavioural dependency*, behavioural changes (modification of methods) of O_i may affect the behaviour of O_j; *value dependency*, related to the fact that changes in the attribute values of O_i can be reflected to the attribute values of O_j. It is also possible to have "non-parallel" interdependencies, that is, changes in attribute values can influence changes in methods. Another possibility is to have different combinations of the interdependencies ranging from a single one to a totally combined one.

Rusinkiewicz et al. [87] stated that interdatabase dependencies should specify the dependency conditions and the consistency requirements that must be satisfied by the related data, and also the consistency restoration procedures that must be invoked when the consistency requirements are violated. They proposed a model that allows specifications of constraints among multidatabase in a declarative fashion. The model uses a *data dependency descriptor* that provides more semantic information then constraints that are used to specify database integrity. This descriptor defines dependencies between related data, consistency requirements and consistency restoration procedures. The authors also proposed a system architecture to be used to maintain interdependent data objects in a multidatabase environment.

In this architecture it is assumed that every database has associated with it an interdatabase dependency system that acts as an interface between different databases. The dependency system of different sites can communicate with each other. There is also a centralised or distributed *interdatabase dependency schema* (IDS). The IDS is always consulted by a dependency system whenever a transaction is submitted for execution. This consult is performed in order to determine if the data accessed by the transaction depends on a data of other components. When a transaction updates data in a component which is related to data in other components, then various transactions are performed to guarantee mutual consistency of related data. These transactions are submitted to the database management systems of the correspondent related data.

We believe that the problem of representing and maintaining information about interdependencies among data in different databases has to be carefully treated, since it is very important to allow consistency during the exchange and share of data. Therefore,

[22]Comparing ODF with federated architecture [48] we observe that private area corresponds to "private schema" concept, protected and public areas to "export schema", and derivation dictionary to "import schema".

the architectures for heterogeneous databases have to contain a way of representing and managing with these interdependencies. On the other hand, it is very difficult to have information related to database dependencies when dealing with a great number of different databases (great quantity of data) that are constantly and dynamically modified. In spite of some existing approaches it is necessary to find a way of representing, manipulating and maintaining those interdependencies.

2.5 Transaction Management

Transaction management is considered to be a complex task in a federation of heterogeneous databases. In this environment it is necessary to manage two different types of transactions: *local transactions* and *global transactions*. The local transactions are directly submitted to a component by its local user, originated at this local component. The global transactions are submitted by multidatabase users, originated at a remote component. As outlined in [13, 90, 98] it is difficult to guarantee the ACID[23] and serializability properties, since each system has its own mechanism to ensure these properties. It is necessary to find some approaches that develop strategies to use different transaction processing mechanisms together such as: mixed concurrency control algorithms and commit protocols, without violating the components autonomy; global coordination of the concurrency control and recovery strategies shared by the systems; different treatment for the local and global transactions; definition of new concepts that permit the execution of transactions in a heterogeneous environment; and so on.

Breitbart [13] affirmed that it is not possible to have transaction management algorithm for a federation without any restrictions on the local database management systems. He outlined that to find such an algorithm it is necessary to impose some restrictions on either the type of global transactions that may be executed in the federation or on the structure of the local concurrency control mechanisms. In [12] the author proposed a concurrency control algorithm that ensures global serializability and freedom of global deadlocks. For this algorithm it is assumed that each local component uses the two-phase locking protocol[24], therefore violating the local autonomy of these components.

Mullen et al. [79] proved that in a heterogeneous database system it is impossible to execute atomic commitment of global transactions[25], even not having system failure, without violating the autonomy of the local components. Since it is assumed that the local components do not differentiate local transactions from global subtransactions that they execute, then in the situation where it is necessary to *undo* or *redo* some subtransactions to ensure the correctness of the atomic commitment protocol, the autonomy of the local components is not guaranteed. In this case it is possible to execute the undo and redo of a local transaction. They also proved that even when the system violates local autonomy by assuming that its components may use only strict two phase locking for concurrency control, it is not feasible to perform atomic commitment that can tolerate at least a single system failure.

A concurrency control schema that permits the local component autonomy and global

[23] ACID for Atomicity, Consistency, Isolation and Durability [23].

[24] All the locks requested by a transaction may be performed before any lock is released.

[25] The atomic commitment protocols for global transactions guarantee that all subtransactions of a global transaction are either uniformly committed or uniformly aborted.

serializability was proposed by Kim and Moon [60]. The idea is to use two different types of concurrency control schemas: timestamp ordering and two phase locking. To detect and resolve conflicts (direct and indirect) the approach uses a preprocessor for each component named LEM (local execution monitor). The local transactions are submitted to LEM to examine the effects of them to global serialisation order, and the indirect conflicts caused by local transactions are detected at each LEM. Sheth [91] affirmed that the problem of fault tolerance has received little attention and that there is also a lack of adequate transaction management algorithms to provide a specified level of consistency. Therefore, it is necessary more work to permit viable and correct algorithms to transaction management.

We believe that more work has to be executed on the transaction management area in order to develop algorithms that allow the manage of local and global transactions without violating the autonomy of the databases involved. Actually this is a very hard task and without a solution until now (open problem).

Chapter 3

The Proposed Architecture

Based on the existing approaches to interoperability of autonomous databases and on the drawbacks in these approaches, we propose a new architecture that tries to diminish these problems. Our goal is to permit heterogeneous autonomous databases, independently created and administered, to interoperate with each other, in a transparent way, and without compromising their autonomy. We aim to avoid the use of centralised structures to help with the information discovery process. The architecture aims to preserve the privacy of data, execute both retrieval and update operations, and support scalability of the system including dynamic modifications of the components. We wish to permit the situation where a certain query q' is performed in component DB_1 (source) and the answer for this query, or part of the answer, is presented in a different component DB_2 (target). Thus, the execution of the different phases of the whole interoperability process has to be automatic and transparent if possible.

3.1 Description

In the proposed architecture the component databases are arranged into *federations* as illustrated in figure 3.1. A federation consists of a set of databases willing to share data with each other. The criteria used to form a federation are based on the shared data of the components and the components that are allowed to access this shared data. Inside a federation, the shared data of a component is public for all the other components in this federation. A database outside this federation cannot access the shared data of the components in that federation. That is, suppose a database DB_1, in a federation F_1, sharing a data d_1. We desire to permit d_1 to be accessed (public) by all the other components that are in F_1. Therefore, it is possible to have a database participating in different federations (overlapping), sharing a distinct set of data in each of these federations. In figure 3.1, $DB_{11}, DB_{12}, DB_{15}$ and DB_{30} participate in federations 2 and 3 at the same time. The architecture also permits a component to participate in a federation without sharing any data, only interrogating (consulting) other databases (named an *enquirer database*). The enquirer database only accesses data of other components, but is not accessed by other components. Therefore, it is possible to have a component that participates in a federation sharing a set of data and in another federation as an enquirer. This approach permits structures which support data confidentiality and security.

The idea of having the same component in different federations, depending on the data that it is sharing and on the other components that have access to this data, is based

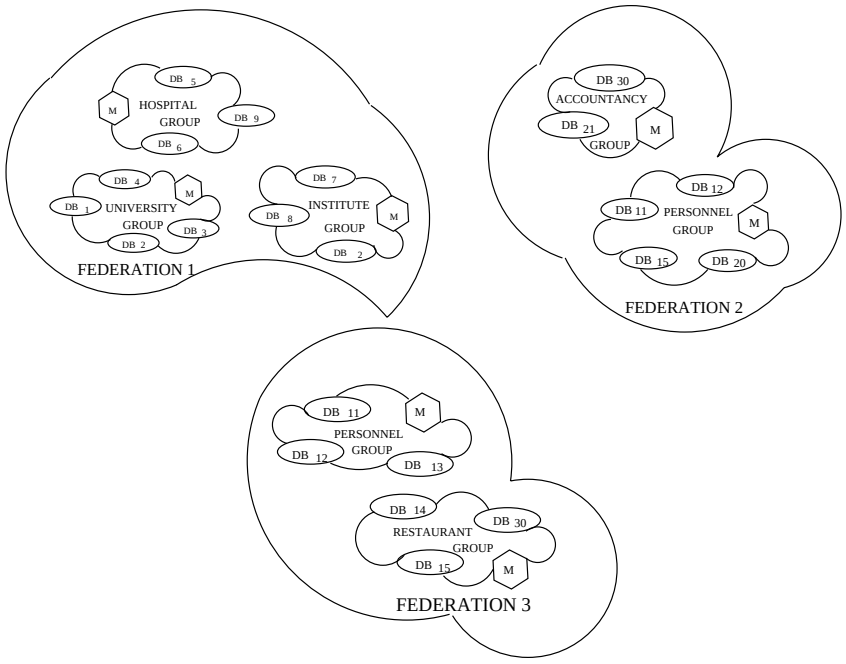

Figure 3.1: Federations of heterogeneous databases

on the concept of avoiding a specific data to be accessed by components not allowed to. Thus, guaranteeing better confidentiality and security of the information. Notice that in the approach proposed by Bouguettaya [7] the idea of using coalition and services, and the way that a data is searched, can lead to the situation of accessing a certain component not belonging to the coalition that contains the requester component. Thus, to guarantee confidentiality and privacy of the data it is necessary to check if the database that contains the requested information is allowed to give this information to the user.

To illustrate the use of federations consider a holding formed by four different companies. Each company contains independently created *personnel databases* having information related to the employees and their salary histories. Consider two different cases. The first case dealing with the accountancy of the holding. The second case assigning the background of the employees (their different types of job). In our approach we create two federations. One federation contains the personnel databases sharing payroll information. The other federation encloses these databases, but sharing the different types of job. It should be unreasonable to make public the salary of the employees when dealing with a different matter.

As proposed by Milliner and Papazoglou [74] we group the components of a federation in order to make the universe of search smaller, thereby facilitating the process of information discovery. In our approach, inside a federation a *group* of components is formed based on the type of data that these components share. For instance, we could have a *university group* (formed by databases sharing information about universities), a *hospital group* (containing databases sharing data about hospitals), and an *institute group* (having databases exchanging data of institutes) forming a federation, as presented in federation 1 of figure 3.1. Since a certain type of data can be related and classified in two or more different areas we can have a database that participates in more than one group in the same federation (DB_2 is related to university and institute groups). The public data (set

of data) of a component in a federation is shared with all the other components in the federation and not only with the components composing its group. It is also possible to have the same group name classifying two different groups in two distinct federations .

Each database has an *administrator* (DBA) to assist with the organisation of the system. The DBA decides about the different data that the component can share, which are the federation that it participates and the relationships with the other components. On the other hand, each federation contains a *manager*. The manager supports the decision about the different groups inside a federation, the terms associated to these groups, and the common data model and language used to facilitate the interoperation among the different components.

We suggest the use of a special structure named Syst-DB containing general information about the whole system. This information is related to the different federations existing in the system, the various groups in each of those federations, the participant databases, the different data shared by each component in the federations that it participates, and so on. The idea of maintaining a structure like this is to assist on the scalability of the system. That is, support the correct insertion (registration) or deletion of either a component or a group of component, and the creation and removal of federations (see chapter 4). Without using a structure like this it is impossible to know the best way of organising the system. It is also difficult to guarantee the functionality, reliability and correct operations of the system according to its specifications. The Syst-DB works as a "mirror" reflecting the whole situation of the system.

We propose to implement the Syst-DB as a centralised relational database. Figure 3.2 presents a schema for the Syst-DB, expressed in the entity-relationship model, with its entities, relationships and respective attributes (the keys are represented by underlined attributes). Chapter 4 contains details of the functionality and operability of the Syst-DB[1].

Notice that in the approaches that make use of a centralised structure containing information about all the databases in the system (federated architecture [44], Remote-Exchange system [37, 38, 45, 46] and InHead [110, 111]), it is not necessary to know about the other existing components before adding a new database[2]. In this case, the interoperability process is performed based on the centralised structure. On the other hand, the approaches that organise their components in either coalition, or clusters, or groups, do not specify a way of deciding about the best and correct place to insert a "new" component. In FINDIT [7] they deal with the scalability of the system from the situation where it is already known the correct coalition and service related to that component. However, before adding a new component, it is not specified how to get knowledge about the coalition, services and databases existing in the system. The approach proposed by Milliner and Papazoglou [74] contains a centralised initialisation phase where the databases are registered, building the system. After this phase, the system evolves by the automatic merge and split of existing Global Concepts (GCs) and the update of link weights. The authors assume that all nodes in the system are registered during the initialisation phase. No new nodes join and no established nodes leave the system.

[1]Notice that even using a centralised structure having general information about the system to assist in the scalability process, this structure is not used during the interoperability phases, in particular the discovery process of a certain data. Our goal is to permit a decentralised discovery process.

[2]Record that the use of a centralised structure generates other drawbacks like: performance bottleneck, difficulty of updating, failure of the structure, no guarantee of privacy and confidentiality, and so on.

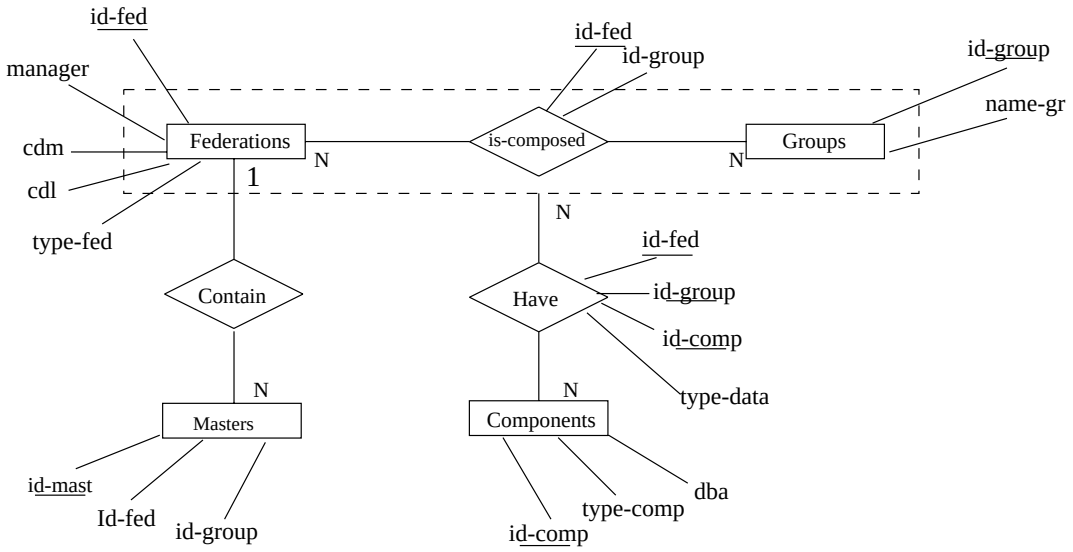

Figure 3.2: A relational schema for the Syst-DB

As presented in figure 3.1 each group contains a component named the *master* (M) that records the location of the different components existing in a federation. The master controls the process of adding and removing components. It permits a component (or a group of components) to join or leave a federation without stopping the whole system and notifying this to the other remaining components (see chapter 4). The master is also important in the discovery process since it contains the more up to date information about the existing components in its group (see chapter 5). During the discovery process, the master is consulted when data is not found in any of the components in its group, helping to identify a database, not yet visited, that possibly contains this data. Thus, when the masters have no knowledge about components containing the searched data, it is guaranteed that either no component in the system has this knowledge or the data does not exist in the system. The type of information that the master holds and the way that it is structured is the same as the LOC structure specified below in item 4.

Before adding a new database into a federation some preparation has to be made. Simpson and Alonso [96] affirmed that: "an autonomous database should have the freedom to present itself differently to different users, or to refuse to provide service or information about itself to certain users." Therefore, apart from the notion of federation that guarantees privacy of information, we propose the addition of new structures to each database. These additional structures guarantee autonomy and transparency of the databases. A component should be able to accommodate information necessary to achieve interoperability without modifying the original structure and preserving its autonomy. The new structures of each component, LPS, MLPS, SPD and LOC, are related to each federation with which the component participates. Hence, for each federation a component possesses a different set of these structures. Figure 3.3 shows a database prepared to join a federation with all of its additional structures. In this example the database DB_i participates in two different federations, j and k. In the following, we explain the need and functionality of these additional structures.

1. **LCS/LPS**

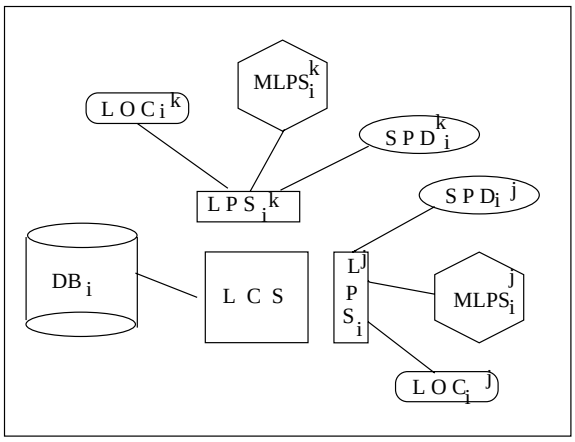

Figure 3.3: A component database with its extra structures

We use the idea of the five-level [91] and federated [48] architectures of separating shared data from private data. As can be seen in figure 3.3, the non-public data (private) of each component is represented in the *local component schema* (LCS). The LCS is defined in the original data model of the component and is the original schema of the component. On the other hand, the information that a component shares with the other participants of its federation is called *public data* and is registered in a special schema called *local public schema* (LPS). The LPS is specified in the same data model of its component. Notice that this new approach differs from the existing methodologies since it permits each component to contain more than one LPS, one for each federation, distinguishing the shared data for each federation. It is possible to have overlapping information in the different LPSs of a component, i.e., common data in these different LPSs. The use of various LPSs prevents the allocation of all the shared data of a component in the same place and guarantees better confidentiality and privacy of information. In the existing methodologies all of the shared data of a component is stored into a special structure (for example, export schema in the federated architecture), becoming public for all of the components in the system. In the federated architecture the unit of data sharing is the schema. In the propose architecture the unit of data sharing is also a schema (the LPS), but containing specific data to be shared with a federation, not all the exported (public) data. To illustrate this, consider the situation of 3 different databases, DB_1, DB_2 and DB_3, with DB_1 willing to share a certain data d' with DB_2 but not with DB_3, and another data d'' with DB_3 but not with DB_2. For the existing methodologies both d' and d'' are presented in the export schema of DB_1. To avoid the access of d' by DB_3 (d'' by DB_2) it is usually necessary to use some special way of making the information secure, by introducing additional information to the data. In the proposed approach, with the notion of different federations and LPSs for each federation, security is automatically preserved, e.g. federation 1 containing DB_1 and DB_2 with d' in LPS_1^1, and federation 2 having DB_1 and DB_3 with d'' in LPS_1^2.

2. **MLPS**

   Associated with each LPS there is a *mirror local public schema* (MLPS). The MLPS contains the public data of the component expressed in either a common (canonical)

data model (intermediate) for all the federation, with a corresponding common data language, or in an intermediate meta-model [4, 82][3]. The choice of the data model used to describe the MLPSs of a federation depends on the data models of the components in the federation. Thus, two different federations could use distinct canonical data models to express their MLPSs. Unlike the existing approaches, our architecture permits the existence of the shared data in the local data model and in a canonical data model previously chosen. The advantage of having public data of each component represented in two different data models is twofold. First, the LPS is used to avoid unnecessary translation work in the situation where it is known that the data sharing is made between two components that use the same data model for their local component schemas. In this case, it is better to use the original data model (direct interoperability) than another one playing the role of an "intermediator". Second, the MLPS is used to maintain a linear number of translators when performing interoperability among databases with different data models (indirect interoperability). That is, without using an intermediate data model (meta-model) it is necessary to have mappings between each type of data models in the federation for all different data models. The MLPS also permits the interoperability among heterogeneous components by detecting interdatabase semantic relationships, and bridging the gap between the different data models of the components. The MLPSs preserve the autonomy of the components.

3. **SPD**

We propose the use of a structure called *semantic public data* (SPD), local for each component and federation, to facilitate the task of identifying similarities, equivalences and conflicts, when discovering and integrating data. The SPD expresses and describes semantic information of the public data in a different way from the LPS and MLPS (not as a data model). In the Remote-Exchange approach proposed by Hammer et al. [45, 46] and in the InHead architecture [110, 111], they suggest the use of a local structure, for each component, containing a certain semantic of the data (local lexicon and data/knowledge packets, respectively). However, these structures are used with a centralised structure containing the relationships between the data of the federation. In our case, we are avoiding the use of any centralised structure to assist in the interoperability process. Therefore, all the information necessary to resolve semantic and syntactic conflicts and to identify similarities and equivalences is presented in the SPD. The semantic information in the SPD is peculiar to each component and not for all data in the system. This helps to preserve the autonomy of each component. Therefore, whenever a database wishes to change its sharable data and/or change the meaning of this data, it can perform these tasks locally without modifying or interfering on the other databases of the federation. Another function related to the SPD is to assist in determining when a component contains particular information. That is, the SPD is responsible for the identification and specification of the requested data in a component.

[3] A meta-model is a formalism for the definition of models. The use of an intermediate meta-model is suggested to avoid the complex task of choosing a suitable canonical data model from the existing commercial data models, and to avoid the difficulties in translating different data models into a central more expressive data model.

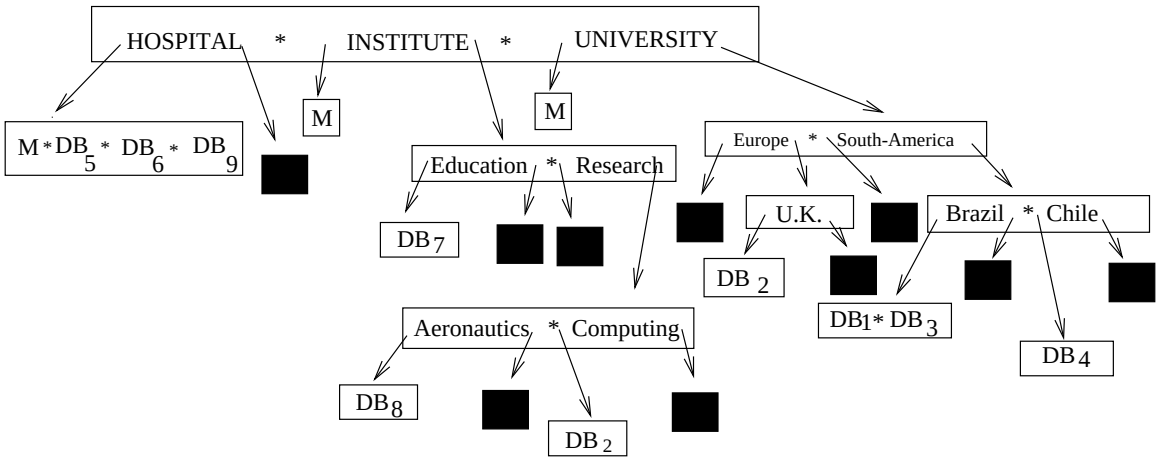

Figure 3.4: Hierarchical structure - LOC and M

4. **LOC**

The information discovery process can be performed with the use of a hierarchical structure named *locate* (LOC)[4]. In this case, we avoid both the use of a centralised structure in the whole system, containing information of the shared data and its relationship, and the broadcasting to all the components of the system. This structure records the position of the different databases existing in a federation and maintains (organises) the components grouped by the type of information being shared.

The hierarchical concept of the LOC is different from that of co-databases proposed by Bouguettaya [7]. The co-database of a component contains information about coalitions and services of its component and descriptions about the databases presented on the coalition and services. Our hierarchical structure organises all the databases in a federation by *specialised terms* (names) depending on the type of information that a component is sharing. Each group of components in a federation is divided in virtual subgroups[5], depending on the specialisations that can be described for the components of a group. These specialisations are defined based on the type of shared information.

Figure 3.4 presents the LOC structure for federation 1 in the example of figure 3.1. The first node in the hierarchy (root node) contains the names of the groups inside federation 1. The internal nodes contain the terms used to express specialisations of these groups (virtual subgroups). Notice that a node in one level contains more specific names than the terms existing in its parent node. Thereby, the structure consists of a natural hierarchy from general terms to more specific terms.

Inside a node the terms are strings of characters defined by DBAs, separated by a special mark (*)[6] and organised in alphabetical order. Associated with each term

[4]Note that the master and the LOC are the same structures. They differ only in the fact that the master can contain more up to date information. In this part of the text we use only the term LOC.

[5]The term *virtual subgroup* stands for the fact that the different groups are not physically divided into subgroups meant by specialisations. They are divided from the organisation point of view based on some specific terms.

[6]Our suggestion is to implement this special mark as 1 byte with all of the bits equal to "zero", or

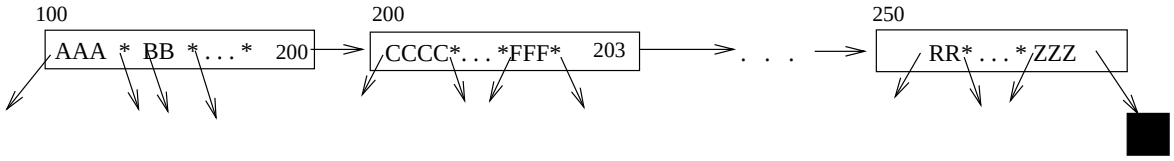

Figure 3.5: A chain of nodes containing specialised terms of a certain name

there are left and right pointers. The left pointer either points to a leaf node or is undefined (nil), represented as black square boxes in the figure. The right pointer either indicates an internal node holding specialised terms or is undefined (nil). The contents of the left leaf node depends on the position of this leaf. The left leaf node related to a term in the root node can contain two different types of information: (a) the address of the master of the group associated with this term (represented as M), when the term is specialised (the right pointer is defined); (b) the address of the master together with the addresses of different components in the group (represented as $DB_i, 1 \leq i \leq n$, where n is the number of different components in the system, and separated by (*)), when the term is not specialised (the right pointer is undefined). The left leaf node related to a term inside an internal node: (a) can be undefined, when the term has specialisations (right pointer is defined); (b) can contain address(es) of component(s) related to this term separated by (*), when the term does not have specialisations (right pointer undefined).

In the case where a certain query needs to access some components by a more general term and this term is specialised, this can also be performed by visiting all the left leaf nodes of the branch related to that term. For instance, consider a query that asks for the total number of lecturers in Computer Science in all of the Universities in the world. In this case DB_1, DB_2, DB_3 and DB_4 (see figure 3.4) have to be visited.

Each node corresponds to a disk page, thus the limit of information stored in a node is the size of a disk page. When a node in the hierarchy contains a great amount of data (various types of specialisation for a term) and the size of a page is not enough to accommodate all of the data, then the last information in this node (the one more in the right) is the address of another disk page. This disk page contains the rest, or part of the rest, of the terms, forming a chain, as shown in figure 3.5. We affirm that this is a very uncommon situation since supposing a disk page with size of 2 Kbytes, each pointer occupying 2 bytes, the special mark using 1 byte, and each name possessing 35 bytes, it will be possible to store 50 different names into each node.

The proposed structure is unbalanced. It depends on the level of specialisations that can be achieved for each group of the federation. In our opinion, the use of a balanced tree has limitations on the natural hierarchy of the terms used to classify and organise the structure. The decision to use a hierarchical structure is because of the fact that a tree permits better performance in the search process and it is a natural way of expressing the hierarchy of terms.

---

equal to "one".

In [3, 96] the authors proposed the use of an *external index* to help on the information discovery process. The external index is a partial index of external resources locally for each node. It contains network addresses of source nodes together with condensed descriptions of the information that these sources hold. In [3] they suggested the idea of having a tree of keywords for each site where this tree contains references to the information that the local site knows and the names of sites that are good sources for this information [7]. Each keyword in the tree contains attached weights to help to match a query (the queries have weights associated with their keywords as well). The authors affirmed that the main problem when using a full n-level tree (balanced) to store information is the lack of a natural hierarchy to place the keywords. In our opinion the use of weights is another difficulty since it is necessary to specify criteria not only on how to determine the keyword weights, but also on the query weights. Notice that unlike the approach presented in [3] we are suggesting the use of an unbalanced hierarchical structure that classifies the different databases in a federation in a natural hierarchy, going from general terms to more specific terms. Our hierarchy contains only addresses of the databases related to the terms.

The proposed architecture allows a component to join or leave a federation without stopping the whole system. The other components are updated assisted by the master. Even the LOC structure containing information about other components in the federation can be simple updated. Notice that it is also possible to modify the public data, which has to be reflected into the LPS, MLPS and SPD. This is done without affecting the other components of the federation.

In the architecture, an enquirer database is not addressed in the LOCs of the other components in a federation. Thus, no component tries to access it. On the other hand, the enquirer component has a LOC structure for discovering components to consult, but it does not contain the LPS, MLPS and SPD structures. Whenever the enquirer database decides to share some information this can be easily executed by adding it in the system as a "new" component (see chapter 4).

[7] In [74] the authors affirmed that the approach proposed in [96], of using external indices tends to centralise the search within a single index. However, we do not agree with this statement.

Chapter 4

Initialisation Process and Scalability of the System

A fundamental question when dealing with autonomous heterogeneous database systems is to permit a dynamic evolution of the system. It is important to allow databases to join and leave the system without stopping it and without causing many modifications to the other databases. In this chapter we outline how to initialise a system composed by autonomous heterogeneous databases using the proposed architecture. We also guarantee the scalability property by identifying all the possible cases.

4.1 Initialisation Process

This process is related to the construction of an interoperable system containing different federations and components. The construction of an interoperable system is performed whenever a group of components want to share and exchange information among each other. In this case, the DBAs of the involved databases elect a person (or team) to coordinate the initial formation of this system. This person is responsible to identify a scheme for the system possessing all specifications that allow its correct functioning.

That scheme has to include the specification of the different federations of the system, depending on the different sets of data to be shared; the definition of the databases composing each federation; the canonical data model and language to be used in each federation; definition on how to classify the distinct components in groups and specialised terms, and so on. Based on these specifications each DBA is responsible to define how to build the extra structures of its component. Notice that all of these specifications are defined having conformity of the involved people. Following, the various masters of the system are established and the Syst-DB is constructed with general information about the whole system.

After this first phase the system starts to operate and any modification related to its "grow" or "shrink" is executed as specified in the next section. This initialisation phase can be performed for a restrict number of databases (for instance, 2 databases in a single federation[1]). Following, based on the scalability property the system can evolve easily.

[1]It does not make sense to have a system that allow interoperability among heterogeneous databases composed by a single database.

4.2 Scalability of the System

The propose architecture permits a dynamic evolution of the system. It allows a component or a group of components to join or leave a federation without having to stop the whole system and preserving the autonomy of all of the databases. In the approach, the choice of either joining or leaving the system is unilateral. That is, a component (or group) has autonomy to decide whenever it wants to participate in the system, either sharing or exchanging data, or only interrogating the other components (enquirer databases). However, whenever adding a component into a federation it is necessary to have authorisation from the other databases of this federation. Record that the criteria to form a federation depends on the data to be shared and on the other components with which they want to share this data. Therefore, the existing databases have to agree on sharing and exchanging data with that new component. In the case of an enquirer database it is also necessary to have authorisation of the components that are going to be interrogated by it. Sometimes it is possible to have a database that does not want certain databases to have access to its data.

We divide the scalability process into three cases: (a) the addition case, (b) the deletion case, and (c) the modification case. The first two cases are subdivided into four subcases: (1) addition (deletion) of a single component, (2) addition (deletion) of a group of components, (3) creation (removal) of a federation, and (4) addition (deletion) of an enquirer database. Apart from the removal of a federation and deletion of an enquirer, the execution of these subcases is performed in two different steps. The first step is related to organisational aspects of the system. That is, the specification of where to allocate new components and when to create new federations and groups. The second step refers to the inclusion (removal) of components and to the notification of the changes performed in the system, into the other databases. This notification is necessary due to the fact that the discovery process does not use any centralised structure containing information about the other databases. Thus, whenever a component joins or leave a federation (or the whole system) the other components have to be informed (updated) about the changes.

In the next two sections we present all of the possible cases. The execution of these cases are supported by the Syst-DB structure, the DBAs of the involved components and the managers of the involved federations. Thereby, they guarantee the non-interruption of the system whenever a component joins or leaves a federation. Notice that when we refer to the addition of a "new" component it is possible to have already this component in the system. However, the term "new" refers to the case where this component wants share and exchange data with a new group of components. That is, it refers to the participation of a component in a different (new) federation. In the case where a component is completely "new" for the system (does not participate in any federation until now) and it wishes to participate in more than one federation, we consider the addition into each federation as a different process. On the other hand, the deletion of a component does not mean the complete removal of this database from the system. Actually, it means the elimination of a component from one of the federations that it participates. Nevertheless, it can still participating in other federations. Hence, the complete removal of a component from the system can be performed through various delete processes (one for each federation that this database is related to).

The execution of the addition and deletion processes is assisted by the DBAs and managers of the involved components and federations, respectively. The necessity of

human help is to decide about the characteristics of how to organise the system, to allow (authorise) inclusion of databases and to maintain the system consistent. In the addition cases (see subsection 4.2.1), the Syst-DB is always consulted before inserting a new structure[2]. This is to identify and analyse the correct place to insert it. Therefore, the dynamic characteristics of the system permit modifications of it between the analysis and the inclusion of the new structure. Hence, reflecting on the decisions taken. To better illustrate, we list below the situations that can occur between the consultation and the insertion of a structure. (a) Remotion of a federation where either a single or a group of components, or an enquirer database is supposed to be inserted; (b) remotion of a component that participates in a federation where either a single or a group of components are considered to be added; (c) remotion of a component that is assumed to participate in a new federation being formed; (d) insertion of another new component into a federation where another new single or group of components were considered to be added (record that the decision where to include a structure is related to the existing components and asks for the authorisation of the related databases). All of these cases can be supported by the DBAs and managers involved.

4.2.1 Addition Case

This case is divided into the addition of a single sharing component, the addition of a complete group of components, the creation of a new federation and the addition of an enquirer database. For the two first cases, during the inclusion and notification phase, it is not necessary to inform to all the components in a federation, at the same time, about the existence of a new component (or group). We adopt the idea of first informing strategic points (the masters) and then, whenever possible, notifying the other components of the federation. The way in which the discovery process is executed permits the access of new components even when they are not notified in all the existing databases.

Single Sharing Component

Consider a component DB_i to be added into the system willing to share data with other components.

a) Organisation Phase:

In this phase, based on the set of data that DB_i is willing to share with the other components, its DBA consults the Syst-DB in order to identify either (i) a federation where DB_i can be inserted, or (ii) some existing components to form a new federation with DB_i. According to this consultation, the DBA can realize that it is better to include DB_i in more than one federation by dividing its set of sharable data. In this case, the insertion of DB_i into each federation is performed separately as a different addition case.

In case (i) suppose that the DBA identifies federation F_j as a possible federation to include DB_i, and than s/he contacts the manager of F_j. This manager is responsible to verify if the other databases in F_j agree with the participation of this new component and to specify the correct group to allocate DB_i. Therefore, the manager contacts the

[2]In this part of the text, unless otherwise specified, by *structure* we mean either a single component, or a group of components, or a federation, or an enquirer database.

DBAs of the other components in order to get an authorisation of inclusion. When all the DBAs agree with the addition of the new member (authorise it) the addition process starts. Notice that this authorisation process does not stop or lock the involved databases and preserve the autonomy of all the components, since it is performed by the DBA of each database who has enough control and knowledge about it. Nevertheless, whenever some of the existing components in F_j do not agree to form a federation with DB_i, then it is possible to compose a new federation without these databases (case (ii)). The inclusion of DB_i into F_j can generate the creation of either a new group in F_j or the specification of specialised terms that better classify DB_i in an existing group of F_j. The creation of a new group is explained in subsubsection 4.2.1. The specification of new terms is executed by the manager of F_j, since s/he has good knowledge about the federation. Thus, after identifying the correct group to introduce DB_i and the necessary terms, the manager informs the DBA of DB_i about these facts. Following, the inclusion and notification of DB_i is executed as explained in item (b) below.

In circumstance (ii), after consulting Syst-DB, the DBA realizes that there is no suitable federation to allocate DB_i. S/He specifies possible databases appropriate to form a new federation with DB_i. This case is related to the creation of a new federation outline in subsubsection 4.2.1.

b) Inclusion and Notification Phase:

After performing the organisation phase suppose that the DBA of DB_i identifies federation F_j, and the manager of F_j notifies group G_l in F_j as the correct position to insert DB_i. Thus, after identifying how to classify DB_i in G_l (specification of terms) the additional structures ($LPS_i^j, SPD_i^j, MLPS_i^j, LOC_i^j$) are created and added to DB_i (this is done individually and separated). The LOC_i^j can be built based on the master of G_l named M_l. Notice that M_l contains the more up to date information related to G_l, as clarified next.

Following, M_l is notified about the existence of a new component DB_i willing to join the federation. This notification is made by updating the tree of M_l with the information related to DB_i. After introducing DB_i to M_l we guarantee that DB_i can be achieved by the discovery process of any query q' related to G_l. This is really possible since, during the discovery process, the master of a group is visited, whenever it is necessary (see chapter 5).

To update the other components inside F_j we propose M_l to send the new information to the other masters in F_j. At this point, each master is responsible to update the LOC structures of the databases inside their respective groups. This task can be done in two different ways.

One way consists in broadcasting the new information by the master of each group to all the components of its group. Nevertheless, depending on the number of components involved and on the frequency of new databases joining the federation, this method can generate a great amount of traffic in the network. The second mode consists in updating the components during the discovery process, assisted by the discovery graph (see chapter 5), as a "reply" of a visit in the LOC of a component containing more up to date information (specially when this component is a master). Therefore, every time that a component DB_i sends a query to another component DB_k having more up to date information in

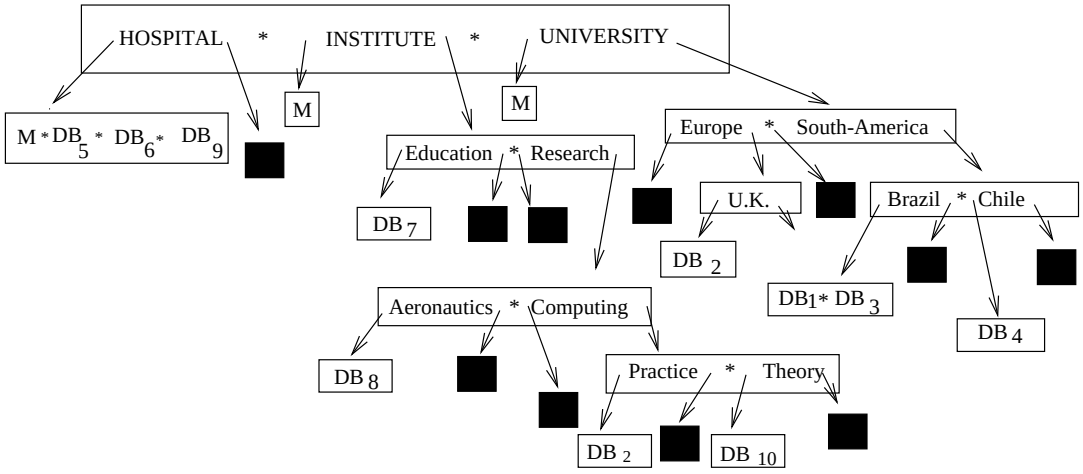

Figure 4.1: Example of LOC with the new component DB_{10}

LOC_k^j than in LOC_i^j of DB_i, the additional information of DB_k is introduced into DB_i. This can be done by comparing the two structures LOC_i^j and LOC_k^j.

Notice that the proposed update approach permits the LOC structures of a federation to contain different information at the same time, i.e., all the LOCs do not handle the same information. However, the way in which the discovery process is executed guarantees the access of any existing (public) data registered in the components of a federation. In the worst case the process achieves a master that contains up to date information.

We propose the use of another strategy together with the "reply" approach to increase the performance of the updating process. This strategy consists in updating the databases located "near", in a classification point of view, to the new component DB_i, after notifying the master M_l. Hence, the first components to be updated are the components classified with the term near to the term that DB_i is related to, succeeded by the components related to the above term in the hierarchy, and so on, until achieving the root node. The information of which are the components near DB_i can be discovered by accessing the master.

To illustrate the "near" strategy consider the hierarchy structure of figure 3.4. Suppose that a new component DB_{10} is added to the federation. Consider DB_{10} as a database related to a *research institute in theory computing*. Assume that DB_2 is related to *practical aspects of Computer Science*. The up to date LOC structure containing DB_{10} is presented in figure 4.1. The "near" strategy consists in updating DB_2, followed by DB_8 and DB_7, respectively. With this strategy we guarantee that DB_{10} can be accessed in less steps, even when a query is performed in a source outside the group of DB_{10}, not having information about DB_{10}. The use of this strategy, together with the "reply" method, permits a better performance in the process of updating the other components.

Consider the case where a query q' is performed in DB_1 and it does not know about DB_{10} (DB_1 has the LOC structure presented in figure 3.4, for instance). Suppose that q' is related to a *research institute in theory computing*. When the discovery process traverses the LOC of DB_1 it identifies DB_2 as a possible target component and sends q' to it. However, since DB_2 contains data about *practical aspects in Computer Science* it does not hold the answer for q'. Thus, a search is executed into the LOC of DB_2 and DB_{10} is identified (we are assuming that DB_2 contains information about DB_{10} by the use of

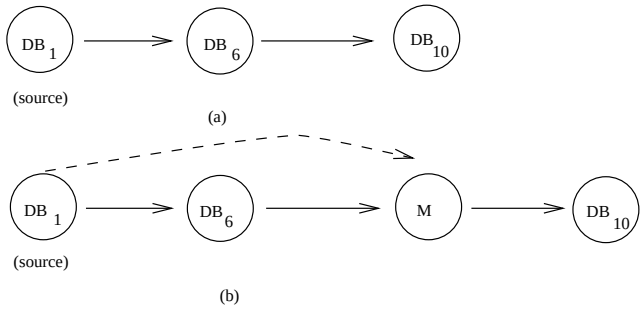

Figure 4.2: Discovery graph to access DB_{10}

the "near" strategy). Thereby, the correct component is accessed in less steps comparing to the case when the master is accessed. Figure 4.2 presents the graph for the discovery process of q' with (a) and without (b) using this "near" approach.

Notice that the "near" strategy cannot be used to update a federation instead of using the master. It is impossible to know which is the more up to date LOC inside the federation that can assist in the construction of the LOC structure of a "new" component. Another drawback is related to the identification of all the "near" components. The LOC structure that may be chosen to support this task can contain no information about other components recently introduced.

Despite the notification of the "new" component in other components of the federation it is necessary to update the Syst-DB. This can be performed, by the manager of the federation, at the same time (in parallel) that the first master (M_l) is being updated.

Group of Sharing Components

The creation of a new group of components G_k can be performed because of two different reasons: (i) as a reflection of the addition of a single component DB_i, or (ii) as the inclusion of a collection of related databases willing to share and exchange information.

Case (i) occurs when the insertion of a new single component (DB_i), in a federation (F_j), forces the creation of a new group (G_k), to better classify DB_i into F_j. Situation (ii) can be subdivided into two cases: (1) the new group to be added generates the creation of a new federation containing this group, and (2) the new group is added (as a "block") into an existing federation. Notice that here we are not mentioning the case where a collection of components is introduced into a federation and these components are related to the different groups of the federation. Actually, this case is treated as the addition of a single sharing component executed various time (one addition for each component in the collection).

a) Organisation Phase:

This phase is related to the inclusion of a collection of databases (case (ii) above)[3]. The first step consists in selecting one of the DBAs of the involved components to consult the Syst-DB. This DBA consults the Syst-DB and identifies if either the new group can

[3]The organisation phase for case (i) has been already executed when trying to include a single sharing component.

be inserted in an existing federation, or a new federation has to be created for this group. Notice that this decision can be executed together by all the involved DBAs based on the information of Syst-DB.

In the first case, suppose that federation F_j is chosen to receive the new group. The manager of F_j is contacted in order to verify if the other members of F_j agree with the inclusion of this new group, to specify how to allocate (classify) the new group into F_j, and to define new terms whenever necessary. Sometimes, some of the components of this group can participate in (be allocated to) other groups existing in F_j[4]. The authorisation process is executed in the same way as performed in the addition of a single component, that is, consulting the DBAs of the other members. After having the authorisation and defining the new specifications for F_j, the manager notifies them to the DBAs of the new components. Following, the inclusion and notification of the new group is performed as present below in item (b).

The case where a new federation has to be created containing the databases of the group is specified in the next subsubsection.

b) Inclusion and Notification Phase:

Consider the situation of adding a complete new group G_k into federation F_j. This requires the creation of a new term in the root node of the LOC structures, together with a new branch related to this term.

After preparing all the components of the new group G_k with its additional structures it is necessary to update all the masters of the other existing groups inside F_j. That is, we assume that G_k is introduced into F_j (reachable by the other components) when all the masters have the information about it. Following, each master is responsible to update the components of its group. This can be done in the same way as proposed for the case of introducing a single component, i.e., either broadcasting or as a "reply" during the information discovery process. In the current case the "near" approach cannot be applied, since there are no "near" components in the tree (a new term is inserted into the root node).

We affirm that when all the masters of F_j are notified about the existence of a new group, this information is reachable by the other components. These components consult the related master whenever necessary.

Associated with the notification of the masters about the new group it is also necessary to update the Syst-DB. This is performed by the manager of the federation.

Creation of a Federation

The creation of a new federation in the system can occur as a result of: (i) the addition of a single component DB_i, or (ii) the addition of a group of components G_k. In case (i), the DBAs of the databases identified as candidates to form a new federation with DB_i are consulted to determine if they have interest to participate in this federation[5].

[4]Record that the architecture permits a component to be classified in different groups, inside the same federation, sharing the same set of data.

[5]Notice that the creation of a new federation has the similar steps executed in the initialisation of the system.

a) Organisation Phase:

After specifying the databases to participate in a new federation F_y a manager is chosen to coordinate this federation. The DBAs of those components are responsible to determine the set of data to be shared by each database in the new federation. Based on this data and on the type of components, the manager specifies the federation. The specification includes definitions: of different groups and associated terms, of common data model and language to be used in the federation, of semantic relationships between the different data, and so on. That is, a complete design (scheme) is defined for the new federation F_y.

b) Inclusion and Notification Phase:

Based on the scheme defined for the new federation F_y, the DBAs of the involved components have to define their additional structures ($LPS, MLPS, SPD$ and LOC). Following, the masters are identified and constructed. After preparing all the structures, the manager of F_y updates the Syst-DB about the existence of a new federation. Thus, the components of F_y can interoperate among each other.

Addition of an Enquirer

This is a special case where a database wants to participate in the system as an enquirer, i.e., only interrogating other components without sharing any data. Notice that a database can be an enquirer in one federation and can participate in other federations by sharing data. A database is an enquirer depending on the federation that it participates. Due to the fact that an enquirer does not share any data, it does not have to be known by the other participants. Nevertheless, the enquirer has to know about the other components to interrogate them and to have authorisation to participate in a federation. We are not specifying the addition of a group of enquirers. This can be seen as the addition of various single enquirers.

a) Organisation Phase:

Consider an enquirer database DBE_i willing to participate in the system. In this phase, the DBA of DBE_i consults the Syst-DB to specify which are the federations that best suit its characteristics. For each different federation that DBE_i decides to participate, it maintains the addresses of the masters of these federations, together with a copy of one of these masters, in its LOC structure[6]. Suppose that DBE_i wants to participate as an enquirer in federation F_j. The DBA of DBE_i asks the manager of F_j permission to participate. This manager is responsible to contact the DBAs of the other components to get this authorisation.

b) Inclusion and Notification Phase:

[6]Record that for an enquirer database it is not necessary to maintain the $LPS, MLPS$ and SPD structures.

When the DBAs of all the existing components of F_j authorise the participation of DBE_i, it is inserted as an enquirer. Thus, every time that DBE_i needs to interrogate F_j it consults its LOC_i^j and executes the normal information discovery process. Notice that the other components in the federation does not know about the existence of the enquirer (they only authorise it).

The DBE_i is responsible to update its different LOC structures. That is, periodically, it consults the masters of the federations that it participates and update its different LOCs. Even when the DBE_i has a not up to date LOC structure, the information discovery process permits the location of the correct data. However, it is necessary to update the different LOCs of DBE_i because of the insertion of new groups of components into the system. They can be used as new sources to be consulted. Whenever an enquirer database joins a federation in the system this fact is registered into the Syst-DB.

4.2.2 Deletion Case

Unlike the addition case, the task of updating the components of a federation about the removal of either a single component or a group of components is complex and laborious. The information about the non-existence of a certain component (or group) has to be known by all the components. Otherwise, during the discovery process of a query q', it is possible to identify a possible target component not present in the federation anymore. We affirm that the removal process does not occur as often as the addition process. The number of components that leave a federation is less than the number of components that join a federation. Differently, after a certain period of time, it will be possible to achieve an empty federation.

Nevertheless, we permit the LOC structures of the components to hold incorrect information for a certain period of time. This is allowed to avoid a great amount of messages in the network by broadcasting to all the LOCs about the removal of a database. The idea consists in updating the system gradually, not all the components at the same time. However, this can lead to the situation of trying to access a database that does no more exist in the system. We propose the insertion of a "special" message into the network address related to the component removed of a federation (or the system), specifying that it does not last. Therefore, whenever a transaction (query) tries to access that database it achieves the message and identifies the non-existence of the component. We present below an approach to deal with the problem of removing a single sharing component, a complete group of components, a federation and an enquirer database. For the two first cases we present the organisation, and the removal and notification phases. The removal of a federation and of an enquirer database does not have to be notified in the other components of the system.

Single Sharing Component

Consider a database DB_i in a federation F_j that does not want to share and exchange data with the other components of F_j anymore. The removal of DB_i from F_j can only be performed when DB_i is not executing remote transactions, that is, transactions from other components of F_j.

a) Organisation Phase:

In this phase the DBA of DB_i decides that DB_i has to cease sharing data with the other components of F_j. Thus, DB_i tries to finish all of the remote transactions that still in execution and stops accepting new remote transactions. Its DBA contacts the manager of F_j to inform that it wants to leave the federation. The manager identifies the group(s) that DB_i is classified, updates the Syst-DB about the elimination of DB_i from F_j, and inserts the "special" message into the position (network address) of DB_i, reporting that it is not available to F_j anymore.

We propose the idea of first updating Syst-DB in order to avoid "new" components of forming either a group or a new federation, with the set of data that DB_i is sharing with the components of F_j. The use of a special message into the address of DB_i (related to F_j) is to execute a "reply" to any component that tries to access DB_i and does not know about the the non-existence of it yet.

The deletion of a component can generate the elimination of a complete group when this group is formed by a single component (the one being deleted). The removal of a complete group is explained in the next subsubsection. On the other hand, it is impossible to have the removal of a federation caused by the deletion of a single component. It does not make sense to maintain a federation formed by only one component.

b) Removal and Notification Phase:

After updating Syst-DB and inserting the special message into the address of DB_i, the masters of the groups that contain DB_i have to be notified. The notification is executed by removing the identification of DB_i from the tree. Consider G_l a group in F_j, with master M_l, containing DB_i. Using the same idea of the addition case, M_l sends messages to the other masters in F_j about the elimination of DB_i. Each of these masters is responsible to update the components in their group.

Apart from the broadcasting and "reply" approaches used in the addition case, a component can be notified about the non-existence of another database, whenever it detects the special message in the address of this component.

The information about the deletion of a component is not determined for every component at the same time. However, sooner or later every database is notified about this fact. We affirm that it is better to try to access a component that does not exist, than to stop the whole system to notify about the removal of a component (record that the number of deletions is small). For the system, a component does not exist in a federation since the moment that the "special" message is introduced into its position (address).

Group of Sharing Components

The removal of an existing group G_k from a federation F_j can be performed by two different reasons: (i) as a reflection of the deletion of a single component DB_i, or (ii) as the removal of a collection of related databases that do not want to share and exchange information anymore. This latter case is rare, but possible.

Case (ii) can force the removal of a complete federation from the system, when this federation contains only the group being removed. This situation is defined in next subsubsection.

a) Organisation Phase:

This phase is related to the removal of a complete group G_k (case (ii) above)[7]. The DBA of one of the components of G_k informs the manager of F_j that G_k wants to leave the federation. The manager has to identify if there are other groups in F_j containing any of the components of G_k. Following, the manager updates the Syst-DB about the removal of these components. When all the involved components finish the execution of all the remote transactions, the manager inserts the "special" message into the addresses of all these components. Then, the other databases in F_j (not in G_k) have to be notified about the non-existence of G_k. This is performed as presented below.

b) Removal and Notification Phase:

Consider the case of removing a complete group G_k from federation F_j. This requires the removal of the term related to G_k from the root node of the LOC and master structures in F_j. When any of the components of G_k is classified in other groups of F_j, different from G_k, the related branches of the tree have to be updated too. Following, all the masters in F_j are updated and responsible to report the fact for their own components.

When the number of components in G_k is big (more than three, for instance), we propose to execute the updating of the components by broadcasting, in order to accelerate the updating process. The reasons to adopt the broadcast strategy when the number of removed components is big are: (a) a low frequency of removing a complete group caused by case (ii); (b) a high probability of trying to access a component of the removed group. Nevertheless, when the number of components in the deleted group is small we suggest to perform the update in the remaining databases using the "reply" and "near" strategies.

Removal of a Federation

Consider the removal of a federation F_y. The elimination of F_y from the system can occur as a result of: (i) the deletion of a group of components, when the federation is formed only by this group, or (ii) the agreement of a collection of databases that do not want to share and exchange data anymore.

Case (i) is identified by the manager of F_y. However, in case (ii) the manager has to be notified about the intention of all the components to finish with the federation. The manager updates the Syst-DB about the non-existence of F_y in the system. At this point, no other component (or group of components) tries (try) to be added into F_y. It is also necessary to inform the enquirer databases of F_y about its non-existence. Thus, the manager of F_y consults the Syst-DB and identifies the enquirers of F_y. Following, s/he sends a message to these enquirers announcing the removal of F_y. The DBAs of the enquirers are responsible to update the database about this fact. This is done by deleting the LOC structure of these enquirers related to F_y.

When all the remote transactions in execution in F_y finish, the masters are deleted. Each component is responsible to delete its additional structures related to F_y (LPS,

[7]The organisation phase for case (i) has been already executed when trying to remove a single sharing component.

$MLPS$, LOC and SPD).

Deletion of an Enquirer

The elimination of an enquirer database DBE_i from a federation F_j is very simple. Due to the fact that DBE_i participates in F_j in a unilateral way, it is not necessary to notify the other components of F_j about the non-existence of DBE_i.

Whenever DBE_i wants to stop enquiring the components of F_j its DBA notifies the manager of F_j. The manager is responsible to update the Syst-DB. Following the LOC_i^j of DBE_i is removed and it does not participate in the federation.

4.2.3 The Modification Case

This case consists in the situation where a component participating in one or more federations wants to modify some of its shared data. Without loss of generality, suppose a database DB_i that participates (sharing and exchanging information) in federations F_j, F_y and F_t, and has a different set of shared data (possible overlapped) with each of these federations (LPS_i^j, LPS_i^y and LPS_i^t).

Assume that LPS_i^j has to be modified. This modification can be related to the addition of a new type of data and/or the removal of a certain existing type of data. Any change in one of the LPSs related to DB_i forces a review about the maintenance of DB_i in the federation that contains the modified LPSs. The participation of a database in a federation depends on the data that this component is sharing and on the other components. Thus, it is necessary to have a "multilateral" agreement.

The DBA of DB_i contacts the manager of F_j and notifies the necessity of the changes into LPS_i^j. Following, this manager contacts the other DBAs related to the components of F_j to inform about the changes. The manager, together with the DBAs, verify if these changes can violate the specifications related to the formation of F_j.

In the situation where those changes do not interfere with the specifications of F_j, the LPS_i^j is updated. Notice that this is executed without interfering with the other components and preserving the autonomy of the databases. The decision of performing changes is done by the proper component. It only needs permission of the other components. However, in the case where the changes do not guarantee the maintenance of the specification of F_j, we propose the removal of DB_i from F_j. Following, DB_i is added, with the "modified set of sharing data", into a proper federation.

4.2.4 Consulting and Updating Syst-DB

Whenever the addition and deletion processes are performed the Syst-DB has to be consulted and/or updated. This is performed to guarantee confidentiality and correct information of the system. In all of the addition cases, the Syst-DB is always consulted before inserting a new structure into the system. Thus, for the insertion of a structure, the number of consults in the Syst-DB is equal to the number of updates performed into it. On the other hand, in the deletion cases, the Syst-DB is consulted only when removing a federation of the system, to identify the possible enquirer databases in this federation, but it is updated in all the cases. Therefore, we conclude that when a structure is inserted

or removed from the system the number of consultations is less or equal to the number of updates in the Syst-DB.

We propose the use of a centralised structure for the Syst-DB. The idea of having different copies of the Syst-DB (replication) in the network facilitates the consultation task, but makes difficult the update of the Syst-DB. In our case this idea is not effective since the number of consults is less or equal to the number of updates. On the other hand, as proposed in [35], the idea of creating images (copies) of the Syst-DB for each consultation is very expensive. It can also generate the problem of having images that are not up to date.

The Syst-DB reflects the situation of the system. Therefore, it is possible to have changes of the Syst-DB between a consultation and an update related to it. However, those changes do not cause problem in the addition and deletion processes because of the help of the DBAs and managers.

4.2.5 Master vs. Syst-DB

In the propose approach we suggest the use of structures like the master and the Syst-DB to assist in the discovery process and to permit scalability of the system. Notice that the principal role of the master is to support with the discovery process. In particular, to facilitate updating the LOC structures of a federation when databases join or leave a federation. We affirm that the use of the master does not cause a centralisation of the discovery process. During this process, the master is visited in the same way as the other components (LOCs), in order to identify databases that can have the desired data. This is different than consulting the same structure to discover the correct position of a certain data, for every query.

In the updating process the master is used as a starting point to this procedure. The other components "know" which are the structures that possible have more up to date information (masters in this case), consulting them whenever necessary. However, throughout the insertion of a new group into a federation it is possible to have a certain centralisation in the various existing masters. Record that when a group is not identified in the root node of a source component its master is consulted to verify if a new group has been inserted. We guarantee that this does not occur very often, and that the number of consultations in each master (related to this new group) is not so big.

The Syst-DB is used to permit the evolution of the system in a correct way. It assists in the insertion (deletion) of new components, group of components and federations into (from) the system, related to the organisation and classification points of view. The necessity of having a structure like the Syst-DB maintaining a good organisation of the system is because the choice to include or remove a database does not depend on the other components. The inclusion of a component in a wrong position can cause drawbacks to the system, such as: violation of the security and confidentiality of the data and difficulties in locating the correct data.

Chapter 5

The Discovery Process

One important issue when dealing with interoperability of heterogeneous databases is the discover of relevant data and where it is located. Most of the existing approaches in the literature assume that the relevant data is either already known or identified and try to process it efficiently. Thus, based on the lack of approaches and on the importance of information discovery we present a new distributed concept suitable for this task.

5.1 Assumptions

Before presenting the information discovery process we outline some assumptions necessary to better perform the process.

1. There is a way in which a certain query q' can be broken into *atomic subqueries* in order to determine the different data necessary to fulfill this query. The term atomic subquery is used to identify that a subquery is undivided.

2. There is a form of knowing if the component where q' is performed (*source component*) contains part of the information necessary to answer the query and which is this part.

3. Each component has a way to identify which of the federations that it participates has to be searched in order to find the correct data. This last assumption can be performed by maintaining, for each component, an extra structure with the information about the different federations that it participates.

The basic concepts of the process are to try to reduce the universe of search and to permit starting points for the search operation. That is, decrease the number of components that are visited during the discovery of the correct data and identify components that may have the requested data. Other goals are related to the avoidance of centralised structures to assist on the discovery process, and transparency of the process.

5.2 General Description

After identifying the query q' to be searched outside the source component[1] and the correct federation, the discovery process for q' is started. Notice that for each different q' its discovery process is performed inside a single federation. We do not allow a query to pass through federations in order to preserve the confidentiality of the system.

In the first step of the discovery process of a query q' the source component consults its LOC structure related to the proper federation. Based on the kind of data to be searched the LOC hierarchy is traversed. A correct branch is chosen at each step, until a left side leaf node is reached (i.e., the right pointer related to a certain term visited is undefined (nil)). This leaf node can contain three different types of sets of information: (a) the address of the master together with the address of one or more databases inside a specific group (in the case where there is no specialisation of the general name of the group)[2]; (b) the address of a specific component; (c) the addresses of more than one component. In any of the above cases q' is sent to these respective components, named *possible target components*, together with the addresses of the source component and the master of the group to be searched. When q' is sent for more than one component the search is performed in parallel. The address of the source component is sent to permit this component to receive the answer of q'. The address of the master is also important since the propose architecture allows a component to participate in different groups. Thus, it is necessary to determine which is the group related to the current search.

When a possible target component receives q' and it does not contain the requested data, then it consults its own hierarchy structure (LOC) to find other possible target components to be searched. The process is recursive. If a new possible target component is detected, q' is sent to it. Notice that this is done, since the LOC structures of the components in a certain group can have more up to date information related to the group that it belongs to than the LOC of the source component (see chapter 4). When the LOC of a component is searched and all of the identified components have been visited, then the master of its group is searched. Therefore, whenever this component participates in more than one group, the address of the master of the current group, passed together with the query, is used to clarify doubts. Notice that we do not want to allow a master of a different group to be searched since this can generate unnecessary searches. Record that the master of a group contains the most up to date information of this group. Thus, the master of another group can contain either less or the same type of information than the master of the group being searched. We do not propose the discovery process to start at the master of the group identified in the LOC structure of the source component. This can generate a centralisation of the process. On the other hand, a possible variation of traversing the hierarchical structures of the source and master components at the same time, can cause the identification of unnecessary duplication of information.

Sometimes it is not possible to identify a term related to a query q' in the root node of the LOC of the source component for q'. In this case the procedure visits the master of the

[1] Unless otherwise specified, from now on, the term *query* is referred either to a single atomic query, when this query cannot be broken into subqueries, or to one of the atomic subqueries of a certain non-atomic query.

[2] In the case where a group name is not specialised the achieved leaf contains the addresses of the master and of the databases forming this group. We guarantee that the number of these components is really small, otherwise the group will be able to be divided in subgroups.

group of this source, in order to try to find a proper group related to q'. This situation can happen whenever a new group is created into the federation and that source component has not been notified yet. However, in the way which the updating procedure is executed, all the masters have the information about a new group before the other components are updated (see chapter 4). Notice that in this case there is a certain "centralisation" of the process in the master. Nevertheless, we guarantee that the creation of a new group (new term in the root node) is not performed very often. Another point is related to the fact that not all the components visit the same master at the same time. Only the components of a group visit its master when they do not have knowledge about the new information.

We propose the idea of building a "virtual decentralised" direct acyclic graph, named *discovery graph*, for each query q' performed in a source component, to assist on the discovery process

The Discovery Graph

The discovery graph $DG_{q'}(V(DG_{q'}), E(DG_{q'}))$ consists of a nonempty set of vertices $V(DG_{q'})$ and set of edges $E(DG_{q'})$. $V(DG_{q'}) = \{db_1, db_2, \cdots, db_i, db_j, \cdots, db_n\}$, $1 \leq i, j \leq n$, $i \neq j$, $\parallel V(DG_{q'}) \parallel = n$, is composed by the source component of a query q', by the component databases of a group inside a federation visited during the discovery process of q', and by the master of this group (when it is visited during the discovery process of q')[3]. The set of edges $E(DG_{q'})$ contains direct edges of the form $e_{ij} = db_i db_j$, with db_i and db_j different from the source and master components respectively, denoting that db_j[4] was identified after traversing the LOC structure of db_i and discovering that db_j has not been visited yet for the search related to q'. It is also possible to have an edge $e_{ij} = db_i db_j$ where db_i is the source component and db_j the master, whenever the master is visited during the discovery process of q'.

Notice that the graph is built in parallel since the discovery process is also executed in parallel. For each query q', whenever the master of the group of the components being searched is visited, it is represented in $DG_{q'}$ and it is managed in the same way as the other components. In this case, we propose the notification about the visit of the master into the source component. Figure 5.1 represents this fact where the dash arrow denotes the knowledge of the source about the existence of the master in the discovery graph. We affirm that when the master does not appear in the discovery graph for a query q', or appears in a leaf of the graph, this means that the graph is not very deep. The other visited components contain up to date information.

The term "virtual" stands for the fact that $DG_{q'}$ is not completely stored in a certain component and that it exists only during the discovery process of q'. Each component holds the information of the part of the graph related to it. That is, suppose the situation of figure 5.2 where db_j, db_k, db_l, $(1 \leq j, k, l \leq n, j \neq k \neq l)$ are identified to be visited after traversing the LOC of db_i. Hence, after visiting db_j, db_k and db_l, db_i stores the information that there are three direct edges from db_i to db_j, db_k and db_l, respectively. db_j, db_k and db_l are called *sons* of db_i and db_i is called its *parent*.

To prevent cycles, that is, the investigation of a component more than one time for the same query q', we propose each component to handle the information that it has done

[3]Notice that in the graph the master is represented in the same way as a component database.

[4]To simplify the notation we are not specifying the federation related to db_i and db_j.

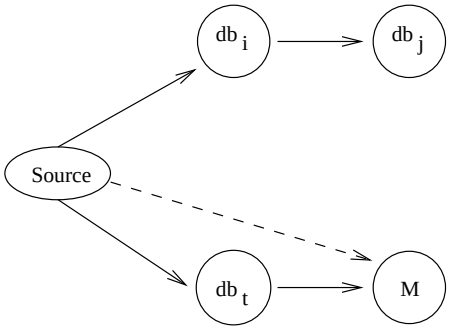

Figure 5.1: A portion of a discovery graph with the master

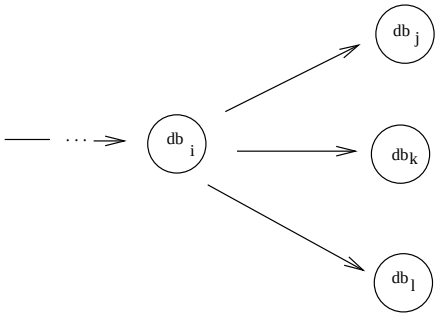

Figure 5.2: A portion of a discovery graph

a search for q'. Thus, before visiting a possible "new" component, this component is consulted to detect if it has been searched for q'. Due to the fact that $DG_{q'}$ is not totally stored in a single place and that $DG_{q'}$ is built in parallel, it is not possible to consult the whole graph (the set $V(DG_{q'})$, for instance) to determine if a certain database has been already consulted. The idea of sending a list of visited components together with q' is also not useful here. It is impossible to have a complete list of these components, at each phase, since the process is executed in parallel.

A discovery process for a query q' can finish because of three reasons: (a) a correct data to fulfill q' is found (success of the process); (b) all the components in a group are visited and none of them contains the correct data (failure of the process); (c) the user/application asks to suspend the process (stop). Since the discovery process is executed in parallel in cases (a) and (c) it is necessary to cancel the discovery process of q'. This is done by suspending all the transactions related to q' that still in execution. On the other hand, if case (b) occurs, then the source component of q' has to be notified.

For cases (a) and (c) we suggest the idea of cancelling the discovery process by sending *interruption messages* (Int-Msg for short). We are not adopting the "time-out" or "number of components visited" approaches as a way of deciding about case (b). In an environment containing a great number of different world-wide databases, either a slow transaction or a large quantity of databases visited can generate the wrong deduction of the non-existence of a data. The source component is responsible to determine about the cessation of a discovery process. Thus, the Int-Msg is sent recursively to all components visited during the process and in the order that they where searched. The graph $DG_{q'}$ is traversed again in the same way that it was built (in parallel). Notice that each compo-

nent visited during the discovery process knows the other components visited after it and identified in it. Hence, it is not necessary to traverse again the LOC hierarchy of each visited component.

If $DG_{q'}$ contains the master component of the current group we propose to send the Int-Msg also from the source to the master. Then the master can send the Int-Msg to its sons (in parallel). This is to achieve a better performance on the interruption process. Record that the source component knows about the existence of the master in the discovery graph (figure 5.2).

One can think that in the propose approach it could be possible to have the situation where an Int-Msg arrives in a component always after the component has dispatched the query (a livelock problem). Nevertheless, we guarantee that the discovery process always stops after a certain point, because the number of components in a group is finite and a component is never visited twice. Therefore, the Int-Msg always achieves all the components visited in the execution of the process. Another problem is related to the fact that an Int-Msg arrives in a certain component before the request query q' has arrived in it. Actually, this situation cannot happen since an edge $e_{ij} = db_i db_j$ (with db_i and db_j different from the source and master) is added to the graph after visiting db_j, i.e., after db_j received the request for q'. Even in the cases where db_i sends an Int-Msg for its other (possible) sons, before receiving a reply from db_j, db_i sends the Int-Msg for db_j immediately after receiving the reply from db_j.

Another approach consists in sending messages from a possible target component to a source component of a query q', asking to continue the process. This can be performed every time before the possible target component sends q' for a "possible" son. We decide not to adopt this approach, since it generates a great amount of traffic in the network. The variation where a possible target component asks its father authorisation to continue, which asks its own father, and so on, until reaching the source component is not useful here. This also generates a lot of messages going and coming, even when the graph is more bushy than deep.

The idea of mixing both approaches together, that is, sending Int-Msgs and allowing a component to asks its father if it may continue, is not so advantageous here. Suppose the situation of figure 5.3, where db_j asks db_i if it may continue (represented by "May I?") before the Int-Msg sent by db_h arrives in db_i. db_i tells db_j to continue (represented by "Yes") and after receiving the Int-Msg from db_h sends it to db_j. However, it is possible that db_j has already sent the request of a query to one of its possible sons. Since every involved component receives the Int-Msg (the group has a finite number of components), we affirm that the message related to the authorisation to continue is unnecessary. It only generates more traffic in the network. The only advantage of this mixed approach is to accelerate the interruption process in some cases. However, with the idea of sending the Int-Msgs starting from the source and master components increases the performance of the process.

When all the possible components for a query q' are visited and none of them contains the correct data, the source component has to be informed. This is done to avoid making the source component to wait. A component stops its search for a query q' when it does not contain both the answer for q' and knowledge about another possible component. It is a node represented in the graph without an outcoming edge. Whenever this happens this component notifies its father about the non-existence of the information. When its father receives this type of message from all of its sons it sends the same message to its father,

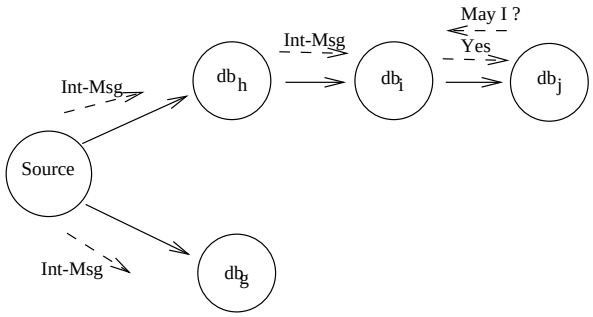

Figure 5.3: Two approaches in execution at the same time

and so on, until the source component is achieved. Hence, the failure of the discovery process is concluded when the source component receives the non-existence message from all of its sons. We allow the Int-Msg to have greater priority than the message of non-existence of an information, that is, the latter message is discarded in the presence of the Int-Msg.

We believe that the way in which a data is searched avoids the examination of all of the components of a group at the same time and it also reduces the universe of search to at most a certain group of databases into the federation. The process does neither broadcast for every component nor use a centralised structure. It is also executed without any interference of the user guaranteeing transparency in the process. Even using a structure like the master, the proposed process cannot be consider centralised. The master is only consulted in special situations and not by every component at the same time. Nevertheless, it is necessary to resolve the problem of having the master down. Another drawback is concerned to the fact that the correctness of the algorithm is directly related to the way that the groups are formed in a federation and to the choice of the specialised terms. Therefore, we have to specify criteria of how to define and organise the groups.

5.3 The Algorithms

The discovery process is described below using algorithms that are presented under the form of decision tables [116]. These tables are divided into two subtables (conditions and actions) where "y", "n" and "-" stand for "yes", "no" and "indifferent", respectively. Numbers in a certain column of an action section indicate the order in which the actions, corresponding to conditions which are all true for that column, are to be performed. Conditions and actions are expressed as "macros" defined after the tables. Some algorithms are described using macros defined in previous algorithms.

5.3.1 Algorithm to identify a group

Comment: This algorithm identifies the correct group of components inside a federation where a query q' can be possible solved, if this group exists.
Input: The query q', the LOC structure of the source component related to the correct federation and the master of the group containing this source component.

Table A:

related-term-root?	y	n
traverse-LOC	1	
traverse-M		1
go-to-table-B		2
return-poss-tgt-comp	2	
return-M	3	

Conditions:
related-term-root?: is the term related to q' identified in the root node of the LOC of the source component?
Actions:
traverse-LOC: traverse the LOC structure until achieving a proper left leaf node.
traverse-M: traverse the master of the group of the source component.
go-to-table-B: execute table B.
return-poss-tgt-comp: return the addresses of the possible target components inside the proper left leaf node achieved.
return-M: return the address of the master of the group containing the possible target components to be visited.

Table B:

related-term-root-M?	y	n
return-poss-tgt-comp	1	
return-M	2	
message-1		1

Conditions:
related-term-root-M?: is the term related to q' identified in the root node of the master of the group containing the source component?
Actions:
message-1: return the message: "the system does not contain any component where the query q' can be solved".

5.3.2 Algorithm to visit a component

Comment: This algorithm is performed for each possible target component. It identifies if this possible target component either contains the answer of q' or knows about other possible target components.

Input: The query q', the address of a possible target component DB_i returned by the algorithm that identifies a group, the address M of the master of the group containing DB_i, the address of DB_j parent of DB_i, and the address of the source component of q'.

DB_i-searched-q'?	y	y	n	n
M-searched-q'?	y	n	-	-
DB_i-contains-q'?	-	-	n	y
DB_j-does-nothing	1			
notification-DB_j				
register-M-DB_j		1		
register-M-source		2		
register-DB_i			1	1
traverse-M-group		3		
traverse-LOC-DB_i			2	
return-poss-tgt-comp		4	3	
send-asw-q'-source				2

Conditions:
DB_i-searched-q'?: does the possible target component DB_i have already been searched for q'?
M-searched-q'?: does the master of the current group has already been searched for q'?
DB_i-contains-q'?: does DB_i contains the answer for q'?
Actions:
DB_j-does-nothing: DB_j does not do anything related to DB_i.
notification-DB_j: DB_j receives a message from DB_i about the non-existence of a correct information to q'.
register-M-DB_j: the master M is registered in DB_j as a son of DB_j, forming the part of the discovery graph related to DB_j.
register-M-source: the source of q' is notified about the existence of the master M in the discovery graph of q'.
register-DB_i: the component DB_i is registered in DB_j as a son of DB_j, forming the part of the discovery graph related to DB_j;
traverse-M-group: traverse the master M of the group being visited;
traverse-LOC-DB_i: traverse the LOC structure of DB_i until achieving a proper left leaf node;
send-asw-q'-source: send a possible answer for q' to the source component;

5.3.3 Interruption algorithm

Comment: This algorithm is recursive. It sends Int-Msg to interrupt the discovery process. For each vertices (component) in the discovery graph the algorithm uses information of the sons of these components.
Input: A component visited during the discovery process of q' that is a vertices of the discovery graph, and the addresses of the sons of this component in $DG_{q'}$. This component can be either the source, or a related master, or a common component.

Table A:

found-asw-q'?	y	y	-	-	-	-	-
all-sons-source?	-	-	y	y	-	-	-
suspended?	-	-	-	-	y	y	-
$DG_{q'}$-contains-M?	y	n	y	n	y	n	-
send-Int-Msg-S-M	1		1		1		
send-Int-Msg-S		1		1		1	
send-Int-Msg							1
go-to-table-B	2	2	2	2	2	2	2

Conditions:
found-asw-q'?: does the source component related to q' is satisfied with the information (answer) found for it?
all-sons-source?: does the source component of q' received a notification of non-existence of the correct information from all of its sons?
suspended?: does the source component of q' received a notification to stop the discovery process?
$DG_{q'}$-contains-M?: does the discovery graph for q' contain the master of the visited group, that is, does the source component knows about the existence of the master?
Actions:
send-Int-Msg-S-M: the Int-Msg is sent in parallel to the sons of the source component and to the master visited component during the discovery process of q'.
send-Int-Msg-S: the Int-Msg is sent in parallel to all of the sons of the source component visited component during the discovery process of q'.
send-Int-Msg: the Int-Msg is sent in parallel to all of the sons in $DG_{q'}$ of an involved component (visited) that received the Int-Msg.

Table B:

received-Int-Msg?	y	y
component-has-sons?	y	n
interrupt-process		1
send-Int-Msg	1	
return-table	2	

Conditions:
received-Int-Msg?: does a component different from the source component received an Int-Msg?
component-has-sons?: does the component that received an Int-Msg have sons?
Actions:
interrupt-process: the current database does not send Int-Msgs and stops the process.
return-table: go to the beginning of this table.

5.4 Proof of the Algorithm

To prove the information discovery algorithm we assume that the different groups in a federation are well defined. Suppose a query q' performed in a source component DB_S that participates in federation F'. We want to proof that the data d' for q' is found, using the proposed algorithm, if there is a component DB' inside F', classified in a group G'.

Proof:

Consider DB' a component in F', participating in G' and containing d'. Assume that d' is not found when applying the discovery algorithm (failure of the discovery process).

By the algorithm, the failure of the discovery process occurs whenever all the components of a certain group related to the query, including the master, are visited and none of those visited databases contain the correct data for this query.

Therefore, if d' is not found, then the discovery graph for q' ($DG_{q'}$) does not contain DB' in its set of vertices. The non-existence of DB' in $DG_{q'}$ means that none of the LOC structures of the visited components and the master of G' contain a reference to DB'. However, if the master does not have a notification for DB', then DB' does not belongs to F'. Record that the master of a group always has the most up to date information of its group. Thus, this is a contradiction.

Hence, if d' exists in a component that participates in a group G' related to q', inside the federation that contains the source component for q', then, applying the discovery algorithm, d' is always found. □

5.5 Variation of the Discovery Process

One way of optimising the search process is to allow the databases of a federation learn with the queries. The idea consists in maintaining a special structure named *discovery-history* in each component, related to each federation that it participates. The discovery-history is used to allow fast execution of future queries similar or equal to any query performed in the past.

Whenever a query is performed in a source component it first consults its related discovery-history to see if a similar query has been executed in the past. In the case where similar queries has been performed, the discovery process consults the components associated. If the correct data is not found in these components the LOC structure of the source is traversed and the process is executed normally. In the case where no similar queries are found in the discovery-history the process follows normally.

Every time that a data is found to a query q', the discovery-history of the source component related to q' is updated. The type of data found, together with its position are introduced into the discovery-history. Notice that the discovery-history does not hold a copy of the data (answer) that fulfils q', but a possible place to find this answer. Therefore, we avoid the problem of the cache structures where it is necessary to update different copies of a certain data (record that the process is executed in a dynamic environment). The information inside the discovery-histories of the system can be used in the future to update the LOC structures.

Chapter 6

The SPD

One fundamental question in the different phases of the interoperability process is related to the semantic aspects of the data. These semantic aspects are also important during the information discovery process. In this process it is necessary to identify a component (target) that contains the data to fulfill a certain query q'. Therefore, some databases are visited (possible target components) in order to verify if they contain the required data. These databases are independent created and administered, and contain differences in the way that an object is modelled. Thereby, it is possible to have a query q' formed by certain terms that are not known by other components. Notice that this situation can happen not only when dealing with databases that use different data models. Even when the involved databases have the same data model in their schemas it is possible to have differences of modelling.

We propose the use of the SPD to help on the determination of similarities, equivalences, differences and conflicts during the discovery process. Thus, the LOC structure is used to determine the possible components that may contain the searched data. The SPD is used to assist in the verification of the existence of the searched data in those components.

In many cases, the translations and mappings between the local data models and the canonical data model (or metamodel) are not sufficient to permit the correct interoperability of the components. There are some important semantic aspects that cannot be represented in a data model. Hence, it is necessary to use a special structure containing this information.

The idea is to have a SPD associated with each LPS of a component. The SPD contains semantic information of the data inside the related LPS. It is possible to have differences (conflicts) involving the data. These differences are classified as: naming, structural, representational, different levels of abstractions and schematic discrepancies (see subsubsection 2.1.1). Hence, the SPD has to contain the specifications for all of these differences, particular for its component. We present next how to possible deal with some of these differences using the SPD.

1. Naming

   In the case of synonyms and homonyms we suggest the idea proposed in [45], where there is a set of general terms and the items can be mapped to these terms. Thus, when a query q' is executed in a source component and has to be sent to possible target components, the necessary items are first translated to general terms before

being sent. Thereby, when it arrives in a possible target component those items are retranslated in order to perform the query.

2. Structural and Representational

   These cases are related to the fact of having the same object described and represented in different ways. It is possible to have either a n object represented as an attribute in one data model and as an entity in another or the same attribute defined differently in distinct databases. As recorded in [24] we can have different types of structural conflicts, such as: (1) type mismatch, e.g., an attribute defined as char in one schema and as numeric in another schema; (2) formats, e.g., date represented by (day, month, year) and by (month, day, year); (3) units, e.g., quantity in pounds and in kilograms; and (4) granularity, e.g., the cost of an object with and without V.A.T. included.

   To resolve these conflicts we suggest to have explanation of these attributes in the SPD structure.

The use of this approach has the advantages of preserving the autonomy of the components and of allowing the execution of changes in the system in an easy way. That is, whenever a component modifies its shared data or a database is added or removed from a federation, these facts do not have to be reflected into the other components. Record that each component contains semantic information about its data. However, the idea requires the use and definition of global concepts to map the items. It also requires translations and retranslations of queries, making the execution more complicated. In some cases these translations can be unnecessary; when two involved databases use the same terms to express the same real world object.

Another idea consists in having a SPD for each component in a federation, but containing semantic information related to the shared data of all the other components in the federation. That is, for a certain data, the SPD contains the equivalent specification of this data in all the other components. Thus, to perform the discover of a data, a query is sent in its original form to the possible target components. When achieving a possible target component, the query is translated, with the help of the SPD, depending on the way that this possible target component specifies the terms related to the query. With this approach the autonomy is also preserved, since each component continues to use the data in the way that it was originally modelled. It performs less mappings. Nevertheless, it is difficult to execute changes in the system. Whenever a component is added or removed from a federation, this has to be updated in all other SPDs of the components. Therefore, we believe that this approach is not suitable for our case.

More work has to be done in order to solve the semantic problem and help in the information discovery process. Apart from the naming and structural and representational aspects, we need to analyse the other possible cases that can occur. It is necessary to specify a way of dealing with all of the possible different cases.

Chapter 7

A Case Study

We present below a case study to illustrate the propose architecture, the information discovery process, and the addition and deletion procedures.

We present below a simple example to illustrate the proposed architecture. Consider the following scenario involving *book-stores, chemists, hospitals, universities* and *credit-card* companies. Suppose that a company named "Cia-Shops" owns two different book-stores and two chemists in different parts of the world. Each of these shops has its own database independently created and administered. To facilitate the control of the company, the owner of the company wants to have an overall view of the accounts without creating a single central database with that information.

Cia-Shops and other book-stores and chemists decide to build a "pool of shopping" (network) to improve the sales in the shops and to facilitate the life of the customers. The aim is to permit people from different universities and hospitals (lecturers, professors, students, doctors, nurses, employees, and others) to buy the products in the pool via the network. Payment is made either by credit-card or directly discounting from the payroll of the staff.

In this scenario, it is important to guarantee privacy of information. That is, a customer willing to buy a certain product in one shop is not permitted to access special information such as accountancy, salary of employees in the shop, stock of the shop, special clients, and so on. It is also desired to permit interoperability among the different databases in the pool without affecting their autonomy. This interoperability has to be performed in a transparent way, guaranteeing scalability of the system and good performance.

The databases are organised in two different federations in order to guarantee privacy and confidentiality. Figure 7.1 presents federation 1 formed by the databases of the book-stores and chemists of the Cia-Shops, interoperating in order to control the accounts,

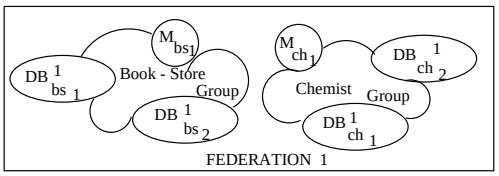

Figure 7.1: Federation of the databases in Cia-Shops

profits and expenses of the company. Federation 2, in figure 7.2, contains the "pool of shopping" with the databases of the book-stores, chemists, hospitals, universities and credit-card companies.

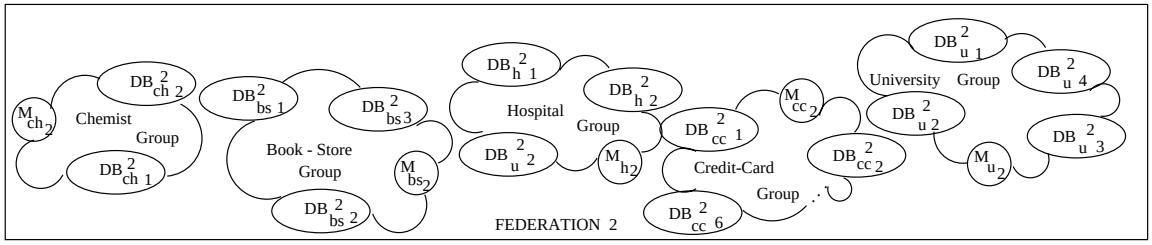

Figure 7.2: Federation of the databases in the pool of shopping

In figures 7.1 and 7.2 we can see the databases of the different book-stores, identified by DB_{bs_v} $(1 \leq v \leq 3)$ and the databases of the chemists, represented by DB_{ch_y} $(1 \leq y \leq 2)$. Similarly, the universities and hospitals are represented as DB_{u_k} $(1 \leq k \leq 4)$ and DB_{h_l} $(1 \leq l \leq 2)$, respectively. DB_{cc_t} $(1 \leq t \leq 6)$ stands for the databases of the credit-card companies. Thus, the databases are represented as $DB^j_{x_i}$, where: $x \in \{bs, cc, ch, h, u\}$; i is an integer expressing a database in any of the different existing type of groups; and $j = \{1, 2\}$, represents the two federations of the example. Notice that there are some databases that appear in both federations, but sharing different information in each. Inside the federations the databases are classified (organised) in different groups in order to facilitate the information discovery process. In federation 2, DB_{u_2} belongs to two different groups (university and hospital groups), since it is a hospital-university.

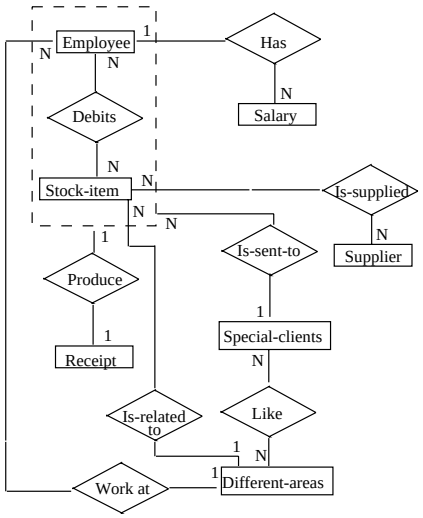

Figure 7.3: Part of the conceptual schema of DB_{bs_1} - LCS

7.1 The Architecture

We select two different components that participate in the system to demonstrate some aspects of the architecture. Suppose that federation 1 and federation 2 use the rela-

tional and object-oriented data models as the canonical data model, respectively. Consider DB_{bs_1} (book-store database) representing its conceptual schema in the E-R model, with part of its LCS presented in figure 7.3. On the other hand, suppose DB_{bs_2} a relational data model with part of its LCS shown in figure 7.4, where the keys are represented by underlined attributes. Both of these databases participate in federation 1 and federation 2. Thus, each one contains two groups of additional structures ($LPS_i^j, MLPS_i^j, SPD_i^j, LOC_i^j, i, j = \{1, 2\}$). Figure 7.5 illustrates DB_{bs_1} and DB_{bs_2} with their additional structures. We present next some of these structures for DB_{bs_1} and DB_{bs_2}.

Employees (emp-id, name-emp, addr-emp, birth-date, start-date, ..., area-id)

Request (req-id, date, descr, emp-id, clie-id)

Receipt (rec-id, req-id, date, ...)

Special-client (clie-id, name-cl, addr-cl,, ...)

Stock-item (item-num, descr, qty, price, min-qty, sup-id, area-id, ...)

Salary-hist (emp-id, date, salary)

Different-areas (area-id, descr, ...)

Supplier (sup-id, name-sup, addr-sup, facilities, ...)

Likes (clie-id, area-id, ...)

Has (req-id, item-num, ...)

Supplies (item-num, sup-id, qty, ...)

Figure 7.4: Part of the conceptual schema of DB_{bs_2} - LCS

In federation 1 the owner of the company wants to have a complete view of his company in order to retain better control. Therefore, the LPS_1^1 and LPS_2^1 structures (local public schema) are equal to the local conceptual schemas of DB_{bs_1} and DB_{bs_2} (figures 7.3 and 7.4, respectively). However, federation 2 deals with the aspect of allowing customers to buy products, having a specific shared data. The LPS_1^2 and LPS_2^2 are presented in figures 7.6 and 7.7, respectively.

We present next the LOC and master structures and the masters of all the involved components. Suppose that the LOC structures of all the components are updated, having the same information inside a certain federation. Thus, figure 7.8 presents the LOC

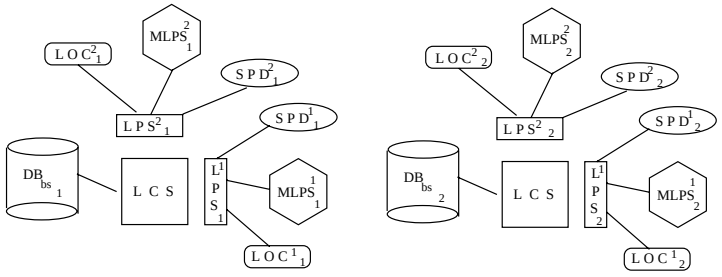

Figure 7.5: DB_{bs_1} and DB_{bs_2} with their additional structures

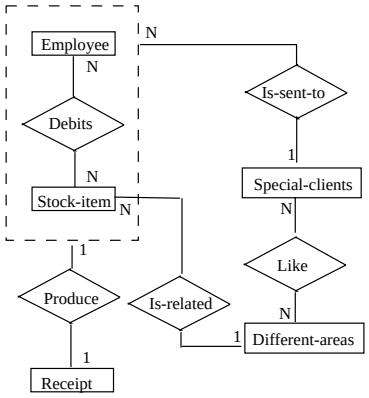

Figure 7.6: Part of the local public schema of DB_{bs_1} in federation 2 - LPS_1^2

Employees (emp-id, name-emp)

Request (req-id, date, descr, emp-id, clie-id)

Receipt (rec-id, req-id, date, ...)

Special-client (clie-id, name-cl, addr-cl,, ...)

Stock-item (item-num, descr, qty, price, min-qty, sarea-id,)

Different-areas (area-id, descr, ...)

Likes (clie-id, area-id)

Has (req-id, item-num)

Figure 7.7: Part of the local public schema of DB_{bs_2} in federation 2 - LPS_2^2

structure for federation 1 and figure 7.9 presents the LOC structure for federation 2. Notice that, although the components of federation 1 are part of federation 2, and some groups have the same names (terms), the part of the tree related to those databases in both federations are different.

The $MLPS_i^1$, $i = \{1, 2\}$, related to DB_{bs_1} and DB_{bs_2} in federation 1, consist in the translation of the LPS_i^1 to the relational data model. On the other hand, $MLPS_i^2$, related to DB_{bs_1} and DB_{bs_2} in federation 2, are the mapped of the LPS_i^2 to an object-oriented data model. The topic related to the translation between different data models has received attention in the literature [22, 54, 55, 89, 107]. Therefore, we do not present here the details of the $MLPS_i^j$, $i, j = \{1, 2\}$. Figure 7.10 presents a part of the Syst-DB, related to the scenario that it is being analysed, with some of its entities, relationships and attributes.

7.2 The Discovery Process

Suppose the situation where a lecturer, an user of database DB_{u_4}, performs a query q' into DB_{u_4}. This query q' is concerned to the price of an art book. Therefore, DB_{u_4} is a university database and does not contain information about books and prices. Thus, it is necessary to identify a federation which DB_{u_4} participates that is related to the query q', In this case it is federation 2. Following, the LOC_4^2 structure is traversed (figure 7.9). DB_{bs_1} is identified as a possible target component and M_{b_2} as the address of the related

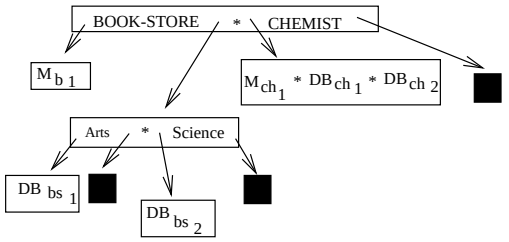

Figure 7.8: LOC and master structures of components in federation 1 - LOC_i^1, M_{x_1}

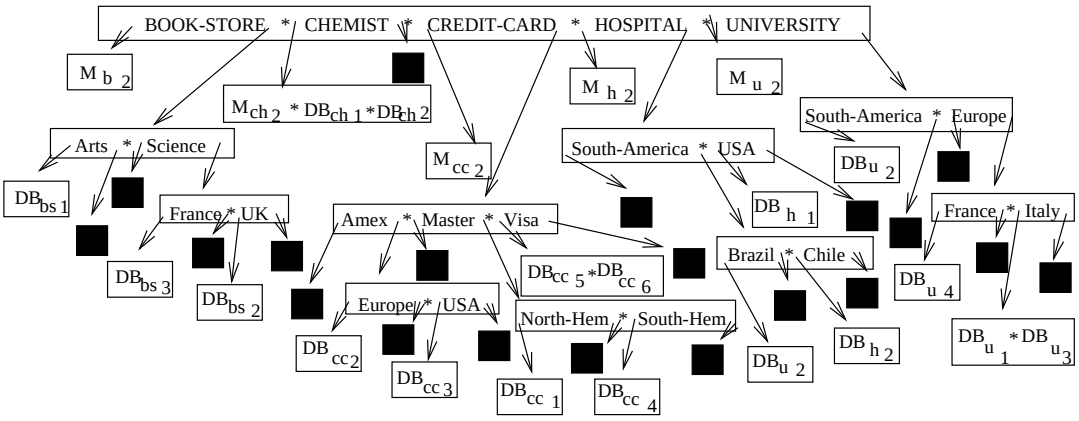

Figure 7.9: LOC and master structures of components in federation 2 - LOC_i^2, M_{x_2}

master. Then, q' and the addresses of DB_{bs_1} and M_{b_2} are sent to DB_{bs_1}. When receiving the request of q', DB_{bs_1} is consulted and the fact that DB_{bs_1} has been consulted.

Assume that DB_{bs_1} does not contain the art book referred in q'. At this point LOC_1^2 of DB_{bs_1} is traversed, but no other possible target component is identified. Then, the master of the book-store group (M_{b_2}) is consulted to identify a possible target component. Consider the case where a new book-store database, named DB_{bs_4}, willing to participate in the pool of shopping, was inserted into the system, but not yet updated into LOC_4^2 and LOC_1^2. Nevertheless, M_{b_2} contains the information about the existence of DB_{bs_4}, as shown in figure 7.11. When M_{b_2} is consulted DB_{bs_4} and DB_{bs_1} are identified as possible target component. Then, the request is sent to DB_{bs_4} and DB_{bs_1}. When receiving the request, DB_{bs_1} ignores it, since it has already been consulted for q' (record that this information is added into DB_{bs_1} after consulting it). On the other hand, when DB_{bs_4} receives the request it is consulted. The discovery graph for q' is presented in figure 7.12. The dash arrow represents that the source component knows about the existence of the master in the graph.

First let us consider the case where DB_{bs_4} contains the data d' for q'. DB_{bs_4} sends d' to DB_{u_4} (the source component). Suppose that DB_{u_4} is satisfied with d' (a correct answer is found). Then, DB_{bs_4} is responsible to cancel the discovery process in execution by sending the Int-Msg to DB_{bs_1} and M_{b_2}, at the same time. Following, this components are responsible to send Int-Msgs to their sons, and so on. The discovery-history (see subsection 5.4) is updated notifying the existence of d' in DB_{bs_4}.

On the other hand, assume that DB_{bs_4} does not contain the answer d' for q' and

Federations

id-fed	manager	cdm	cdl	type-fed
F1	Peter	Relational	SQL	Cia-Shops
F2	Mary	obj-orien.	OSQL	Pool of Shop

Groups

id-group	name-gr
G1	book-store
G2	chemist
G3	credit-card
G4	hospital
G5	university

is-composed

id-fed	id-group
F1	G1
F1	G2
F2	G1
F2	G2
F2	G3
F2	G4
F2	G5

Have

id-fed	id-group	id-comp
F1	G1	DB_{bs1}
F1	G1	DB_{bs2}
F1	G2	DB_{ch1}
F1	G2	DB_{ch2}
F2	G1	DB_{bs1}
F2	G1	DB_{bs2}
F2	G2	DB_{ch1}
F2	G2	DB_{ch2}
F2	G3	DB_{cc1}
F2	G3	DB_{cc2}
F2	G3	DB_{cc3}
F2	G3	DB_{cc4}
⋮		

Components

id-group	type-comp	dba
DB_{bs1}	Relational	John
DB_{bs2}	Relational	Peter
DB_{bs3}	obj-orien.	Mary
DB_{ch1}	Hierarchical	Jane
DB_{ch2}	Relational	Junior
DB_{cc1}	Relational	Lewis
DB_{cc2}	obj-orien.	Rob
DB_{cc3}	Network	Tracy
DB_{cc4}	obj-orien.	John
DB_{cc5}	Functional	Francys
DB_{cc6}	Relational	Paul
DB_{h1}	obj-orien.	Phelip
DB_{h2}	obj-orien.	Alan
DB_{u1}	Relational	Richard
DB_{u2}	Network	Luise
DB_{u3}	Functional	William
DB_{u4}	obj-orien.	Carl

Master

id-master	id-fed	id-group
M_{bs1}	F1	G1
M_{ch1}	F1	G2
M_{bs2}	F2	G1
M_{ch2}	F2	G2
M_{cc2}	F2	G3
M_{h2}	F2	G4
M_{u2}	F2	G5

Figure 7.10: A view of the Syst-DB

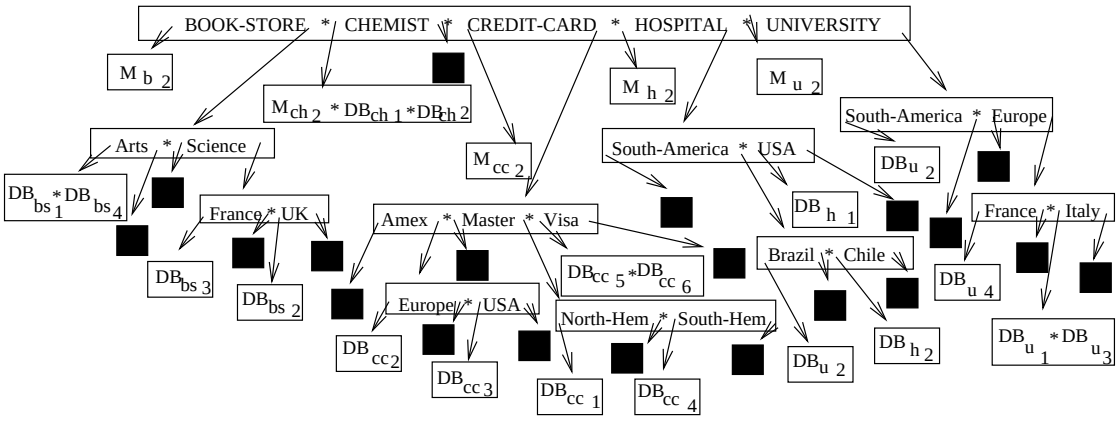

Figure 7.11: M_{b_2} with a new component

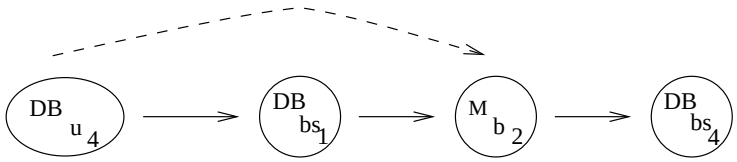

Figure 7.12: Discovery graph for q'

does not know any other possible target component (failure of the process). In this case, DB_{bs_4} notifies M_{b_2} (its parent) about the non-existence of the information. Then, M_{b_2} notifies its own parent (DB_{bs_1}) and so on, until DB_{u_4} receives this information. Notice that it is useless to send the non-existence information from M_{b_2} directly to DB_{u_4} (source component). The failure of the process can only be concluded when the source component receives notification from all of its sons.

7.3 The Scalability Process

Here we do not present all the cases related to the addition, deletion and modification in the system (see chapter 4). For simplicity, we exhibit one example related to the addition of a new component and another one concerned to the deletion of an existing database.

7.3.1 Addition

Consider a university U_5 willing to share and exchange information with other universities related to development of projects. To perform this task DBA of U_5 consults the Syst-DB and discover four different university databases that participate in the system in federation 2. However, it also realises that the type of information that they share and exchange are not related to the projects (federation 2 deals with the pool of shopping). Following, the DBA of U_5 (DBA_5 for short) contacts DBA_1, DBA_2, DBA_3 and DBA_4 in order to verify the possibility of forming a federation with some (or all) of these databases. Assume that U_2, U_3 and U_4 are interested in sharing information related to projects with U_5, composing a new federation (federation 3, for instance).

One of the $DBAs_i$ ($2 \leq i \leq 4$) is elected the manager of federation 3. This manager is responsible to decide how to organise those databases into the federation, to create the terms related to the groups, to build the LOC and master structures, to define the common data model and data language, and so on. At the same time, each DBA of the involved components is responsible to prepare its database to participate in federation 3, by specifying the additional structures. When all the involved components are prepared, Syst-DB is updated with the necessary information related to federation 3 and the databases can start to interoperate. Notice that the creation of this new federation is performed by the reflection of the insertion of a new component (DB_{u_5}, for instance) and it is executed without stopping the system. Figure 7.13 presents federation 3 and figure 7.14 exhibits the LOC and master structures for the components in federation 3.

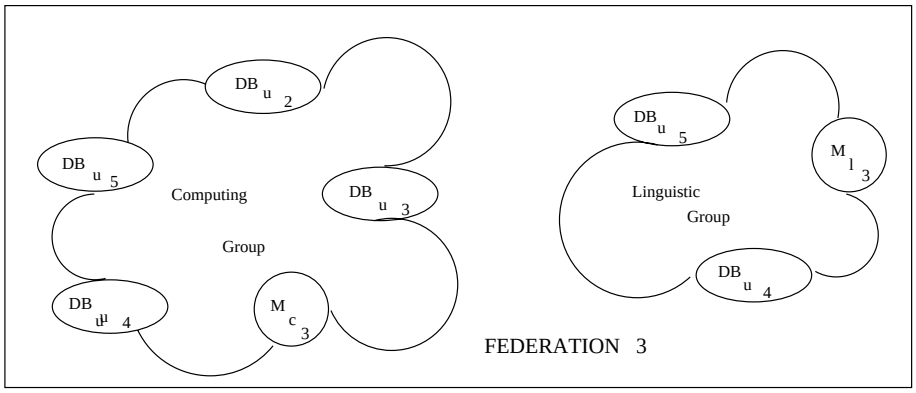

Figure 7.13: Federation of the university databases sharing projects

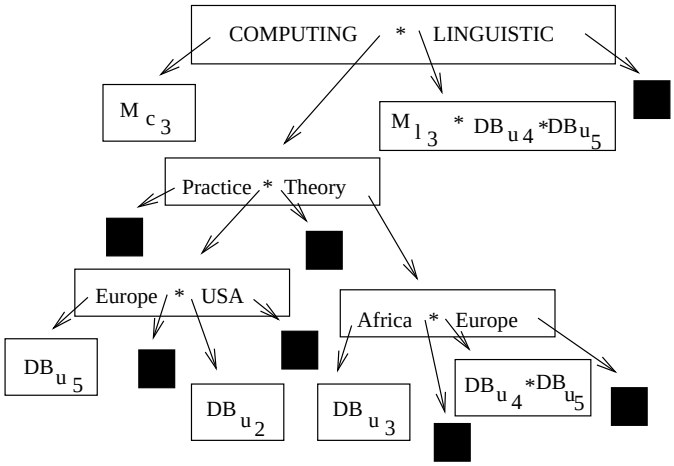

Figure 7.14: LOC and master structures for federation 3

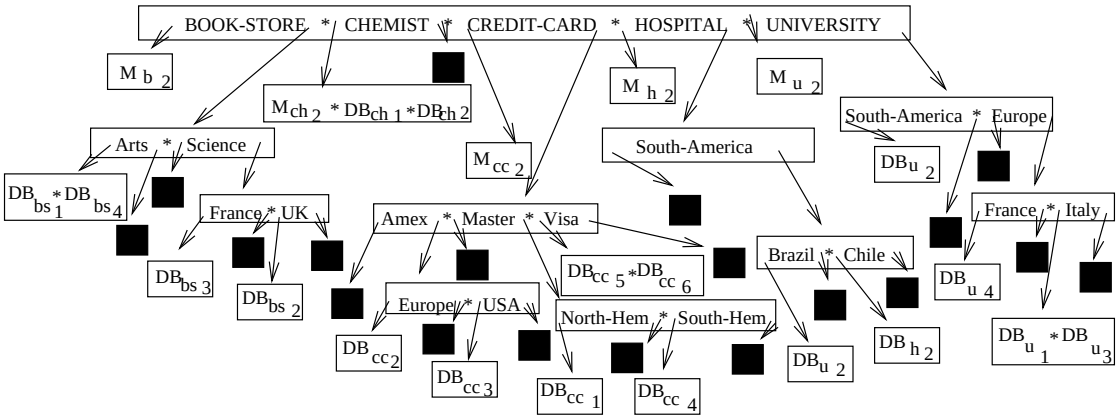

Figure 7.15: New master for M_{h_2} after removing DB_{h_1} from federation 2

7.3.2 Deletion

Assume that DB_{h_1} wishes to suspend to share and exchange information in federation 2. Thus, the manager of federation 2 is informed about this fact. S/He has to identify the groups in federation 2 where DB_{h_1} participates (hospital group, in this case), to update the Syst-DB about the removal of DB_{h_1} from federation 2 and to insert the "special message" into the address of DB_{h_1}. Following the master of the hospital group (M_{h_2}) is updated with the structure presented in figure 7.15. Then, M_{h_2} is responsible to send messages to the other masters in federation 2 updating them. Those masters are responsible to update the components related to their groups. Notice that the removal of DB_{h_1} from federation 2 does not generate the deletion of the hospital group.

Chapter 8

Conclusion and Further Work

In this report we proposed an approach to permit interoperability among autonomous heterogeneous databases. This approach allows the participation of databases that share and exchange data, and the participation of databases which interrogates the other components. We presented an architecture that organises different databases into federations, depending on the data to be shared and on the other components with which they want to share this data. The use of federations aims to guarantee privacy and confidentiality of the shared data. Inside a federation the databases are classified in different groups and subgroups, according to the type of shared data, in order to reduce the universe of search. Before joining a federation a database needs to be prepared and new structures are added to each component. These additional structures provides the information necessary to achieve interoperability and preserve autonomy. The architecture supports participation by both enquirer databases and components that share and exchange information. It permits semantic representation of the shared data of each component in each federation.

We also presented an algorithm to perform the information discovery process in a distributed way, without broadcasting to all components. This process is executed with the help of a hierarchical structure containing specialised terms and components related to these terms. Nevertheless, the correctness of the information discovery process is directly related to the way that the groups are formed in a federation and to the choice of specialised terms. That is, if the groups are well defined, then the discovery process can either find the data or conclude that it does not exist in the system. To improve the discovery process it is important to specify a large variation of groups inside a federation. However, these groups should contain a low number of components associated to them.

We claim that the approach supports scalability of the system, privacy and confidentiality of the data, preserves the autonomy of the databases and performs interoperability in a transparent way.

More work has to be done concerning the problem of how to specify and organise the different groups in a federation. It is necessary to evaluate and analyse what are the problems that can appeared when having a large number of federations, and how to control the participation of a component in many federations. We have to find a way to represent and specify the semantic information of the shared data necessary to perform the information discovery process. It is essential to solve the problem of failures related to the master and Syst-DB structures.

Bibliography

[1] R. Ahmed, P. De Smedt, W. Du, W. Kent, M.A. Ketabchi, W. A. Litwin, A. Rafii, and M. Shan. The Pegasus heterogeneous multidatabase system. *Computer*, 24(12):19–27, December 1991.

[2] R. Alonso and D. Barbara. Negotiating data access in federated database systems. In 5^{th} *Conference on Data Engineering*, pages 56–65, Los Angeles, February 1989. IEEE.

[3] R. Alonso, D. Barbara, and S. Cohn. Data sharing in a large heterogeneous environment. In 7^{th} *International Conference on Data Engineering*, pages 305–313, Held Kobe, April 1991. IEEE.

[4] P. Atzeni and R. Torlone. A metamodel approach for the management of multiple models and the translation of schemes. *Information Systems*, 18(6):349–362, 1993.

[5] C. Batini and M. Lenzerini. A methodology for data schema integration in the entity relationship model. *IEEE Transaction on Software Engineering*, SE-10(6):650–664, November 1984.

[6] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[7] A. Bouguettaya. *A Dynamic Framework for Interoperability in Large Multidatabases*. PhD thesis, Faculty of Graduate School of the University of Colorado, 1992.

[8] A. Bouguettaya and R. King. Large multidatabases: Issues and directions. In D.K. Hsiao, E.J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems (DS-5)*, pages 55–68. Elsevier Science Publisher B.V., 1993.

[9] A. Bouguettaya, R. King, D. Galligan, and J. Simmons. Implementation of interoperability in large multidatabases. In *The Third International Workshop on Research Issues on Data Engineering: Interoperability in Multidatabase Systems*, Vienna, Austria, April 1993.

[10] A. Bouguettaya, S. Milliner, and R. King. Resource location in large scale heterogeneous and autonomous databases. Submitted for publication.

[11] Y. Breitbart, P.L. Olson, and G.R. Thompson. Database integration in a distributed heterogeneous database system. In *International Conference on Data Engineering*, pages 301–310, Los Angeles, California, February 1986. IEEE Computer Society.

[12] Y. Breitbart, A. Silberschatz, and G. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of ACM SIGMOD Conference*, 1990.

[13] Y. Breitbart. Multidatabase interoperability. *SIGMOD RECORD*, 19(3):53–60, September 1990.

[14] M.W. Bright, A.R. Hurson, and S.H. Pakzad. A taxonomy and current issues in multidatabase systems. *Computer*, pages 50–60, March 1992.

[15] M. W. Bright and A. R. Hurson. Linguistic support for semantic identification and interpretation in multidatabases. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 306–313, Los Alamitos, California, 1991. IEEE Computer Society Press.

[16] M. W. Bright, A. R. Hurson, and S. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Transaction on Database Systems*, 19(12):212–253, June 1994.

[17] M. L. Brodie, F. Bancilhon, C. Harris, M. Kifer, Y. Masunaga, E. D. Sacerdoti, and K. tanaka. Next generation database management system technology. In J-M Nicolas W. Kim and S. Nishio, editors, *Deductive and Object-Oriented Databases*. Elsevier Science Publishers, 1990.

[18] M. Brodie and M. Hornick. An interoperability development environment for intelligent information systems. In *Proceedings of the International Workshop on the Development of Intelligent Information Systems*, Niagara-on-the-Lake, April 1991.

[19] O. Bukhres, J. Chen, and R. Pezzolli. An InterBase system at BNR. *SIGMOD RECORD*, 22(2):426–429, June 1993.

[20] M. A. Casanova and V. M. P. Vidal. Towards a sound view integration methodology. In *Proceedings of the 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 36–47, Atlanta, Georgia, March 1983.

[21] M. Castellanos, F. Saltor, and M. Garcia-Solaco. A canonical model for the interoperability among object-oriented and relational databases. In U. Dayal M.T. Ozsu and P. Valdueriz, editors, *Distributed Object Management*, pages 309–314. Morgan Kaufmann Publishers, San Mateo, California.

[22] M. Castellanos and F. Saltor. Semantic enrichment of database schemas: An object oriented approach. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 71–78, Kyoto, April 1991. IEEE Computer Society Press.

[23] S. Ceri and G. Pelagatti. *Distributed Databases Principles and Systems*. Computer Science Series, 1984.

[24] A. Chatterjee and A. Segev. Data manipulation in heterogeneous databases. *SIGMOD RECORD*, 20(4):64–68, December 1991.

[25] H.R. Cho, Y.S. Kim, and S. Moon. Development of an autonomous heterogeneous distributed database system: Dhim. *Microprocessing and Microprogramming*, 37(1-5):119–122, January 1993.

[26] C. Chung. DATAPLEX: An access to heterogeneous distributed databases. *Communications of the ACM*, 33(1):70–80, January 1990.

[27] C. Collet, M.N. Huhns, and W.M. Shen. Resource integration using a large knowledge base in Carnot. *Computer*, 24(12):55–62, December 1991.

[28] C. Date. *An Introduction to Database System.* Addison-Wesley, Reading, Mass., 4th edition, 1986.

[29] S. Dao, D. M. Kersey, R. Williamson, S. Goldman, and C. P. Dolan. Smart data dictionary: A knowledge-object-oriented approach for interoperability of heterogeneous information management systems. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 88–91, Kyoto, April 1991. IEEE Computer Society Press.

[30] U. Dayal and H. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Transaction on Software Engineering*, SE-10(6):628–645, November 1984.

[31] L. G. Demichiel. Resolving database incompatibility: An approach to performing relational operations over mismatched domains. *IEEE Transactions on Knowledge and data Engineering*, 1(4):485–493, December 1989.

[32] D. Edmond, M. Papazoglou, and Z. Tari. Using reflection as a means of achieving cooperation.

[33] F. Eliassen and R. Karlsen. Interoperability and object identity. *SIGMOD RECORD*, 20(4):25–29, December 1991.

[34] A. K. Elmagarmid, J. Chen, and O. A. Bukhres. Remote system interfaces: An approach to overcoming the heterogeneity barrier and retaining local autonomy in the integration of heterogeneous systems. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):1–22, 1993.

[35] A. K. Elmagarmid, J. Chen, W. Du, O. Bukhres, and R. Pezzoli. InterBase: An execution environment for global applications over distributed, autonomous, and heterogeneous software systems. Technical Report TR112P, Purdue University, 1994.

[36] D. Fang, J. Hammer, and D. McLeod. A mechanism and experimental system for function-based sharing in federated databases. In E.J. Neuhold D.K. Hsiao and R. sacks davis, editors, *Interoperable Database Systems (DS-5)*, pages 239–253. Elsevier Science Publisher B.V., 1993.

[37] D. Fang, J. Hammer, and D. Mcleod. The identification and resolution of semantic heterogeneity in multidatabase systems. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 136–143, Kyoto, April 1991. IEEE Computer Society Press.

[38] D. Fang, J. Hammer, D. McLeod, and A. Si. Remote-Exchange: An approach to controlled sharing among autonomous, hetreogeneous database systems. IEEE Spring Compcon, pages 510–515, San Francisco, February 1991.

[39] P. Fankhauser and E.J. Neuhold. Knowledge based integration of heterogeneous databases. In D.K. Hsiao, E.J. Neuhold, and R. Sacks-Davis, editors, *Interoperable Database Systems (DS-5)*, pages 155–175. Elsevier Science Publisher B.V., 1993.

[40] C. Francalanci and B. Pernici. View integration: A survey of current developments. Technical Report 93-053, Departamento di Eletronica e Informazione - Politecnico di Milano, 1993.

[41] Y. Freundlich. Knowledge bases and databases. *IEEE Computer*, 23(11):51–57, November 1990.

[42] D. Gangopadhyay and T. Barsalou. On the semantic equivalence of heterogeneous representations in multimodel multidatabase systems. *SIGMOD RECORD*, 20(4):35–39, December 1991.

[43] H. Garcia-Molina nad b. Lindsay. Research directions for distributed databases. *SIGMOD RECORD*, 19(4):98–103, December 1990.

[44] M. Hammer and D. McLeod. On database management system architecture. *MIT Lab for Comp. Sc.*, (MIT/LCS/TM-141), October 1979.

[45] J. Hammer and D. McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.

[46] J. Hammer, D. McLeod, and A. Si. An intelligent system for identifying and integrating non-local objects in federated database systems. Technical report, Computer Science Department, University of Southern California, 1994.

[47] S. Heiler, M. Siegel, and S. Zdonik. Heterogeneous information systems: Understanding integration. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 14–21, Kyoto-Japan, April 1991. IEEE Computer Society Press.

[48] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transaction on Office Information Systems*, 3(3):253–278, July 1985.

[49] A. Herbert. Databases in distributed systems: The new frontier. In *4th International Conference on Extending Database Technology*, Cambridge, 1994.

[50] D. K. Hsiao and M. N. Kamel. The multimodel, multilingual approach to interoperability of multidatabase systems. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 208–211, Kyoto, April 1991. IEEE Computer Society Press.

[51] D. K. Hsiao. Federated databases and systems: Part i - a tutorial on their data sharing. *VLDB Journal*, 1:127–179, 1992.

[52] C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous, distributed environments: A metadatabase approach. *IEEE Transactions on Software Engineering*, 17(6):604–625, June 1991.

[53] R. Hull and R. King. Semantic database modeling: Survey, applications, and resource issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[54] P. Johannesson. Schema transformation as an aid in view integration. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Advanced Information Systems Engineering - 5th International Conference, CAISE'93*, number 685 in Lecture Notes in Computer Science, pages 71–92. Springer Verlag, June 1993.

[55] P. Johannesson. A method for transforming relational schemas into conceptual schemas. In *10th International Conference on Data Engineering*, pages 190–201, Houston Texas, February 1994. IEEE Computer Society.

[56] M. N. Kamel and N. Kamel. Federated database management system: Requirements, issues ans solutions. *Computer Communications*, 15(4):270–280, May 1992.

[57] W. Kent. The breakdown of the information model in multi-database systems. *SIGMOD RECORD*, 20(4):10–15, December 1991.

[58] W. Kim, N. Ballou, J. F. Garza, and D. Woelk. A distributed object-oriented database system supporting shared and private databases. *ACM Transactions on Information Systems*, 9(1):31–51, January 1991.

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

[60] Y. S. Kim and S. C. Moon. Update synchronization pursuing site autonomy in hetreogeneous distributed databases. *Microprocessing and Microprogramming*, 34:41–44, 1992.

[61] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In J. Clifford and R. King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management Data*, volume 20, pages 40–49, Denver, Colorado, May 1991. SIGMOD RECORD.

[62] J. A. Larson, S. B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Transaction on Software Engineering*, 15(4):449–463, April 1989.

[63] S. C. Laufmann. Coarse-grained distributed agents for transparent access to remote information. In M.P. Papazoglou and J. Zeleznikow, editors, *The Next Generation of Information Systems: From Data to Knowledge*, number 611 in Lecture Notes in AI, pages 223–237. Springer Verlag.

[64] Y. Lee and S. Moon. Heterogeneous schema integration method for multidatabase system. *Micropeocessing and Microprogramming*, 38(1-5):265–272, September 1993.

[65] Q. Li and D. Mcleod. An object-oriented approach to federated databases. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 64–70, Kyoto, April 1991. IEEE Computer Society Press.

[66] Q. Li and D. McLeod. Managing interdependencies among objects in federated databases. In E.J. Neuhold D.K. Hsiao and R. sacks davis, editors, *Interoperable Database Systems (DS-5)*, pages 331–347. Elsevier Science Publisher B.V., 1993.

[67] Y. E. Lien and J. H. Ying. Design of a distributed entity-relationship database system. In *Computer Software And Applications Conference*, pages 277–282, November 1978.

[68] W. Litwin and A. Abdellatif. Multidatabase interoperability. *Computer*, 19(12):10–18, December 1986.

[69] W. Litwin. From database systems to multidatabase systems: Why and how. In W.A. Gray, editor, *Proceedings of The Sixth British National Conference on Databases (BNCOD 6)*, British Computer Society Workshop Series, pages 161–188, July 1988.

[70] W. Litwin, M. Ketabchi, and R. Krishnamurthy. First order normal form for relational databases and multidatabases. *SIGMOD RECORD*, 20(4):74–76, December 1991.

[71] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

[72] P. Lyngbaek and D. McLeod. Object management in distributed information systems. *ACM Transaction on Office Information Systems*, 2(2):96–122, April 1984.

[73] F. Manola. Object-oriented knowledge bases. *AI Expert*, (5):26–36, March 1990.

[74] S. Milliner and M. Papazoglou. A scalable architecture for interactions between autonomous distributed database nodes. Submitted for publication.

[75] S. Milliner and M. Papazoglou. Reassessing the roles of negotiation and contracting for interoperable dtabases. *Int'l Workshop on Advances in Databases and Information Systems, Russian ACM SIGMOD*, May 1994.

[76] I-Min, A. Chen, and D. McLeod. Derived data update in semantic databases. In *Proceedings of the Fifth International Conference on Very Large Data Bases*, pages 225–234, Amsterdan, 19989.

[77] A. Motro and P. Buneman. Constructing superviews. In ACM SIGMOD, editor, *International Conference on management of Data*, pages 56–64, April-May 1981.

[78] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Transactions on Software Engineering*, SE-13(7):785–798, July 1987.

[79] J. G. Mullen, W. Kim, and J. Sharif-Askary. On the impossibility of atomic commitment in multidatabase systems. Technical Report TR113P, Purdue University, 1994.

[80] S. Navathe, R. Elmasri, and J. Larson. Integrating user views in database design. *Computer*, 19(1):50–62, January 1986.

[81] M. K. Papazoglou, S. C. Laufmann, and T. K. Sellis. An organizational framework for cooperating intelligent information systems. *International Journal of Intelligent and Cooperative Information Systems*, 1(1):169–202, 1992.

[82] M. P. Papazoglou, N. Russel, and D. Edmond. Database homogenization using an intermediate meta-model. Submitted for publication.

[83] W. D. Potter and R. P. Trueblood. Traditional, semantic, and hyper-semantic approaches to data modelling. *IEEE Computer*, 21(6):53–63, June 1988.

[84] S. Ram. Heterogeneous distributed database systems. *Computer*, 24(12):7–10, December 1991.

[85] J.B. Rothnie and N. Goodman. A survey of research and development in distributed database management. In *Very large Data Bases*, pages 48–62, Tokio-Japan, October 1977.

[86] J.B. Rothnie JR.and P.A. Bernstein, S. Fox, N. Goodman, M. Hammer, T.A. Landers, C. Reeve, D.W. Shipman, and E. Wong. Introduction to a system for distributed databases (sdd-1). *ACM Transaction on Database Systems*, 5(1):1–17, March 1980.

[87] M. Rusinkiewicz and A. Sheth. Specifying interdatabase dependencies in a multidatabase environment. *Computer*, 24(12):46–53, December 1991.

[88] F. Saltor, M. Castellanos, and M. Garcia-Solaco. Suitability of data models as canonical models for federated databases. *SIGMOD RECORD*, 20(4):44–48, December 1991.

[89] F. Saltor, M. G. Castellanos, and M. Garcia-Solaco. Overcoming schematic discrepancies in interoperable databases. In E.J. Neuhold D.K. Hsiao and R. sacks davis, editors, *Interoperable Database Systems (DS-5)*, pages 191–205. Elsevier Science Publisher B.V., 1993.

[90] P. Scheuermann, C. Yu, A. Elmagarmid, H. Garcia-Molina, F. Manola, D. McLeod, A. Rosenthal, and M. Templeton. Report on the workshop on heterogeneous database systems held. *SIGMOD RECORD*, 19(4):23–31, December 1990.

[91] A.P. Sheth and J.A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[92] A. P. Sheth. Semantic issues in multidatabase systems. *SIGMOD RECORD*, 20(4):5–9, December 1991.

[93] A. P. Sheth, S. K. gala, and S. B. Navathe. On automatic reasoning for schema integration. *International Journal of Intelligent and Cooperative Information System*, 2(1):23–50, 1993.

[94] A. Sheth and V. Kashyap. So far (schematically) yet so near (semantically). In E.J. Neuhold D.K. Hsiao and R. sacks davis, editors, *Interoperable Database Systems (DS-5)*, pages 283–312. Elsevier Science Publisher B.V., 1993.

[95] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *SIGMOD RECORD*, 199(4):6–22, December 1990.

[96] P. Simpson and R. Alonso. A model for information exchange among autonomous databases. Technical report, Princeton University, May 1989.

[97] J.M. Smith, P.A. Bernstein, U. Dayal, N. Goodman, T. Landers, K.W.T. Lin, and E. Wong. Multibase - integrating heterogeneous distributed database systems. In *National Computer Conference*, volume 50 of *AFIPS Conference Proceedings*, pages 487–499, 1981.

[98] N. Soparkar, H.F. Korth, and A. Silberschatz. Failure-resilient transaction management in multidatabases. *Computer*, 24(12):28–36, December 1991.

[99] S. Spaccapietra and C. Parent. Conflicts and correspondence assertions in interoperable databases. *SIGMOD RECORD*, 20(4):49–54, December 1991.

[100] S. Spaccapietra, C. Parent, and Y. Dupont. Model independent assertions for intregration of heterogeneous schemas. *VLDB Journal*, 1(1):81–126, 1992.

[101] S. Spaccapietra and C. Parent. View integration: A step forward in solving structural conflicts. *IEEE Transactions on Knowledge and Data Engineering*, 6(2):258–274, April 1994.

[102] M. Takizawa, M. Hasegawa, and S. M. Deen. Interoperability of distributed information system. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 239–242, Kyoto, April 1991. IEEE Computer Society Press.

[103] Z. Tari. Interoperability between database models. In E.J. Neuhold D.K. Hsiao and R. sacks davis, editors, *Interoperable Database Systems (DS-5)*, pages 101–118. Elsevier Science Publisher B.V., 1993.

[104] G. Thomas, G.R. Thompson, C. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, September 1990.

[105] H. R. Tirri, J. Srinivasan, and B. Bhargava. Integrating distributed data sources using federated objects. In U. Dayal M.T. Ozsu and P. Valdueriz, editors, *Distributed Object Management*, pages 315–328. Morgan Kaufmann Publishers, San Mateo, California.

[106] D. Tsichritzis. Integrating data base and message systems. In *Very large Data Bases*, pages 356–362, Cannes- France, September 1981.

[107] D. Tsichritzis and F. Lochovsky. *Data Models*, chapter 14. Prentice-Hall, Englewood Cliffs, N.J.

[108] S. D. Urban and J. Wu. Resolving semantic heterogeneity through the explicit representation of data model. *SIGMOD RECORD*, 20(4):55–58, December 1991.

[109] S. D. Urban. A semantic framework for heterogeneous database environments. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 156–163, Kyoto-Japan, April 1991. IEEE Computer Society Press.

[110] D. Weishar and L. Kerschberg. Data/knowledge packets as a means of supporting semantic heterogeneity in multidatabase systems. *SIGMOD RECORD*, 20(4):69–73, December 1991.

[111] D. Weishar and L. Kerschberg. An intelligent heterogeneous autonomous database architecture for semantic heterogeneity support. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 152–155, Kyoto, April 1991. IEEE Computer Society Press.

[112] R. Williams, D. Daniels, L. Haas, G. Lapis, B. Lindsay, P. Ng, R. Obermarck, P. Selinger, A. Walker, P. Wilms, and R. Yost. *R^*: An Overview of the Architecture*, chapter three, pages 196–218. Readings in Database Systems.

[113] S. B. Yao, V. E. Waddle, and B. C. Housel. View modeling and integration using the fucntional data model. *IEEE Transactions on Software Engineering*, SE-8(6):544–553, November 1982.

[114] C. Yu, W. Sun, S. Dao, and D. Keirsey. Determining relationships among attributes for interoperability of multi-database systems. In *First International Workshop on Interoperability in Multidatabase Systems*, pages 251–257, Kyoto-Japan, April 1991. IEEE Computer Society Press.

[115] C. Yu, B. Jia, W. Sun, and S. Dao. Determining relationships among names in heterogeneous databases. *SIGMOD RECORD*, 20(4):79–80, December 1991.

[116] S.L. Pollack, H.J. Hicks, and W.J. Harrison. *Decision Tables: Theory and Practice*. John Wiley and Sons, New York, 1971.